Sound Particles GUI Tester

Mestrado em Engenharia Informática – Computação Móvel

Ricardo dos Santos Silva Marques Maltez

Leiria, novembro de 2020

Sound Particles GUI Tester

Mestrado em Engenharia Informática – Computação Móvel

Ricardo dos Santos Silva Marques Maltez

Estágio realizado sob a orientação do Professor Doutor Nuno Carlos Sousa Rodrigues

Leiria, novembro de 2020

Originalidade e Direitos de Autor

O presente relatório de estágio é original, elaborado unicamente para este fim, tendo sido devidamente citados todos os autores cujos estudos e publicações contribuíram para o elaborar.

Reproduções parciais deste documento serão autorizadas na condição de que seja mencionado o Autor e feita referência ao ciclo de estudos no âmbito do qual o mesmo foi realizado, a saber, mestrado em Engenharia Informática – Computação Móvel, no ano letivo 2019/2020 da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria, Portugal, e, bem assim, à data das provas públicas que visaram a avaliação destes trabalhos (se aplicável).

Agradecimentos

Gostaria de agradecer a todos os membros da equipa da Sound Particles, pela excelente integração na equipa e pela partilha de conhecimentos que contribuiu bastante para me tornar um melhor profissional, em especial ao Doutor Nuno Fonseca pela sua supervisão durante a formação em contexto de trabalho. E ainda, agradecer ao orientador do estágio, professor Doutor Nuno Rodrigues, pelo contributo prestado em todas as etapas do mesmo.

Resumo

Este relatório descreve o estágio curricular do mestrado em Engenharia Informática – Computação Móvel na empresa Sound Particles Lda., com a duração de 9 meses. Neste aprendeu-se a utilizar o *software* Sound Particles e desenvolveu-se uma aplicação *Command Line Interface* (CLI) que interage com a sua interface de utilizador (UI - *User Interface*), com recurso à biblioteca SikuliX, de forma a testar o correto funcionamento do Sound Particles. O software desenvolvido foi integrado na *pipeline* de desenvolvimento da empresa, garantindo a atualização do estado de funcionamento do produto Sound Particles, diariamente. No presente documento inclui-se sucintamente o conhecimento adquirido sobre o software Sound Particles, alguma informação sobre bibliotecas ou ferramentas que permitem a interação com a UI, o processo de desenvolvimento do software criado para testar a UI do Sound Particles, o processo de integração deste software na *pipeline* e uma reflexão sobre todo o processo da formação em contexto de trabalho.

Palavras-chave: Sound Particles, UI Testing, SikuliX, Java

Abstract

This report describes the Master of Computer Engineering – Mobile Computing internship in the company Sound Particles Lda., with the duration of 9 months. During this internship some knowledge about how to work with the software Sound Particles was adquired and an CLI application, which interacts with it is UI was developed, using the library SikuliX, in order to check if it is working as it should. The developed software was integrated in the development pipeline used in this company, to make sure that the information about the working state of the product Sound Particles is updated daily. In the present document the knowledge acquired about the Sound Particles software was included succinctly, as well as some information about libraries or tools that allow the interaction with the UI, the development process of the software created to test the UI of the Sound Particles, the integration of that software in the pipeline and a reflection about the internship experience.

Keywords: Sound Particles, UI Testing, SikuliX, Java

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Acrónimos

CEO	*Chief Executive Office*
CLI	*Command Line Interface*
ESTG	Escola Superior de Tecnologia e Gestão de Leiria
DAW	*Digital Audio Workstation*
GUI	*Graphical User Interface*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
JAR	*Java Archive*
LFE	*Low-frequency Effects*
OCR	*Optical Character Recognition*
QA	*Quality Assurance*
UI	*User Interface*
VNC	*Virtual Network Computing*

1. Introdução

O presente documento tem como objetivo reportar o processo de formação em contexto de trabalho do mestrado em Engenharia Informática – Computação Móvel da Escola Superior e Tecnologia e Gestão (ESTG) do Politécnico de Leiria, que ocorreu entre outubro de 2019 e junho de 2020, com um total de 1440 horas. Este decorreu na empresa Sound Particles, Lda. com os seguintes objetivos:

- Adquirir competências básicas na utilização do software Sound Particles;
- Explorar soluções que permitam o teste da *user interface* (UI) de aplicações *desktop;*
- Desenvolver uma solução que permita o teste da UI do software Sound Particles;
- Adquirir conhecimentos básicos sobre o funcionamento de pipelines no desenvolvimento de software;
- Integrar a solução desenvolvida na pipeline usada no desenvolvimento do software Sound Particles.

Para além da presente introdução, este documento é constituído por mais 5 capítulos, o primeiro capítulo, Programa de Estágio, aborda a entidade no qual ocorreu o estágio curricular e as tarefas que fizeram parte deste.

Os seguintes três capítulos abordam cada uma destas tarefas em detalhe. Começando pela familizarização com o software Sound Particles, onde são resumidas as suas funcionalidades e explicando a constituição da sua UI. De seguida são analisadas algumas bibliotecas ou ferramentas que foram encontradas, para realizar, ou auxiliar, o teste da UI de softwares *desktop* como o Sound Particles. Termina com a tarefa de desenvolvimento de um software para testar a UI do Sound Particles e a integração deste software na pipeline de desenvolvimento usada pela empresa.

O último capítulo, Conclusão, apresenta uma análise crítica sobre o estágio curricular, bem como a sua importância na aprendizagem e na consolidação dos conhecimentos adquiridos durante o mestrado.

2. **Programa de Estágio**

Neste capítulo é apresentada uma caracterização da entidade onde a atividade de formação em contexto de trabalho se realizou, bem como as atividades que ocorreram durante esta.

2.1. Caracterização da Entidade de Acolhimento

A entidade de acolhimento onde ocorreu este estágio curricular designa-se "Sound Particles, Lda" e consiste numa sociedade por quotas. A sua sede localiza-se na incubadora D. Dinis, Rua de Carvalha Nº 570 em Leiria. Consiste numa *startup* com catorze funcionários, entre estes seis programadores, dois *testers*, uma pessoa responsável pelo marketing, uma pelo *design* e outra pela parte administrativa. Existem ainda dois comerciais e, por fim, o *chief executive office* (CEO), Doutor Nuno Fonseca, que fundou a empresa em 2016.

Esta empresa dedica-se exclusivamente ao desenvolvimento de software, com principal foco na área do áudio. Todos os seus clientes são internacionais, sendo que a maioria destes se localiza nos Estados Unidos da América. Nestes estão incluídos alguns estúdios de produção de filmes e áudio bastante famosos, como a Skywaker Sound e Walt Disney Studios.

Atualmente a empresa dispõe de três produtos no mercado:

- O Sound Particles, que consiste no produto principal desta empresa. Sendo que o seu desenvolvimento começou antes da sua fundação. Este software já foi nomeado a vários prémios, como por exemplo, o Outstanding Product pela Audio Society [1].
- Air e Doppler, dois *plugins* para *digital audio workstations* (DAW). Que podem ser executados, por exemplo, no software Reaper [2] ou no Pro Tools [3].

Neste momento a empresa encontra-se a tentar abranger um maior público-alvo, em vez do nicho de mercado onde se tem concentrado. O objetivo é incluir um maior número de *sound designers* amadores em vez de apenas estúdios de cinema profissionais.

Com base na experiência com esta entidade, durante o estágio curricular, foi elaborada a seguinte análise SWOT, que pode ser observada na Tabela 2.1.

Tabela 2.1 – Análise SWOT

Forças	Oportunidades
• Funcionários com boas qualificações • Software inovador • Software usado em grandes empresas da área de filmes e áudio	• Aumento do uso de tecnologias que tirem partido de *surround sound / 3D Audio* (*Virtual Reality* headsets) • Desenvolvimento de um *software* que seja um grande sucesso
Fraquezas	**Ameaças**
• Pouco tempo no mercado • Poucas vendas • Equipa de desenvolvimento pequena comparada com os concorrentes que existem no mercado	• Empresas já estabelecidas no mercado desenvolverem softwares semelhantes

2.2. Tarefas realizadas durante o estágio

A primeira tarefa elaborada durante a atividade de formação em contexto de trabalho consistiu na familiarização com o software Sound Particles. De seguida, foi realizado o levantamento das funcionalidades que poderiam ser testadas pela interface gráfica deste software. Após esta tarefa, iniciou-se uma análise e seleção de bibliotecas/frameworks que permitem a interação com a *graphical user interface* (GUI), tornando possível o desenvolvimento de um protótipo que teste o Sound Particles através da sua GUI, que consistiu na tarefa seguinte. Após o desenvolvimento desse protótipo foi iniciado o desenho e implementação da solução final. Para concluir o estágio, o software desenvolvido foi integrado na pipeline de desenvolvimento da empresa. As datas de início e fim do desenvolvimento de cada uma destas tarefas podem ser observadas na Tabela 2.2.

Tabela 2.2 – Cronograma de tarefas

Tarefa	Data de Início	Data de Fim
Familiarizacão com o software Sound Particles e levantamento de funcionalidades	01/10/2019	18/10/2019
Análise e seleção de bibliotecas/*frameworks* para interação com a GUI	21/10/2019	25/10/2019
Desenvolvimento de protótipo	28/10/2019	22/11/2019
Desenho e implementação da solução final	25/11/2019	13/06/2020
Integração da solução desenvolvida na pipeline	15/06/2020	30/06/2020

3. Familiarização com o software Sound Particles

O processo de familiarização com o software Sound Particles, versão 2.1.0, representado na Figura 3.1, iniciou-se com a equipa de *Quality Assurance* (QA) a demonstrar as funcionalidades deste. Sendo que, ao longo do estágio, o conhecimento sobre este software foi sendo melhorado com base na interação com o mesmo e formações que foram ocorrendo para toda a equipa.

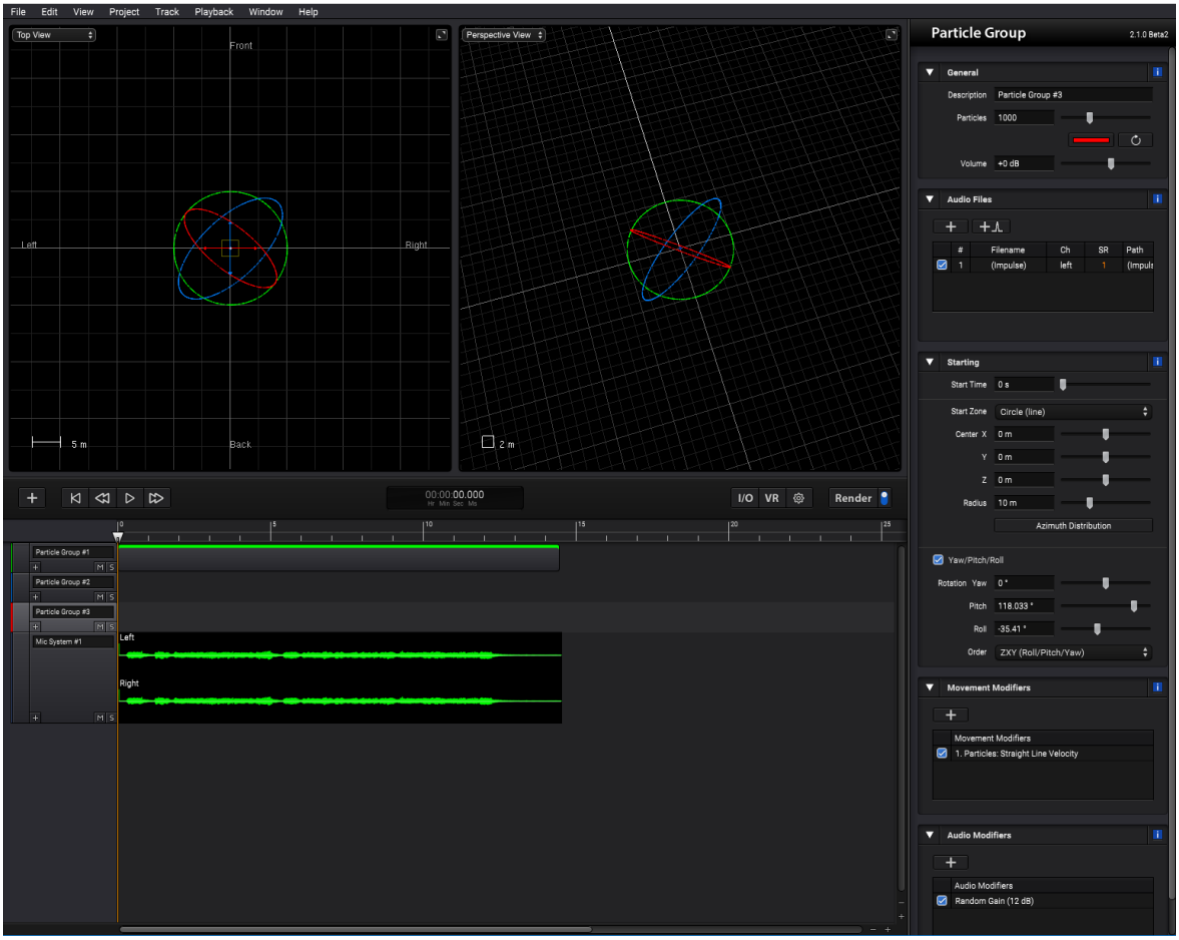

Figura 3.1 - Interface do software Sound Particles

Percebe-se assim que este software se destina a *sound designers*, permitindo a criação de sons tirando partido das propriedades de um mundo virtual 3D. Trata-se de um software inovador no mercado de áudio 3D. Nasceu da ideia de aplicar os sistemas de partículas [4] que são frequentemente utilizados em filmes e jogos (para simular explosões, fogo ou fumo) ao áudio. No caso particular deste software as partículas consistem em emissores de som, que podem ser associados a um ou mais ficheiros de áudio. Permite aplicar efeitos a estas partículas, tanto a nível de posição e rotação, como a nível de edição de som, alterações de volume, *pitch* e equalização, entre outras, de forma a obter os resultados desejados. Permite

ainda a automatização de vários parâmetros de forma a variar os seus valores ao longo do tempo, criando efeitos bastante interessantes. O Sound Particles permite a emissão de som no ambiente 3D a partir de 3 diferentes elementos, sendo eles:

- *Audio Track* – Emissor simples de som que suporta um ficheiro de áudio;
- *Particle Group* – Conjunto de partículas que permite vários ficheiros de áudio;
- *Particle Emitter* – Emissor de partículas que gera partículas a uma certa taxa por segundo durante o tempo definido.

Para captação do som este software possuí sistemas de microfones virtuais com vários tipos de cápsulas e orientações. É possível alterar a configuração da elevação e rotação dos canais de microfones multicanal, bem como o número de canais principais e o número de canais *low-frequency effects* (LFE), de forma a corresponder exatamente ao *setup* de colunas onde o áudio será reproduzido, criando assim uma experiência bastante realista.

A interface deste *software* assemelha-se à de softwares de modelação 3D, como o Blender [5], onde existem objetos num espaço 3D e estes possuem características como a sua posição e rotação, que podem ser alteradas através de campos localizados em menus do lado direito ou esquerdo. O Sound Particles encontra-se disponível para os sistemas operativos Mac OS e Windows. Já foi utilizado em vários filmes de Hollywood e em várias séries televisivas mundialmente conhecidas, como o Game of Thrones.

Nos seguintes subcapítulos são apresentados os diferentes componentes que constituem a interface do Sound Particles.

3.1. Inspector

Um dos elementos gráficos mais importantes deste *software* é intitulado de *Inspector*. Localiza-se do lado direito da interface, inicia-se no canto superior direito e estende-se até ao canto inferior direito. É composto por parâmetros configuráveis pertencentes ao item selecionado atualmente, podendo ser este um *particle emitter*, um *particle group*, uma *audio track* ou um microfone. Estes parâmetros permitem a definição de valores como o número de partículas a serem usadas, o número de emissão de partículas por segundo e o tipo de

microfone, entre outros, como pode ser observado na Figura 3.3, Figura 3.2, Figura 3.4 e Figura 3.5.

Figura 3.3 – Parâmetros de audio track

Figura 3.2 – Parâmetros de microfone

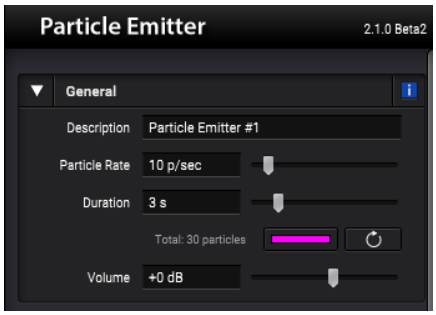

Figura 3.4 – Parâmetros de *particle emitter*

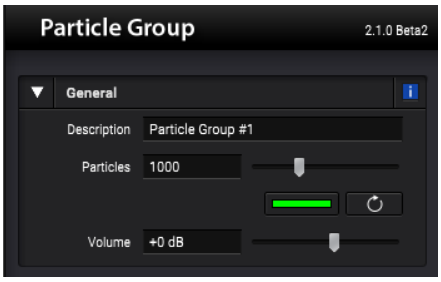

Figura 3.5 – Parâmetros de *particle group*

Além destes parâmetros gerais, existem grupos de parâmetros que são comuns a vários itens, por exemplo os parâmetros de *Start Zone* que definem a posição inicial das partículas, que podem ser observados na Figura 3.6. Estes parâmetros incluem o ponto central e a forma pelo qual as partículas devem ser espalhadas, por exemplo, em forma triangular (Figura 3.7) ou circular (Figura 3.8).

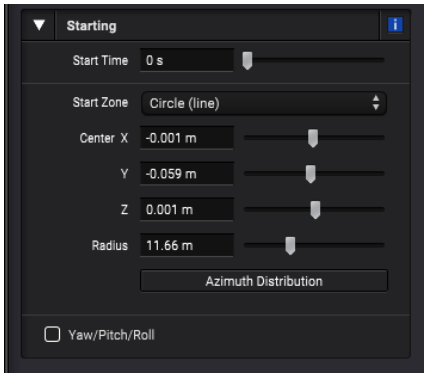

Figura 3.6 – Parâmetros de Start Zone

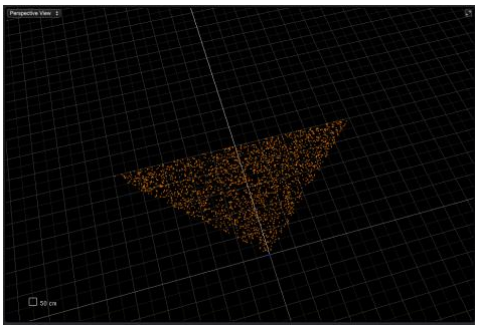

Figura 3.7 – Triangle Start Zone

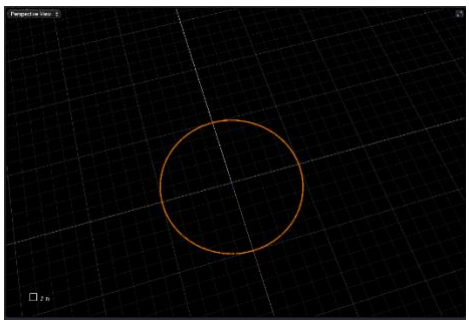

Figura 3.8 – Circle Start Zone

Além de ser possível alterar a localização e forma do grupo partículas, é ainda possível alterar a rotação desta forma, ativando a opção "Yaw/Pitch/Roll", que pode ser observada na Figura 3.6. Ao ativar esta opção são mostrados 3 novos parâmetros, representados pela Figura 3.9, que permitem então alterar a rotação da forma do grupo de partículas. Para um melhor entendimento do significado de cada um destes parâmetros pode ser observada a Figura 3.10 ou consultar [6].

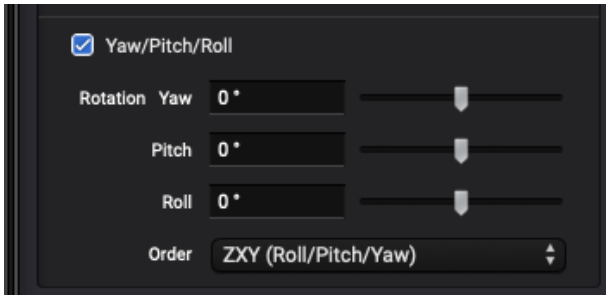

Figura 3.9 – Parâmetros de rotação

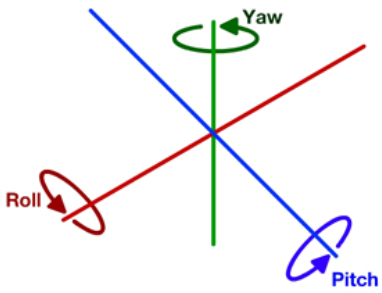

Figura 3.10 – Exemplificação dos parâmetros de rotação

Além dos parâmetros apresentados, o *Inspector* permite ainda adicionar *movement modifiers* e *audio modifiers*, *representados pela* Figura 3.11 *e* Figura 3.12 respetivamente. Os *movement modifiers* dividem-se em duas categorias, *straight line movement* e *rotation movement*, presentes na Figura 3.11. Cada uma destas categorias possui alterações de velocidade e aceleração. No caso dos *audio modifiers* existem 5 categorias: *gain*, *delay*, *equalizer*, *granular* e *time/pitch* com vários parâmetros de configuração, como exemplificado pela Figura 3.12.

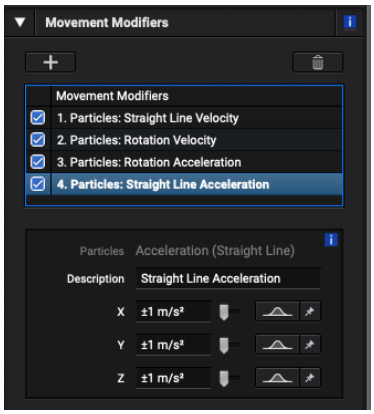

Figura 3.11 – Movement Modifiers

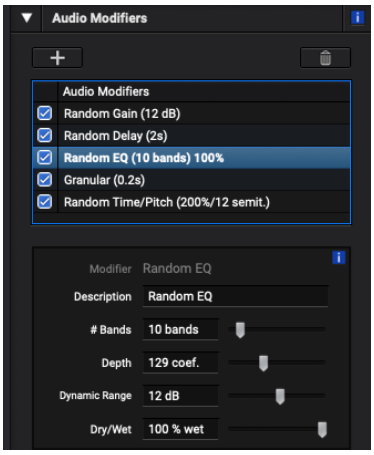

Figura 3.12 – Audio Modifiers

3.2. Random Distributions

Alguns parâmetros presentes neste software têm a capacidade de representar um conjunto de valores em vez de apenas um valor. Sendo este um conjunto de valores gerados aleatoriamente, existe a opção de personalizar, para cada parâmetro, a forma como são gerados os valores aleatórios. O utilizador pode escolher um dos 6 modos disponíveis (*Uniform*, *Normal*, *Triangle*, *Ramp*, *Discrete* e *Custom*) e ajustar algumas opções em cada modo, como pode ser observado na Figura 3.13. Na janela que inclui estas opções também pode ser observado um histograma com a frequência dos valores gerados agrupados por intervalos.

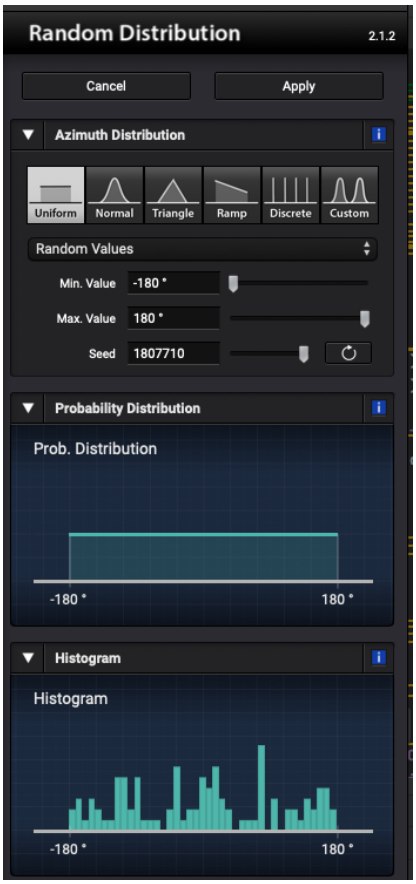

Figura 3.13 – Janela Random Distributions

3.3. 3D View

Na zona central da janela da aplicação localiza-se a *3D View* onde podem ser observadas partículas e objetos 3D importados para o Sound Particles. Este componente possui dois modos, *Single View* e *Dual View*. O primeiro modo apenas apresenta uma vista, o que pode ser útil para o uso em resoluções mais baixas, ao contrário do segundo modo que apresenta duas vistas, exemplificado na Figura 3.14, ideal para observar os objetos 3D de duas maneiras diferentes. Cada vista permite expor os objetos 3D de cinco formas diferentes: topo, frente, esquerda, direita e em perspetiva. Também suporta a vista 360 graus e vista VR (*Virtual Reality*).

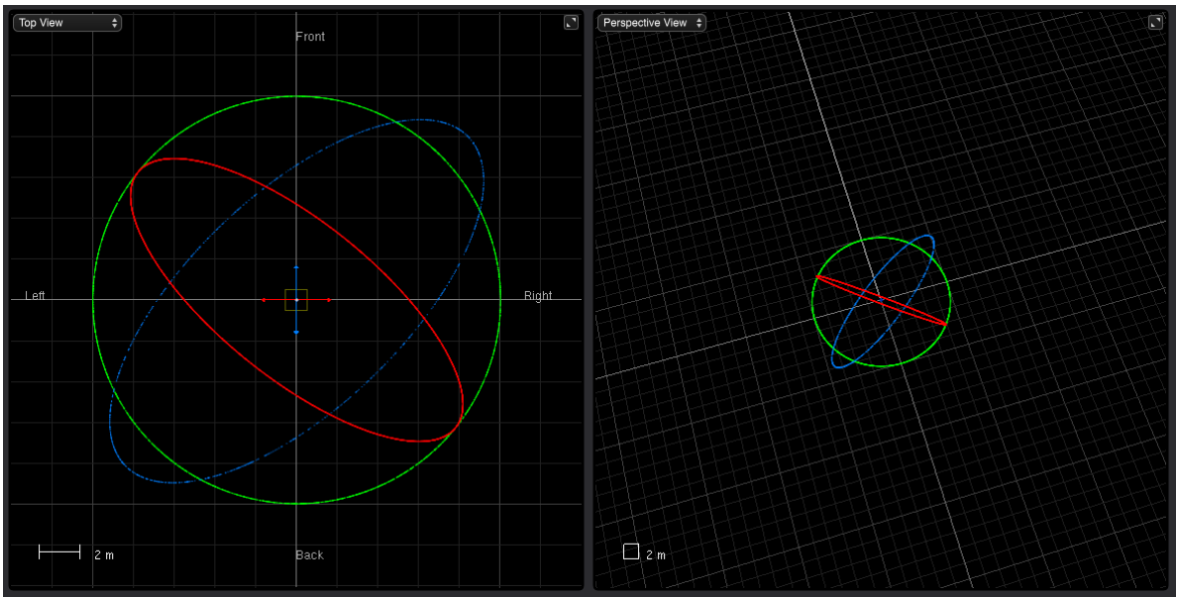

Figura 3.14 – 3D View em modo Dual View

3.4. Transporter

O *Transporter*, representado pela Figura 3.15, consiste no componente responsável por apresentar os itens presentes atualmente em cena, *audio tracks*, *particle groups*, *particle emitters* e microfones. Dentro do *Transporter* existe uma *timeline* onde cada item possui a sua própria *track*, ou seja, um espaço horizontal da *timeline*. Além de apresentar estes itens o *Transporter* também permite adicioná-los ao projeto através do botão "+". Possui ainda a funcionalidade de arrastar *tracks* de forma a alterar o seu *start time*, o que permite ao utilizador atrasar o início de um som.

No *Transporter* é possível aceder a três categorias de configurações: do projeto, de I/O (input/output) e de cliente VR, através dos botões localizados no lado superior direito do *Transporter*, que podem ser observados na Figura 3.15. Neste componente é também disponibilizado um botão que permite o *render* do áudio dos microfones do projeto e, dentro deste botão, existe um *switch* que permite ativar e desativar a funcionalidade de *auto-render*. Esta funcionalidade permite que o utilizador consiga ouvir o áudio captado pelo microfone em cena e, ao mesmo tempo, alterar configurações dos itens presentes nesta, servindo como uma previsão do resultado final.

Após a renderização do áudio é apresentada a *waveform*, na *track* corresponde aos microfones que existirem no projeto, como exemplificado pela Figura 3.15.

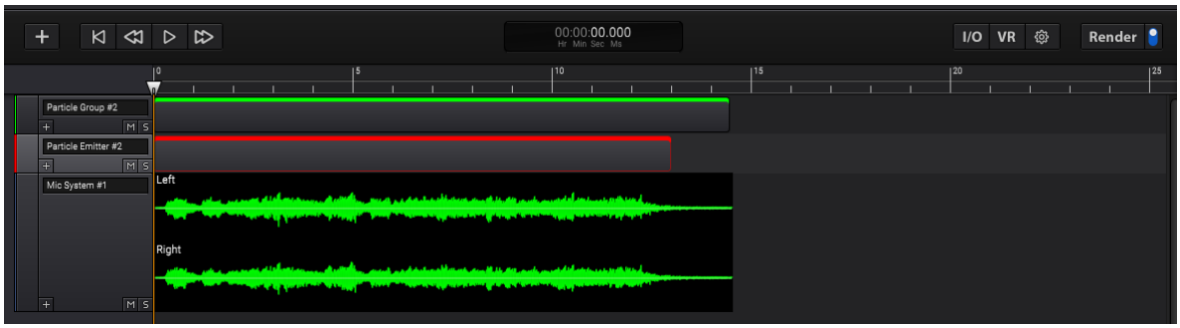

Figura 3.15 – *Transporter*

3.5. Export Window

Após o processo de renderização o utilizador pode aceder ao ecrã de exportação, representado pela Figura 3.16, onde é possível selecionar o microfone a exportar, o formato, a *bit depth*, ordem de canais, bem como os metadados a incluir no ficheiro a exportar e os canais a serem exportados.

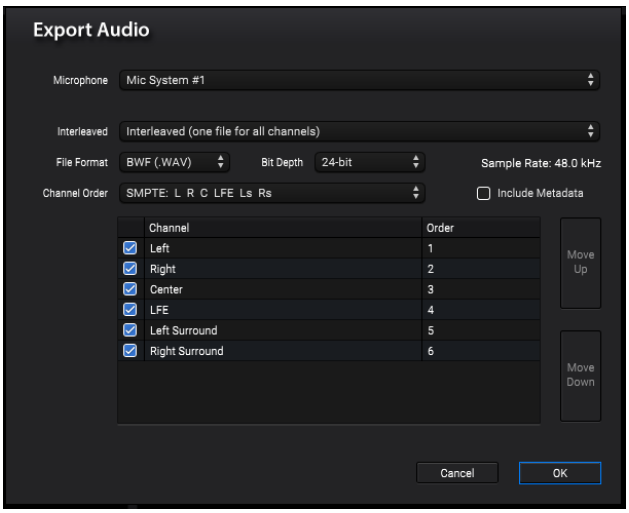

Figura 3.16 – Export Window

3.6. Preferences Window

Para permitir o utilizador adaptar o *software* às suas preferências existe a janela de preferências, representada pela Figura 3.17, onde é possível alterar definições como o sistema métrico, a unidade de tempo, tamanho de letra e o nível global de normalização do áudio, entre muitas outras.

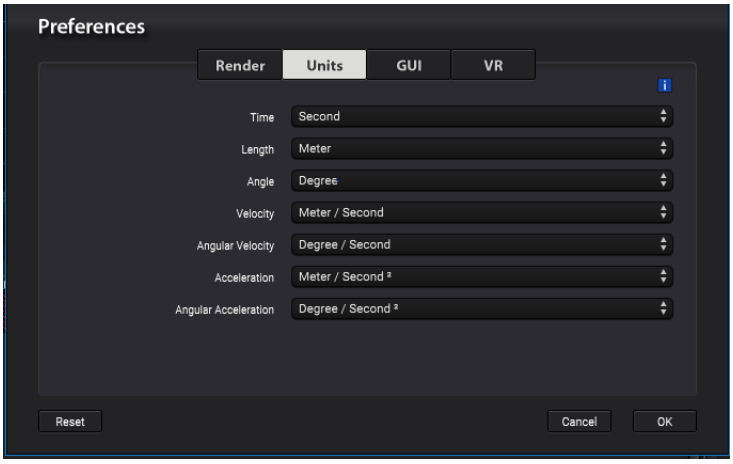

Figura 3.17 – Preferences Window

3.7. Management Account Window

O Sound Particles possui 4 tipos de licenças: Indie, Pro, Team e Education. Um utilizador pode ter associados vários tipos de licenças à sua conta. Para a gestão destas licenças é disponibilizado um ecrã intitulado *Management Account*, representado pela Figura 3.18, onde são apresentadas as licenças da conta que foi usada para iniciar sessão no software, bem como a informação sobre a licença ativa, que normalmente será a que tiver mais privilégios. Neste ecrã é possível mudar de subscrição ativa, bem como sair da conta atual.

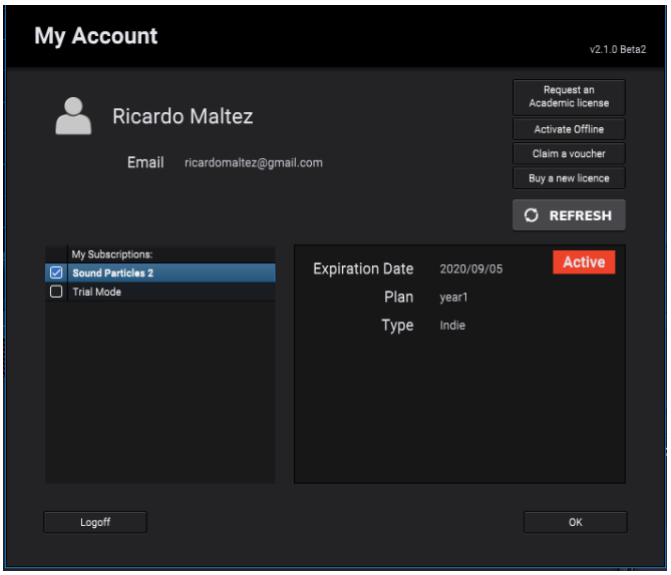

Figura 3.18 – Account Management Window

3.8. Batch Processing Window

Para permitir o processamento de vários ficheiros de áudio em sequência o Sound Particles possui o ecrã *Batch Processing*, representado pela Figura 3.19. Nele é possível selecionar uma pasta de origem de ficheiros de áudio e uma de destino, bem como alterar definições de exportação e de meta dados. Esta funcionalidade é bastante útil em situações em que é necessário aplicar o mesmo efeito de áudio a uma série de ficheiros.

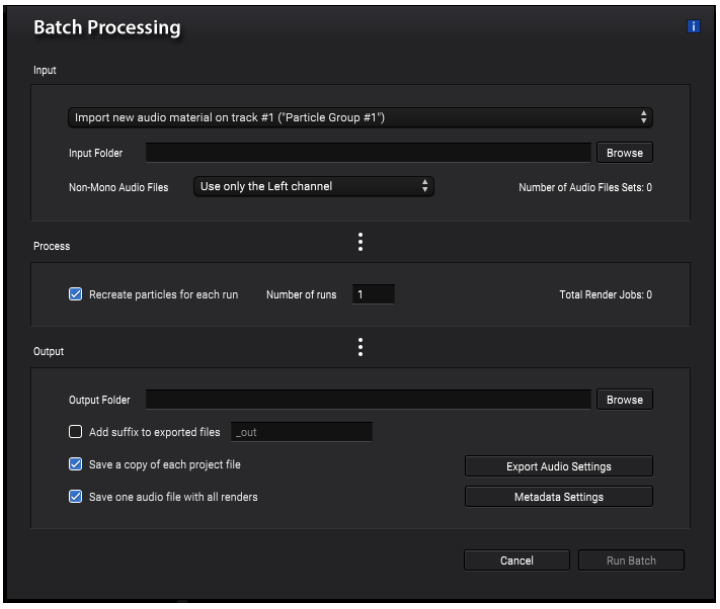

Figura 3.19 – Batch Processing Window

3.9. Levantamento de funcionalidades a serem testadas

Depois de ser analisado o software Sound Particles iniciou-se o levantamento das funcionalidades mais importantes, e mais utilizadas, que poderiam ser testadas através da interface gráfica. Este processo foi elaborado em conjunto com a equipa de QA da empresa e resultou num documento com 1027 testes, cada um com os passos necessários para aplicar um efeito ou alguma alteração que altere o áudio exportado. O objetivo da criação deste documento foi servir como guia para a programação dos testes automatizados, que foram realizados posteriormente. Os testes deste documento foram divididos pela categoria de funcionalidades que testavam, de forma a facilitar a sua organização, como pode ser observado na Tabela 3.1.

Durante este processo foram gerados ficheiros de áudio, para cada um dos testes, com o objetivo de serem usados como referência futura para a execução dos testes, permitindo que no fim da execução de um teste o áudio possa ser exportado e comparado com o áudio previamente gerado.

Tabela 3.1 – Resumo das funcionalidades a ser testadas por categoria

Categoria	Número de testes
Audio Files	2
Audio Modifiers	47
Audio Tracks	34
Automation	313
Batch Processing	61
Export	53
Import CGI	25
Import	10
Microphone	56
Movement Modifiers	60
Particle Emitter	20
Particle Group	24
Particle Track	75
Preferences	2
Project Settings	22
Project Templates	27
Random Distributions	146
Render Actions	7
Resampling	10
Time Editor	10
Track Presents	23
Total	1027

4. Estudo de bibliotecas/*frameworks/ferramentas* de interação com a GUI

Após a elaboração da lista de testes iniciou-se uma pesquisa por bibliotecas, *frameworks* ou ferramentas que permitissem a elaboração da solução pretendida de uma forma eficiente. Nos seguintes subcapítulos são apresentadas as opções encontradas. A seleção foi realizada com base nos seguintes requisitos:

- Suporte para aplicações *desktop* nativas
- Suporte para Windows e Mac OS

4.1. Squish

A Squish [7] consiste numa ferramenta comercial de automação de testes de GUI que permite o teste de aplicações *desktop*, *mobile* e *web*. É desenvolvida e mantida pela empresa Froglogic. Possui várias funcionalidades como gravação de testes e reprodução, criação de pontos de verificação e validação na gravação de testes e em *scripts*, suporte a *data-driven testing*, execução de conjuntos de *scripts*, suporte a *Optical Character Recognition* (OCR) [8] e reconhecimento de elementos gráficos, como botões. Possui suporte a várias linguagens de *scripting* como Python e Javascript. A licença tem um custo de 2400€[1] por utilizador, com 12 meses de suporte.

4.2. Appium

Appium [9] consiste numa *framework* de automação *open-source* para aplicações *mobile* nativas, híbridas e *web*. Contudo inclui suporte beta para aplicações *desktop*. Consiste num servidor *hypertext transfer protocol* (HTTP) que recebe pedidos para efetuar as tarefas pretendidas na UI, o que permite a utilização de qualquer linguagem de programação que suporte pedidos HTTP. Para facilitar a integração com bibliotecas já existentes implementa o protocolo Selenium WebDriver [10]. Possui uma licença Apache 2.0 e não possui custo de utilização.

[1] https://www.froglogic.com/squish/prices/named-user-licensing/ - Data de consulta: 09/04/2020

4.3. Katalon Studio

Katalon Studio [11] consiste num conjunto de ferramentas para testes de GUI para aplicações *web*, *mobile* e *desktop*. É desenvolvido e mantido pela empresa Katalon LLC. Foi construído com base em duas *frameworks* de automação *open-source*, Selenium e Appium. Possui suporte para gravação de testes e reprodução, *data-driven testing* e suporte para as linguagens Java e Groovy. Para complementar, ainda possui integração para *pipelines* de *Continuous Integration/Continuous Delivery* (CI/CD), como Jenkins [12], Bamboo [13] e CircleCI [14]. A licença tem um custo de $759 por utilizador.

4.4. SikuliX

SikuliX [15] consiste numa ferramenta *open-source* de automação que permite detetar elementos na GUI e simular ações do teclado e rato. Foi inicialmente desenvolvida como um projeto de pesquisa *open-source*, pelo User Interface Design Group do MIT. Possui suporte para Windows, Mac e Linux. Utiliza a biblioteca OpenCV para reconhecimento de imagens de forma a encontrar elementos presentes no ecrã, por exemplo um botão que se pretenda pressionar. Também possui funcionalidades de OCR para permitir a procura de texto no ecrã. É constituído por 2 componentes, uma API java que permite a sua utilização numa aplicação desta linguagem, ou equivalentes, e um *integrated development environment* (IDE), representado pela Figura 4.1, que permite a execução e edição de *scripts* utilizando as linguagens Jython e JRuby. Este IDE também possui a funcionalidade de capturar partes do ecrã e incluir as imagens resultantes diretamente no código, tornando a programação mais intuitiva. Possui uma licença MIT e não possui custo de utilização.

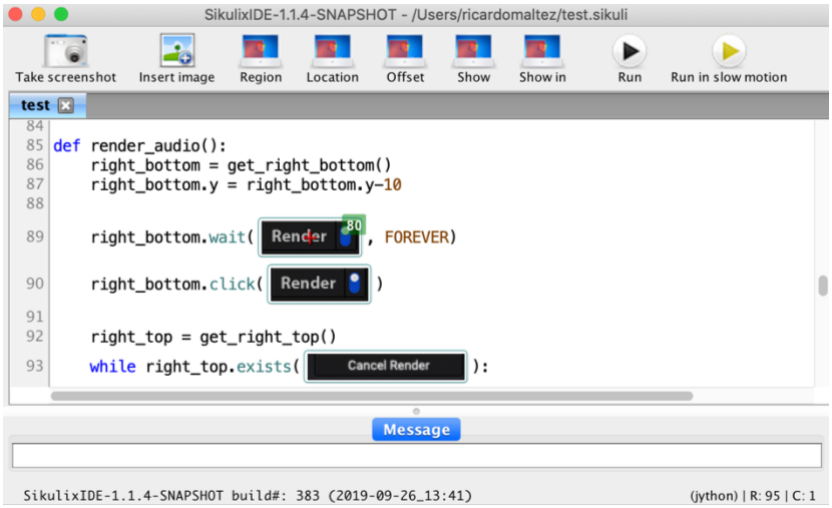

Figura 4.1 – SikuliX IDE

4.5. Seleção de bibliotecas/*frameworks*/ferramentas

Após a análise sobre as bibliotecas, *frameworks* e/ou ferramentas que permitem a interação com a GUI e podiam ser utilizadas no desenvolvimento da solução, foi necessário analisar as vantagens e desvantagens de cada uma, de forma a selecionar a melhor para o caso de estudo. Para facilitar esta seleção foi elaborada a seguinte tabela.

Tabela 4.1 – Comparação de bibliotecas/*frameworks*/ferramentas de integração da GUI

Nome	Suporte multiplataforma (Windows, Mac)	Licença	Custo de utilização	Funcionalidades extra
Squish	Sim	Comercial	2400€ por utilizador por ano	• Gravação de testes • Validação de testes • Suporte a Python e Javascript • *Data-driven testing* • Reconhecimento de texto no ecrã • Reconhecimento de elementos gráficos
Appium	Sim, mas limitado (beta)	Apache 2.0	Gratuito	
Katalon Studio	Não possui suporte para aplicações nativas Mac OS	Comercial	$759 por utilizador	• Gravação de testes • *Debugger* avançado • *Data-driven testing*
SikuliX	Sim	MIT	Gratuito	• Reconhecimento de texto no ecrã • Reconhecimento de elementos gráficos • Integração numa aplicação Java como biblioteca

Devido ao limitado suporte (Appium), ou não existente (Katalon Studio), para aplicações nativas de Mac OS o Appium e o Katalon Studio foram descartados como hipóteses, ficando apenas o Squish e o SikuliX. O SikuliX acabou por ser selecionado para lidar com a interação da UI, por ser uma solução *open-source*, que permite analisar o seu código, e não conter licença de utilização. Contudo, ao SikuliX faltavam funcionalidades fundamentais para a elaboração de casos de teste, pois este apenas permite a interação com a UI. Apesar de possuir funcionalidades de verificação, por exemplo, verificar se um botão se encontra ativo, não possui suporte à elaboração de vários testes consecutivos. Para ultrapassar esta limitação decidiu-se recorrer a uma das *frameworks* mais utilizadas de testes,

titulada JUnit, que foi abordada nas aulas de Qualidade de Software, UC do mestrado de Engenharia Informática - Computação Móvel (ESTG – Politécnico de Leiria).

5. Sound Particles GUI Tester

Sound Particles GUI Tester foi o nome dado à solução, desenvolvida durante o estágio, que permite testar a GUI do Sound Particles. Este capítulo aborda o desenvolvimento desta solução, começando por um protótipo (de forma a testar a exequibilidade e de recolher informações importantes para o seu desenvolvimento), definição da arquitetura e o processo de implementação. O capítulo abora também a integração desta solução na pipeline de desenvolvimento do produto Sound Particles e termina com uma análise crítica sobre a solução desenvolvida.

5.1. Desenvolvimento de Protótipo Funcional

Após a seleção das *frameworks*/bibliotecas a serem usadas decidiu-se, em conjunto com a equipa de desenvolvimento da empresa, que seria bastante importante desenvolver um protótipo para perceber quais as limitações do SikuliX e decisões arquiteturais que seriam importantes ter em conta na construção da solução final. Esta secção aborda o desenvolvimento deste protótipo.

O desenvolvimento do protótipo iniciou-se com a criação de uma aplicação CLI utilizando a linguagem de programação Java, escolhida devido aos requisitos da biblioteca SikuliX. Esta biblioteca foi incluída no projeto através do ficheiro, de formato Java Archive (JAR), disponibilizado no seu site oficial. Foi selecionada a versão 2.0.1, por ser a mais recente no momento da seleção.

Para este protótipo idealizou-se uma arquitetura com 2 módulos, sendo eles:

- JUnit Tests – Responsável por agrupar o código referente aos passos de cada teste e validação do objetivo (ex: verificar se o áudio corresponde ao esperado);
- Sound Particles UI API – Responsável por abstrair as chamadas à biblioteca SikuliX de forma a tornar intuitiva a programação de testes, pois uma ação no Sound Particles pode corresponder a várias interações com a interface gráfica.

O primeiro teste programado possuía o objetivo de percorrer a lista de *templates* disponibilizada pelo *software* Sound Particles, selecionar o ficheiro de áudio correspondente, renderizar, exportar o áudio e verificar se o áudio exportado correspondia ao pretendido. Durante a elaboração deste teste foram imediatamente encontrados vários obstáculos que

teriam de ser ultrapassados durante o desenvolvimento da solução final. Por exemplo, lidar com as diferenças entre os sistemas operativos, pois dependendo do sistema operativo, o caminho para o Sound Particles irá ser diferente, bem como a janela de gravar o ficheiro exportado, e ainda, o acesso à barra de menu superior, que no Mac OS não faz parte da janela da aplicação. E ainda, o próprio desenho do texto apresentou ligeiras diferenças de um sistema para outro. Apesar de ser uma diferença mínima, afetou o nível de confiança nas deteções de elementos gráficos feitas pelo SikuliX.

Após a elaboração de alguns testes da lista produzida anteriormente, secção 3.9, verificou-se que existem vários elementos presentes no componente *Inspector* que podem não estar presentes no ecrã, ou até mudar de sítio, e pode ser necessária a realização de "scroll" para tornar esses elementos visíveis. Notou-se aqui a necessidade de um sistema que permita a localização dinâmica dos elementos que constituem o *Inspector*.

Outra conclusão que foi possível retirar do desenvolvimento deste protótipo foi a variedade de funcionalidades que o Sound Particles possui e que podem variar consoante o microfone ou tipo de *track* selecionado. Por estas razões considerou-se bastante importante desenvolver uma arquitetura que permita um bom suporte ao grande número de funcionalidades presentes no software.

Com o desenvolvimento deste protótipo foi possível perceber que a elaboração da solução pretendida é exequível e que seria fundamental para assegurar a qualidade do produto Sound Particles. Também permitiu perceber que é importante a captura de imagens usando uma resolução idêntica à que seria usada para executar os testes. De facto, verificou-se que o SikuliX deteta mais facilmente elementos gráficos, quando usadas imagens capturadas com a resolução semelhante à qual se encontra, no momento da pesquisa do elemento gráfico no ecrã. Foi selecionada a resolução 1920x1080 tanto para a captura de imagens, como para o ambiente de execução dos testes.

5.2. Definição da arquitetura do Sound Particles GUI Tester

Após o desenvolvimento do protótipo apresentado na secção 5.1, e com base na informação recolhida neste processo, definiu-se a arquitetura da solução que iria permitir a realização automática dos testes de interface gráfica ao software Sound Particles. Este capítulo descreve os resultados desse processo.

Decidiu-se manter os 2 módulos da arquitetura do protótipo, *JUnit Tests* e *Sound Particles GUI API*, e adicionar um novo módulo, titulado *Operating System Abstraction*.

5.2.1. Módulo Operating System Abstraction

O módulo Operating System Abstraction foi criado com o intuito de abstrair as diferenças que existem entre os sistemas operativos, por essa razão foi desenhado com 3 classes, uma classe abstrata, titulada *OsController*, destinada a definir as funções necessárias ao módulo *JUnit Tests* para interagir com o sistema operativo, e duas classes concretas, *WindowsOsController* e *MacOsController*, que estendem da classe *OsController*. Cada uma destas classes implementa as funções declaradas na classe *OsController* para cada um dos sistemas operativos correspondentes. Uma vez que estas duas classes interagem com a interface gráfica, a classe *OsController* estende da classe *ScreenController*, que já possui a capacidade de usar o SikuliX para este fim. O diagrama de classes deste módulo pode ser observado na Figura 5.1 [16].

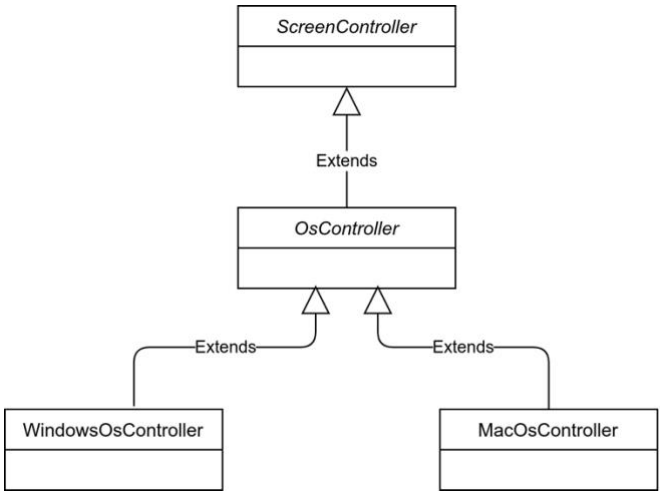

Figura 5.1 – Diagrama do módulo Operating System Abstraction

5.2.2. Módulo Sound Particles GUI API

Devido ao grande número de funcionalidades que se constatou durante o desenvolvimento do protótipo decidiu-se, em conjunto com a equipa de desenvolvimento da empresa, que seria uma boa ideia a definição de uma arquitetura de componentes para este módulo, de forma a agrupar as várias funcionalidades em grupos mais pequenos (facilitando a criação e manutenção de toda a solução). Para a criação desta arquitetura de componentes foi criada uma interface, titulada *Component*, para definir as ações genéricas possíveis num componente, como cliques em botões e escrita de texto em campos de texto. Foi também criada uma classe abstrata, titulada *BaseComponent*, que implementa a lógica de registo de

subcomponentes, de forma a permitir um componente agrupar outros componentes dentro deste.

Uma vez que no software Sound Particles o *Inspector* é composto por conjuntos de parâmetros relacionados, como os parâmetros que fazem parte dos *audio modifiers* ou dos *movement modifiers*, optou-se pela criação de componentes para representar cada um destes grupos, de forma a que cada componente fosse responsável por disponibilizar funções que permitem preencher cada um dos seus parâmetros respetivos. Cada componente criado estende da classe abstrata *BaseComponent*, como mostrado pelo diagrama da Figura 5.2 [16], de forma a herdar as funcionalidades desta.

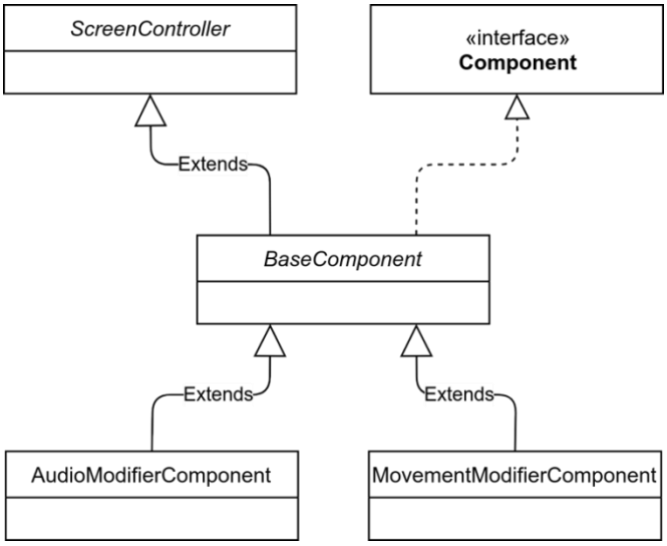

Figura 5.2 – Diagrama de classes de componentes

Para representar os componentes presentes no Inspector, quando se seleciona, por exemplo uma *audio track*, foi criada uma classe correspondente, titulada *AudioTrackInspector*. Esta abordagem foi aplicada aos outros tipos de *tracks* presentes no software. De forma a evitar a repetição de código nestas classes definiu-se uma classe abstrata titulada *BaseInspector*. Esta estrutura pode ser observada no diagrama da Figura 5.3 [16].

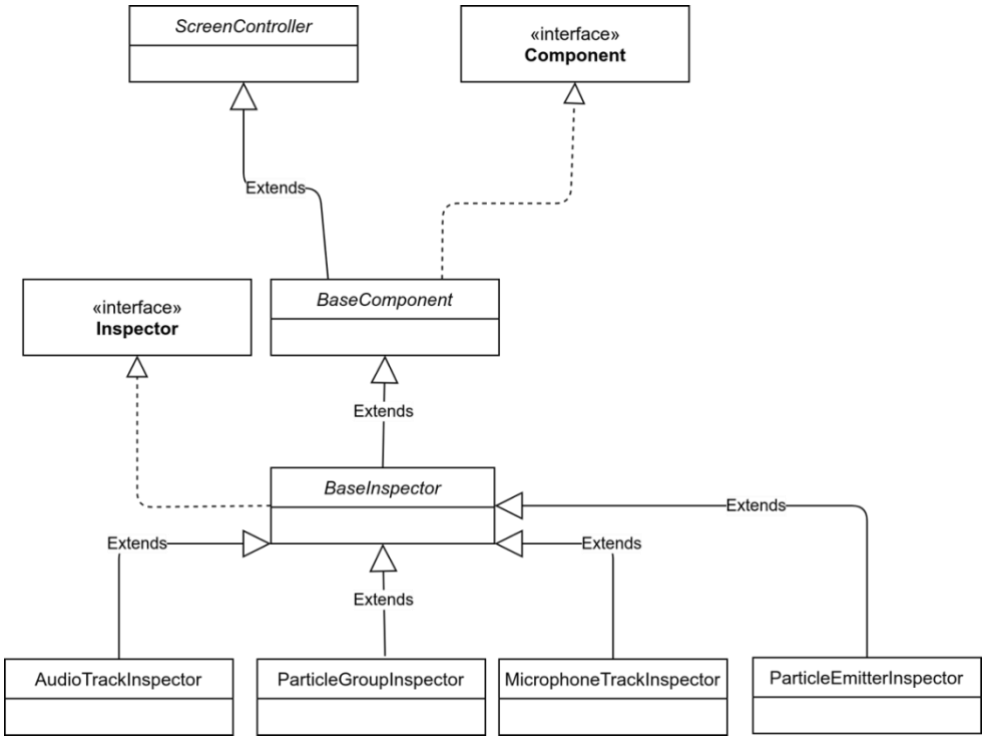

Figura 5.3 – Diagrama dos vários Inspectors

5.2.3. Módulo JUnit Tests

Este módulo foi desenhado com uma classe abstrata, titulada *SoundParticlesBaseTest*, e de várias classes finais que estendem desta, cada classe correspondendo a uma categoria de testes da lista apresentada na secção 3.9. O intuito da classe *SoundParticlesBaseTest* consiste na definição de código comum a cada classe de teste, evitando a repetição de

código, incluindo o código de *setup* necessário ao SikuliX. O diagrama de classes deste módulo pode ser observado na Figura 5.4 [16].

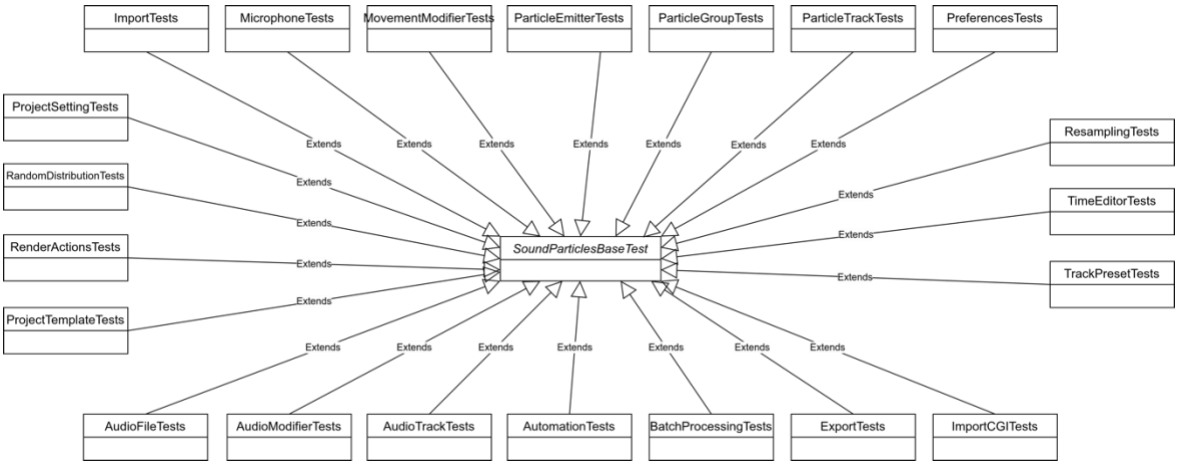

Figura 5.4 – Diagrama do módulo JUnit Tests

5.3. Desenvolvimento do Módulo Operating System Abstraction

Neste módulo foram incluídas as funcionalidades que dependem do sistema operativo, abstraindo essas diferenças de forma a que o código dos outros módulos não precisasse de tratar estas diferenças.

Foram incluídas as seguintes funcionalidades neste módulo:

- Abrir o Sound Particles
- Esperar que o Sound Particles seja iniciado
- Abrir e gravar um ficheiro através do explorador de ficheiros do sistema
- Executar combinação de teclas para selecionar todos os itens do ecrã
- Executar combinação de teclas para copiar e colar

Estas funcionalidades foram definidas na classe *OsController* como funções abstratas e implementadas nas classes de cada sistema operativo, *MacOSController* e *WindowsOSController*.

5.4. Desenvolvimento do Módulo Sound Particles GUI API

O módulo Sound Particles GUI API ficou responsável por disponibilizar funções que permitem ao módulo JUnit Tests interagir com a interface do Sound Particles, abstraindo as chamadas à biblioteca SikuliX. Foi desenvolvido de forma a ser constituído por vários componentes, podendo eles serem componentes simples ou componentes mais complexos, constituídos por vários subcomponentes, como os vários Inspectors que foram definidos para cada tipo de *track*, como referido na subsecção *5.2.2*. A presente secção aborda as várias funcionalidades que foram desenvolvidas para este módulo.

5.4.1. Registo de Componentes

De forma a permitir o registo de componentes, a classe *BaseInspector* dispõe de uma função, titulada *registerComponents*, que cada classe *Inspector* reimplementa, de forma a registar os componentes que o compõem. Este processo tira partido de um *hash map* presente na classe pai, a classe *BaseComponent.* Neste *hash map* a chave corresponde ao tipo de classe de um componente e o valor corresponde à instância desta classe. Desta forma as ações de um teste do módulo JUnit Tests podem reutilizar a instância de um componente, em vez de ser criada uma nova instância por cada ação. Devido a esta operação ser algo que se repete em todos os testes selecionou-se um *hash map* em vez de uma lista, reduzindo o custo do acesso aos componentes.

5.4.2. Reconhecimento do Inspector Atual

Devido à possibilidade da troca do Inspector no meio de um teste, por exemplo, ao selecionar outro tipo de *track*, foi necessário desenvolver a capacidade deste módulo conseguir calcular o Inspector atualmente no ecrã. Uma das opções seria registar todas as ações que provocam a alteração do Inspector, contudo, caso fosse adicionada uma nova ação ao Sound Particles que altere o Inspector, seria necessária a alteração deste módulo para ter em conta essa alteração. Por esse motivo optou-se pela deteção do Inspector através das capacidades do SikuliX, uma vez que cada Inspector possui um título diferente. Utilizando uma imagem correspondente ao título de cada Inspector, representado pela Figura 5.5, foi possível verificar qual se encontrava presente no ecrã. Esta lógica foi implementada na classe *SoundParticlesApplication*. Cada teste possui uma instância, desta classe, e usam-na para invocar as ações dos componentes, que por sua vez atuam na GUI através da biblioteca SikuliX, como abordado com mais detalhe na subsecção 5.4.6.

Figura 5.5 – Título do Inspector

Com a capacidade da deteção do Inspector foi possível criar validações de forma a não permitir a interação com componentes não presentes no Inspector atual ou a execução de ações inválidas, abordado com mais detalhe na subsecção 5.4.5.

5.4.3. Cálculo Dinâmico da Região do Componente

Devido a alguns grupos de parâmetros com os da Start Zone alterarem o seu tamanho dependendo das opções selecionadas, a localização dos parâmetros respetivos de cada componente não é fixa. Pode ser observada essa diferença na Figura 5.6 e Figura 5.7.

Figura 5.6 – Parâmetros da Point Start Zone

Figura 5.7 – Parâmetros da Triangle Start Zone

Além deste problema existem parâmetros com título similar ou mesmo igual, geridos por componentes diferentes. Apenas efetuar a pesquisa de uma imagem com o texto correspondente a um parâmetro, utilizando o SikuliX, poderia não ter os resultados esperados. Para resolver estes problemas foi desenvolvida a funcionalidade de calcular a região de um componente em tempo real, desta forma cada componente calcula a zona do

ecrã respetiva a este, naquele momento, e não afeta os parâmetros de outros componentes por engano. Esta funcionalidade foi desenvolvida de forma a procurar o título de cada componente e o fim deste, assinalados pela Figura 5.8. Cada componente, que estende da classe *BaseComponent*, especifica a imagem que corresponde ao seu título no seu construtor. Este processo não foi necessário para o fim de cada componente, uma vez que este é igual a todos. Apenas foi necessário capturar uma imagem e definir o caminho nesta na classe *BaseComponent*, de forma a que os componentes que estendam desta, conheçam este caminho. Com este caminho cada componente pode utilizar as capacidades do SikuliX para encontrar as coordenadas, onde uma imagem semelhante a esta se encontra no ecrã, e comparar com as coordenadas onde o seu título foi encontrado. Com base na diferença entre estas duas coordenadas cada componente consegue calcular a zona do ecrã que lhe corresponde, evitando interagir com os parâmetros ou botões de outros componentes.

Figura 5.8 – Áreas usadas para cálculo de área do componente

Esta funcionalidade foi implementada na função *computeCurrentRegion* da classe *BaseComponent*, que por sua vez é chamada na função *prepare*. A função *prepare* foi desenvolvida com o intuito de cada componente conseguir fazer ações que necessita antes de realizar uma ação pedida por um teste JUnit. Uma vez que cada componente precisa de calcular a sua região, antes de realizar uma destas ações, esta função mostrou-se ideal para o efeito. O diagrama de sequência do cálculo da região do componente pode ser observado na Figura 5.9 [16].

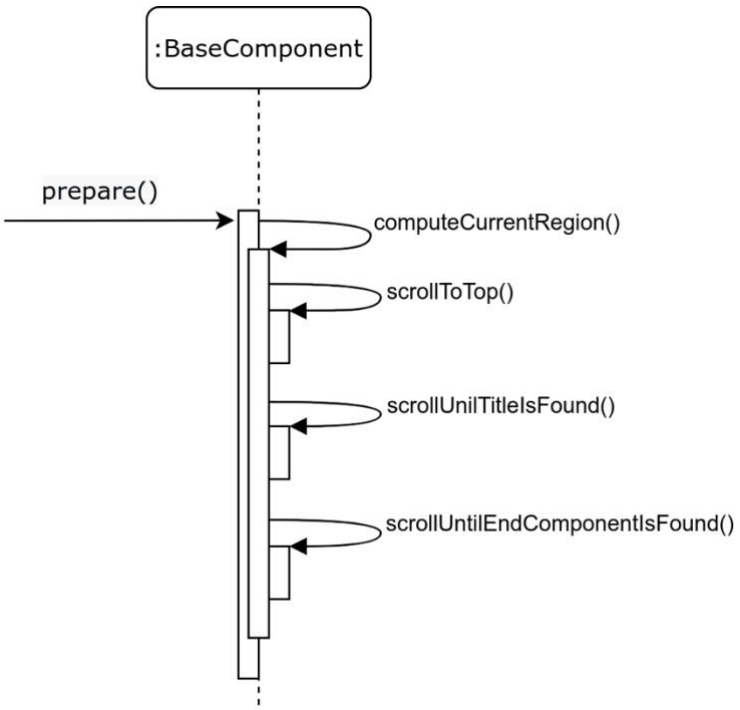

Figura 5.9 – Diagrama de sequência do cálculo da região de um componente

5.4.4. Encaminhamento Dinâmico de Ações

De forma a permitir a execução de ações dos componentes sem ser necessária a criação de várias funções, nas classes que representam os Inspectors que contêm esses componentes (funções que seriam apenas de "ponte" e que teriam que ser duplicadas para os vários componentes que incluem outro componente), optou-se pelo uso de *reflection* de forma a permitir a chamada dinâmica de funções evitando esta duplicação de código. Contudo o programador ter que saber exatamente o nome da função, de um componente, para a conseguir executar não pareceu ser uma boa abordagem. Por essa razão definiu-se que cada componente ia possuir propriedades. Para representar uma propriedade foi criada a classe *Property*, que apenas possui o atributo "name", que indica o nome da propriedade. Cada componente passou a possuir instâncias da classe *Property* com visibilidade pública e definidas como "static", de forma a permitir que um teste aceda a esta propriedade sem

necessitar de uma instância do componente. As funções dos componentes passaram então a ser nomeadas com base na junção das ações definidas na interface *Component* (set, check, add, remove, click, rightClick, toogle, open) e o nome da propriedade. Por exemplo um componente com a propriedade "NumberOfParticles" possui uma função nomeada "*setNumberOfParticles*". Estas propriedades, além de permitirem indicar as possíveis funções de componentes, também permitem identificar o próprio componente. Durante o registo dos componentes, além de ser guardada a instância do componente, como referido na subsecção 5.4.1, são registadas todas as propriedades, do componente que está a ser registado, no *Inspector* que o contém. Desta forma o *Inspector* pode direcionar a execução da ação para o componente respetivo. Estas propriedades foram recolhidas através de *reflection*, que permitiu obter as instâncias estáticas do tipo *Property* dos componentes.

Na Figura 5.10 [16] pode ser observado um diagrama de sequência, que ilustra a execução do encaminhamento da ação de mudar o número de partículas, para o componente responsável por essa ação. Os parâmetros das funções não foram incluídos de forma a simplificar o diagrama.

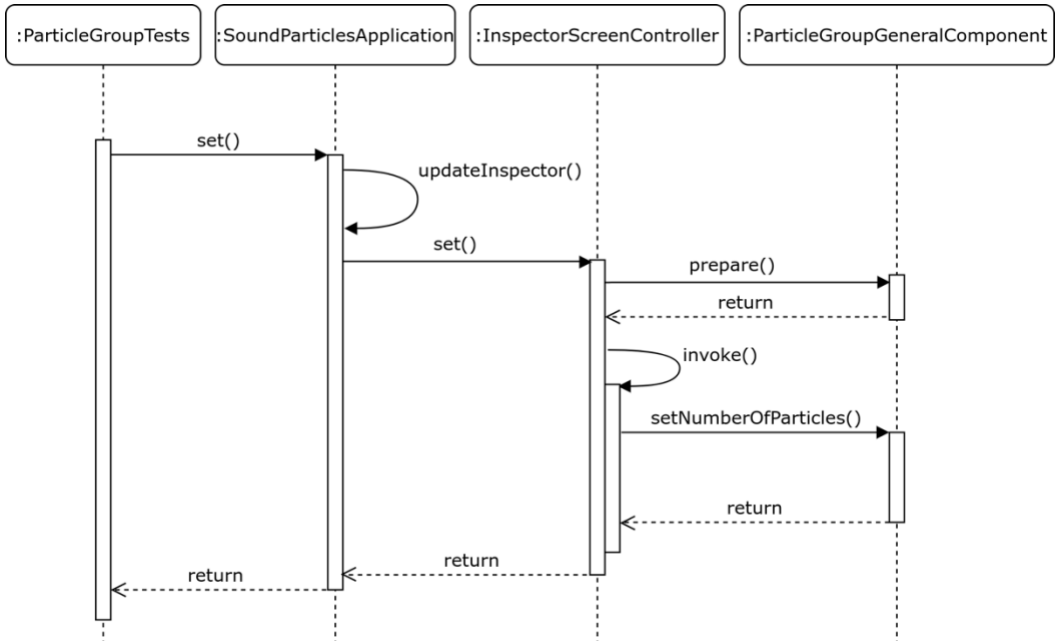

Figura 5.10 – Diagrama de sequência da chamada a uma ação de um componente

O encaminhamento dinâmico de ações para subcomponentes, componentes dentro de outros, também foi suportado, mas é necessário indicar a classe do componente "pai". A lógica do encaminhamento de ações descrita é utilizada apenas com a alteração de que a pesquisa da propriedade, para encontrar o componente que irá executar a ação, é executada não na classe correspondente ao Inspector atualmente ativo mas no componente no qual for indicada a classe. Um exemplo de código do uso desta funcionalidade é mostrado na secção 5.5.

5.4.5. Validação de Ações

Devido à função deste módulo de interagir com a interface gráfica de outro programa, existe a possibilidade de uma ação não ser executada com sucesso, seja por demorar mais tempo do que o suposto, por o campo, ou botão, referente à ação desejada ter mudado de sítio ou por até a ação anterior ainda não ter sido terminada. Existe também o problema de as interações com o Sound Particles serem assíncronas, este pode demorar alguns segundos a responder a ações. Ainda existe a possibilidade de que o Sound Particles, possua um *bug* e a ação não seja concluída com sucesso.

Para mitigar estes problemas foram implementadas várias verificações/mecanismos para confirmar se a ação tinha sido executada com sucesso e esperar pela conclusão desta. Por exemplo, para verificar se um valor foi efetivamente escrito num parâmetro, após a suposta inserção do valor no campo, é selecionado o texto do parâmetro, copiado para o *clipboard* do sistema, obtido o texto do *clipboard* pela função que executou a ação e comparado com o valor esperado. Caso este valor não seja o correto é lançada uma *exception*. Outro exemplo, neste caso para garantir que uma ação foi terminada com sucesso, consiste em aguardar até que o resultado esperado seja encontrado no ecrã pelo SikuliX. É considerado um tempo máximo de 15 segundos de forma a evitar que o teste atual fique "preso" numa ação. Este mecanismo é usado na ativação das *checkboxes*, por exemplo, em que após serem ativadas é aguardado que na zona ocupada por estas apareça a representação de uma *checkbox* ativa.

Foram encontrados casos excecionais onde não foi possível esperar que uma ação fosse concluída através deste método, pois por vezes o SikuliX não tinha precisão suficiente para conseguir distinguir as diferenças presentes antes e depois da execução de uma ação. Para esses casos foi necessário esperar um tempo definido para cada ação. Apesar de o tempo de espera verdadeiramente necessário depender do computador em questão e da ocupação do processamento deste, esta foi a única forma encontrada para tentar atingir o objetivo de

aguardar que a ação fosse terminada com sucesso, evitando passar para a próxima ação sem ter terminado a anterior.

Foram criadas duas classes que estendem de *Exception* de forma a serem lançadas quando uma ação não é realizada com sucesso. Quando não foi possível encontrar o elemento gráfico no ecrã, um botão por exemplo, é lancada a *exception* nomeada de *ScreenElementNotFound*. No caso de a ação não ter o efeito esperado, por exemplo tentar alterar o valor de um parâmetro, mas o parâmetro não ficar efetivamente com esse valor, é lançada a *exception ActionFailException*.

5.4.6. Interface Pública

Com o intuito de facilitar o uso do módulo *Sound Particles GUI API* decidiu-se que seria importante definir um ponto de entrada a este módulo, de forma a encaminhar os pedidos para o componente respetivo. Para atingir este fim foi definida a classe *SoundParticlesApplication*. Nesta classe implementou-se a lógica de receber os pedidos, de interação com a interface gráfica, necessários aos testes e o encaminhamento deste pedido ao Inspector atualmente ativo, abordado na subsecção 5.4.2, ou a outro componente, no caso de ações não relacionadas com Inspectors, como o caso das ações do *Batch Processing*. Outra das responsabilidades desta classe foi a de disponibilizar métodos para permitir abrir a aplicação Sound Particles e fechá-la através do *OsController* associado ao sistema operativo atual.

Uma vez que cada ação que um teste pretenda executar necessita de aceder à instância da classe *SoundParticlesApplication*, ou seja, no código do teste iria repetir-se várias vezes o nome da variável que representava esta instância, esta classe foi desenvolvida com suporte a *method chaining* (algo bastante usado em programação funcional, pela mesma razão). Para implementar este suporte bastou que cada função desta classe devolvesse a referência para o próprio objeto, no caso da linguagem usada, “return this;” no fim de cada função.

5.5. Desenvolvimento do Módulo JUnit Tests

O desenvolvimento deste módulo consistiu na programação de testes de regressão, utilizando a *framework* JUnit, com o objetivo de validar que as funcionalidades do software Sound Particles continuavam a funcionar após serem feitas alterações ao seu código.

Este módulo depende do módulo Sound Particles GUI API para interagir com a interface gráfica do software Sound Particles, de forma a executar as ações necessárias a cada teste. Cada teste foi desenvolvido com o objetivo de validar o correto funcionamento de alguma funcionalidade em particular, ou a combinação de funcionalidades, e exportar o áudio produzindo pelo Sound Particles de forma a ser comparado, no final do teste, com o resultado esperado. O objetivo desta abordagem consistiu em conseguir determinar se uma funcionalidade deixou de funcionar a nível da interface gráfica ou no processamento do áudio.

O resumo das funcionalidades mais importantes cobertas pelos testes programados, separadas por categorias, podem ser observadas na Tabela 5.1. A forma detalhada desta tabela pode ser consultada no Apêndice A. Uma vez que todos os testes fazem importação de ficheiros de áudio, *render* e exportação, estas funcionalidades não foram incluídas. De forma a organizar o código dos testes desenvolvidos, cada categoria tem um ficheiro Java correspondente. O nome deste ficheiro corresponde ao nome da categoria com o sufixo "Tests", por exemplo o código dos testes da categoria "Audio Files" está presente no ficheiro "AudioFileTests.java".

Tabela 5.1 – Funcionalidades cobertas pelos testes

Categoria	**Número de Testes**	**N° de Funcionalidades Cobertas**
Audio Files	2	1
Audio Modifiers	47	4
Audio Tracks	34	6
Automation	313	8
Batch Processing	61	7
Export	53	9
Import CGI	25	5
Import	10	6
Microphone	56	8
Movement Modifiers	60	4
Particle Emitter	20	5
Particle Group	24	5
Particle Track	75	5
Preferences	2	1
Project Settings	22	8

Project Templates	27	1
Random Distributions	146	6
Render Actions	7	3
Resampling	10	4
Time Editor	10	4
Track Presents	23	3

Na Figura 5.11 pode ser observado o código de um dos testes desenvolvidos da categoria Audio Tracks. Este teste cria uma instância da classe *SoundParticlesApplication*, abordada na subsecção 5.4.6, e tira partido do suporte de *method enchaining* desta. Abre o Sound Particles, adiciona uma *audio track*, adiciona um ficheiro de áudio de *input* e um microfone. Para finalizar executa a ação *render* e abre a janela de exportação de áudio e pede que seja exportado. O teste termina com a comparação do áudio exportado com o *output* esperado para este teste. Caso alguma destas ações falhe, seja por um elemento gráfico não ter sido encontrado, um botão por exemplo, ou por uma ação não ter sido realizada com sucesso, é executada uma captura de ecrã para ser analisada posteriormente, de forma a facilitar a análise da razão que levou o teste a falhar.

Na Figura 5.12 pode ser observado o código de outro teste, sendo que este utiliza o suporte à execução de ações dentro de um subcomponente, abordado na subsecção 5.4.4, para alterar os parâmetros de um *movement modifier*.

```java
@Test
public void test1SimpleAudio() throws Throwable {
    String id = "1";
    SoundParticlesApplication app = new SoundParticlesApplication();
    try {
        app.open(id)
                .addAudioTrack()
                .open(AudioFileScreenComponent.OPEN_PANEL)
                .add(ImportAudioFileComponent.AUDIO_FILE, InputFiles.SINE_1S_48K)
                .click(ImportAudioFileComponent.OPEN)
                .addMicrophone()
                .render()
                .openExportAudioPanel()
                .exportAudio(id)
                .close();
        confirmThatOutputIsEqualToExpected(id, FileFormat.BWF);
    } catch (InterruptedException ex) {
        fail(Errors.TEST_INTERRUPTED_ERROR);
    } catch (ScreenElementNotFoundException | ActionFailException e) {
        ScreenController.getScreen().saveScreenCapture(OsController.getCurrentOS().getErrorFolderPath(), id);
        throw e;
    }
}
```

Figura 5.11 – Código de um dos testes desenvolvidos

```java
@Test
public void test30XY() throws Throwable {
    String id = "30";
    SoundParticlesApplication app = new SoundParticlesApplication();
    try {
        app.open(id)
                .addAudioTrack()
                .open(AudioFileScreenComponent.OPEN_PANEL)
                .add(ImportAudioFileComponent.AUDIO_FILE, InputFiles.SINE_1S_48K)
                .click(ImportAudioFileComponent.OPEN)
                .add(AudioTrackMovementModifierComponent.ROTATION_MOVEMENT,
                        RotationMovementModifierType.ROTATION_VELOCITY)
                .withinComponent(AudioTrackMovementModifierComponent.class, (inspector -> {
                    inspector.set(AudioTrackRotationMovementModifierParameterComponent.SPHERICAL_VECTOR,
                            new SphericalVector( azimuth: 360,  elevation: 0,  radius: 0));
                }))
                .addMicrophone()
                .render()
                .openExportAudioPanel()
                .exportAudio(id)
                .close();
        confirmThatOutputIsEqualToExpected(id, FileFormat.BWF);
    } catch (InterruptedException ex) {
        fail(Errors.TEST_INTERRUPTED_ERROR);
    } catch (ScreenElementNotFoundException | ActionFailException e) {
        ScreenController.getScreen().saveScreenCapture(OsController.getCurrentOS().getErrorFolderPath(), id);
        throw e;
    }
}
```

Figura 5.12 – Código de um dos testes com ação num subcomponente

5.6. Integração do Sound Particles GUI Tester na *Pipeline*

Neste subcapítulo é abordada a integração da solução desenvolvida na *pipeline* de desenvolvimento do software Sound Particles. Descreve a constituação da *pipeline* e inclui os problemas que foram detetados, durante a integração, e as alterações que tiveram de ser feitas à solução de forma a ser integrada com sucesso na *pipeline.*

5.6.1. Pipeline

A *pipeline* usada na Sound Particles é constituída por 5 servidores. Estes podem ser divididos em 3 grupos:

- Bamboo Server
- Builders
- Testers

O primeiro grupo (*Bamboo Server*) é constituído por apenas um servidor com o sistema operativo Windows Server, que corre o *software* Bamboo [13], versão 6.7. Consiste num *continuous integration* e *continuous deployment server*. Neste estão configurados os *build plans* para compilar o Sound Particles e gerar os *installers* para Windows e para macOS. Também possui as configurações dos *test plans* para correr os testes unitários e de integração que fazem parte do software Sound Particles.

O segundo grupo (*Builders*) é constituído por dois servidores com o sistema operativo Windows 10 e macOS 10.15. Estes são responsáveis por produzirem os binários do Sound Particles para cada um dos sistemas, a pedido do Bamboo Server. Por último, o grupo Testers é semelhante aos Builders mas com uma responsabilidade diferente, é responsável por executar os testes que fazem parte dos *test plans*, também a pedido do Bamboo Server. Tanto os *Builders* como os *Testers* são acedidos remotamente através do protocolo SSH pelo Bamboo Server.

5.6.2. Alterações na *Pipeline*

De forma a integrar o Sound Particles GUI Tester na *pipeline*, com o objetivo de correr os testes de UI, foi necessário criar três *tasks* no Bamboo Server usando a interface *web* disponibilizada por este. Este processo foi feito em conjunto com a pessoa responsável pela configuração da *pipeline* de forma a garantir a correta configuração da mesma.

A primeira *task* criada teve o intuito de compilar o Sound Particles GUI Tester de forma a ser usado pela segunda *task*. Nesta foram incluídos os passos de obter o código mais recente do repositório respetivo e gerar o ficheiro JAR. Esta *task* foi configurada para correr a cada 10 minutos, para que caso haja algum novo *commit* no repositório o Bamboo tente compilar este projeto e, caso não seja possível, os responsáveis sejam avisados desta ocorrência. Na Figura 5.13 pode ser observado o exemplo de uma *build* do ficheiro JAR realizada pelo Bamboo.

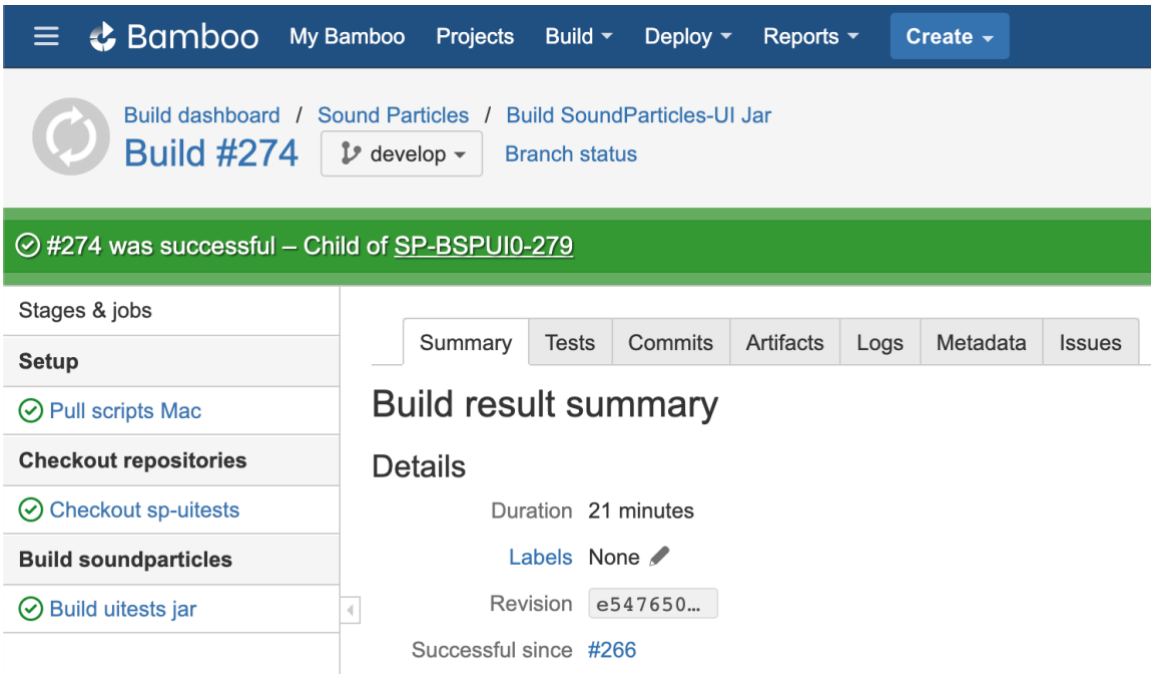

Figura 5.13 – *Build* do Sound Particles GUI Tester

As outras duas *tasks*, uma para cada sistema operativo, tiveram o intuito de transferir o ficheiro JAR, gerado pela *task* anterior, para o computador que iria executar os testes, em conjunto com o binário do Sound Particles que iria ser testado.

Cada *task* foi configurada para correr todos os dias à uma da manhã. Inicialmente eram utilizados os servidores do grupo Testers para executar este ficheiro JAR, contudo, com o aumento do número de testes, o uso de apenas um computador por sistema operativo mostrou-se insuficiente. Correr a lista de testes ultrapassava o tempo disponível destes servidores, que consiste entre a uma e as nove da manhã. Este horário foi definido por dois fatores, o primeiro consiste em que nas horas anteriores estão a ser gerados os binários do Sound Particles e a correr testes unitários e de integração deste.

O segundo fator deve-se a que a equipa de desenvolvimento necessita de obter o resultado dos testes unitários, integração e agora de UI, no dia seguinte de manhã, de forma a facilitar a descoberta da alteração que possa ter levado algum teste a falhar. Uma hipótese para resolver este problema poderia consistir em dividir a lista de testes e correr metade em cada um dos sistemas operativos. Contudo, devido a pequenas diferenças entre estes que possam afetar o resultado final do áudio, considerou-se importante correr toda a lista em cada um dos sistemas. A solução encontrada para este problema foi a paralelização dos testes usando vários computadores, apresentada na subsecção 5.6.3. Esta solução requereu a alteração das *tasks*, anteriormente criadas, de forma a dividir os vários testes pelos computadores disponíveis.

5.6.3. Paralelização dos Testes

Uma vez que os computadores da equipa de desenvolvimento não são usados durante a noite viu-se aqui uma oportunidade de utilizar estes para correr alguns testes, tornando assim possível correr toda a lista de testes durante a noite e obter o resultado de manhã para a equipa de desenvolvimento os analisar. De forma a atingir este objetivo foi necessário configurar as várias máquinas para iniciarem automaticamente antes da segunda *task*, criada no Bamboo, ser executada. Tornar este processo automático era algo importante pois existia a possibilidade de o computador ser desligado no fim do dia. Para automatizar este processo foi necessário alterar algumas configurações. Nos computadores Windows foi necessário alterar uma configuração da BIOS. Para os computadores com o sistema macOS foi possível fazer as configurações necessárias nas definições de sistema. Contudo, para correr os testes de UI do Sound Particles é necessário ter a sessão iniciada no computador, uma vez que é necessário abrir o Sound Particles e isto não é possível sem a sessão iniciada. A solução para este problema é abordada na subsecção 5.6.4.

De forma a permitir a paralelização dos testes foi necessário distribuir estes pelas várias máquinas disponíveis. Uma vez que o número de máquinas consistia num número inferior ao de categorias decidiu-se fazer esta distribuição de forma a que um computador executasse várias categorias, mas sem ultrapassar o horário estipulado. Devido à probabilidade de ser introduzido um *bug*, que bloqueie o Sound Particles durante a execução de uma ação ou o torne excessivamente mais lento, foram definidos *timeouts* para cada uma das categorias dos testes de UI. Estes podem ser observados na Tabela 5.2. Quando este *timeout* é excedido o

Sound Particles GUI Tester é forçado a terminar e o Bamboo recolhe o relatório dos testes concluídos até esse momento.

Tabela 5.2 – *Timeouts* definidos por categoria de teste

Categoria	**Timeout (mins)**
Audio Files	2
Audio Modifiers	60
Audio Tracks	30
Automation	240
Batch Processing	60
Export	60
Import CGI	60
Import	10
Microphone	45
Movement Modifiers	45
Particle Emitter	20
Particle Group	20
Particle Track	60
Preferences	5
Project Settings	20
Project Templates	27
Random Distributions	120
Render Actions	7
Resampling	10
Time Editor	15
Track Presents	25

5.6.4. Desenvolvimento de Aplicação de Início de Sessão Remoto

Após consultar a documentação do SikuliX concluiu-se que este possui suporte à interação com a interface gráfica de um computador remotamente, através do protocolo *Virtual Network Computing* (VNC). Com base na experiência prévia com esta biblioteca considerou-se que não seria necessário grande esforço para criar uma aplicação que se conectasse a uma máquina remota e fizesse o login, com a conta dedicada a correr os testes, através da interface gráfica. Após uma breve análise de soluções existentes para atingir este objetivo, que funcionassem para os dois sistemas operativos usados e que pudessem ser facilmente integradas na *pipeline*, decidiu-se que seria uma melhor opção desenvolver uma aplicação para este fim.

Desenvolveu-se esta aplicação como sendo CLI, de forma a receber por argumentos, ao ser executada, o IP do computador a ser conectado, a *password* configurada para o servidor VNC, o *username* e a *password* a serem usados para efetuar o login no ecrã de início de sessão. Utilizou-se a linguagem de programação Java com a biblioteca SikuliX, tal como no Sound Particles GUI Tester. O funcionamento desta aplicação pode ser resumido nos seguintes passos:

- Validação dos parâmetros recebidos
- Ligação ao computador remoto por VNC
    - Esperar que alguma imagem seja transmitida
    - Encontrar o campo de *username*, usando o SikuliX com base numa imagem recolhida
    - Enviar um clique do rato nas coordenadas onde foi encontrado o campo *username*
    - Transmitir as teclas correspondentes ao *username*
    - Transmitir a tecla TAB para mudar o foco para o parâmetro *password*
    - Transmitir as teclas correspondentes à *password*
    - Transmitir a tecla ENTER e aguardar que o login seja feito com sucesso
- Fechar ligação VNC

Para utilizar esta ferramenta foi necessário instalar, no caso do Windows, ou ativar, no caso do macOS, um servidor VNC em cada uma das máquinas que iria executar o Sound Particles GUI Tester.

5.6.5. Alterações no Sound Particles GUI Tester devido à integração

De forma a facilitar a integração do Sound Particles GUI Tester foram necessárias algumas alterações sendo elas, adicionar suporte ao Apache Ant [17], suporte a receber por argumento as categorias a correr, bem como o número de vezes a correr cada um dos testes. E ainda, a geração de relatórios com os resultados dos testes no formato definido pelo JUnit.

Suporte ao Apache Ant

O Apache Ant [17] consiste numa ferramenta de linha de comandos que permite a compilação de aplicações, normalmente utilizada para aplicações Java. Requer um ficheiro no formato XML para ler as configurações que são necessárias para compilar a aplicação pretendida.

Esta ferramenta foi selecionada devido ao suporte que o *software* Bamboo possui a esta e por o IDE, utilizado durante o desenvolvimento do Sound Particles GUI Tester, Intellij Community Edition, possuir suporte à geração automática do ficheiro XML necessário.

Ao ser utilizada esta ferramenta foi facilitado o processo da geração do ficheiro JAR do projeto Sound Particles GUI Tester, sendo apenas necessário executar o comando:

```
ant -f build.xml
```

Uma vez que o ficheiro gerado com as configurações necessárias foi adicionado à pasta raiz do projeto.

Suporte a Argumentos

De forma a possibilitar a execução de testes em várias máquinas foi necessário adicionar suporte a receber o nome das categorias de testes que seriam necessárias executar. A biblioteca JUnit permite a criação de filtros nos testes a ser executados. Por essa razão programou-se um filtro, criando uma classe que implementa a interface *org.junit.runner.manipulation.Filter*, que filtra os testes a serem executados com base no nome das classes e no nome das funções que foram declaradas usando a *annotation* “Test”. Definiu-se que o formato do argumento para utilizar este filtro seria:

```
test=<Nome do Ficheiro de Teste>#<Nome da função>
```

Sendo que, caso não seja indicado o nome da função, todos os testes da categoria indicada são executados. Por exemplo:

```
test=AudioFileTests
```

```
test=AudioFileTests#test95EnableAudioStreams
```

A seleção de múltiplas categorias também foi suportada, sendo que os nomes dos ficheiros correspondentes às categorias devem ser separados por vírgula. Por exemplo:

```
test=AudioFileTest,AudioModifierTests
```

De forma a facilitar a execução do mesmo teste várias vezes foi adicionado o argumento "runs", que recebe o número de vezes que se pretende correr cada um dos testes. Este argumento é opcional, sendo que por omissão é considerado o valor 1.

Geração de JUnit Reports

Para que o resultado dos testes fosse recolhido pelo Bamboo, de forma a mostrar o número de testes falhados e bem-sucedidos, bem como o histórico de um teste em particular, foi necessário que no fim da execução dos testes fosse gerado um relatório com esta informação num formato que o Bamboo conseguisse interpretar. Uma vez que o Sound Particles GUI Tester utiliza a biblioteca Junit e que o Bamboo suporta o formato e estrutura de relatório gerado por esta biblioteca, tirou-se proveito do JUnit para que durante a execução dos testes fosse gerado este relatório. Na Figura 5.14 pode ser observado um exemplo de um destes relatórios com o resultado dos testes corridos. Neste consta o número de testes corridos, o número de testes que falharam, a razão pela qual falharam e alguma informação sobre o teste, tal como o tempo que demorou a correr e a classe onde pode ser encontrado.

```
<?xml version="1.0" encoding="UTF-8" ?>
<testsuites failures="1" tests="1" time="131.072">
  <testsuite failures="1" hostname="Ricardos-Mac.local"
             name="com.soundparticles.test.render.AudioFilesTests" tests="1" time="131.072" timestamp="2020-06-16T01:02:04">
    <properties />
    <testcase classname="com.soundparticles.test.render.AudioFilesTests" name="test95EnableAudioStreams" time="84.582">
      <failure message="Failed to find audio file title" type="test95EnableAudioStreams(com.soundparticles.test.render.AudioFilesTests)">
          com.soundparticles.exceptions.ScreenElementNotFoundException: Failed to find audio file title
    at com.soundparticles.controller.screen.inspector.components.AudioFileScreenComponent.openOpenPanel(AudioFileScreenComponent.java:64)
    at com.soundparticles.controller.screen.inspector.components.BaseComponent.invoke(BaseComponent.java:95)
    at com.soundparticles.controller.screen.inspector.InspectorScreenController.open(InspectorScreenController.java:136)
    at com.soundparticles.model.SoundParticlesApplication.open(SoundParticlesApplication.java:282)
    at com.soundparticles.test.render.AudioFilesTests.test95EnableAudioStreams(AudioFilesTests.java:30)
    at com.soundparticles.TestRunner.runTests(TestRunner.java:63)
    at com.soundparticles.TestRunner.main(TestRunner.java:34)
</failure>
    </testcase>
  </testsuite>
</testsuites>
```

Figura 5.14 – Exemplo de relatório JUnit

Estes relatórios são recolhidos pelo Bamboo, usado o comando `scp`, e processados permitindo a consulta dos resultados na interface *web* disponibilizada por este, como pode ser observado na Figura 5.15.

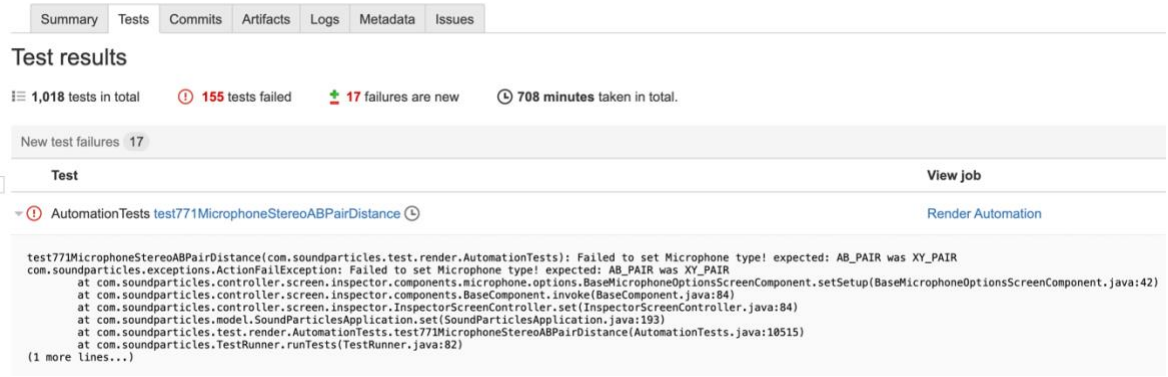

Figura 5.15 – Resultado dos testes na *interface web* do Bamboo

5.7. Análise crítica e proposta de melhorias

Apesar de o Sound Particles GUI Tester permitir detetar funcionalidades que deixaram de funcionar e automatizar grande parte dos testes de UI do Sound Particles, este requer alguma manutenção. Quando a interface gráfica do Sound Particles é alterada, é necessário atualizar as imagens capturadas, ou quando é alterado algo no Sound Particles que afete o áudio exportado, é necessário gerar novamente o áudio de referência. De forma a adicionar suporte a novas ações é necessário recolher imagens dos botões ou outros controlos que seja preciso interagir, o que pode consumir algum tempo. Uma melhoria que poderia ser introduzida consiste no uso da capacidade de OCR do SikuliX para detetar botões através do texto destes, por exemplo, para o botão “ok” tentar encontrar o texto correspondente em vez da imagem do botão. Esta opção foi descartada, durante o desenvolvimento do protótipo, devido aos maus resultados que se obteve, contudo, o SikuliX utiliza o Tesseract [8] para a funcionalidade de OCR. Uma vez que o Tesseract possui suporte ao treino do modelo que utiliza para reconhecimento de caracteres, é possível que, treinando o modelo com imagens de texto com a fonte utilizada no Sound Particles, esta opção se torne viável.

Outro dos aspetos negativos, da solução desenvolvida, foi a pouca cobertura de todas as funcionalidades e combinação de opções que o Sound Particles possui, pois seria necessário um número bem superior de testes, para atingir uma cobertura perto dos 100%. Com um maior investimento de tempo e de recursos é um objetivo possivelmente alcançável, pois o Sound Particles GUI Tester já possui suporte a grande parte das funcionalidades do Sound Particles.

Uma das maiores limitações do Sound Particles GUI Tester é o facto de bloquear a interface gráfica, uma vez que esta é usada para executar os testes. Esta limitação pode ser ultrapassada utilizando vários computadores, como se pude constatar durante a integração na *pipeline*. Outra possível solução seria o uso de máquinas virtuais, contudo esta opção aumentaria bastante o tempo de execução dos testes e não foi explorada.

Após todos os aspetos negativos e limitações da solução desenvolvida, o Sound Particles GUI Tester permite a programação de testes de uma forma bastante intuitiva e fácil, além de garantir que as interações na UI têm o objetivo esperado, algo que a simples gravação e reprodução de interacões do utilizador não permitiria. Estes testes possuem também alguma flexibilidade a pequenos ajustes da UI, pois o Sound Particles GUI Tester não está dependente de coordenadas do ecrã. Por estas razões foi considerado um bom investimento e fundamental para garantir a qualidade de um produto tão complexo como o Sound Particles.

6. **Conclusão**

Todas as etapas inicialmente definidas para este estágio foram concluídas com sucesso, conseguiu-se desenvolver o software inicialmente planeado, que tinha o objetivo de testar o funcionamento do Sound Particles através da interface gráfica, bem como integrar este na *pipeline* de desenvolvimento do produto Sound Particles, reforçando a bateria de testes já existente. Com esta nova ferramenta a empresa passou a conseguir garantir a qualidade, de forma automatizada, de mais uma das partes que constituem o Sound Particles, a sua interface gráfica.

Durante todo este processo foi possível reforçar a componente de programação, bem como adquirir experiência em definir a arquitetura de uma solução que cumpra os requisitos propostos, fundamental na área de engenharia informática. Foi também possível adquirir experiência no desenvolvimento de testes automatizados. Com a programação destes e com a interação com a equipa de QA e de desenvolvimento, foi possível perceber a importância da criação de testes automatizados e da sua integração na *pipeline*, de forma a permitir o desenvolvimento contínuo de um produto.

Como todos os processos existem pontos menos positivos. O caso deste estágio não foi exceção, o desenvolvimento dos 1027 testes, que consistiu na tarefa mais demorada, não significou uma grande aquisição de conhecimentos, devido à sua repetição. Contudo, o contacto com o desenvolvimento de *software* num ambiente profissional, nomeadamente software de áudio, compensou bastante este aspecto menos positivo. A vertente de áudio bastante presente nesta empresa permitu adquirir bastante conhecimento nesta área, bem como perceber como o mundo do áudio e do software se podem combinar em produtos fascinantes, como o Sound Particles.

Bibliografia

[1] "Jornal de Leiria - Sound Particles do IPLeiria nomeado para a categoria 'Outstanding Product.'" https://www.jornaldeleiria.pt/index.php/noticia/sound-particles-do-ipleiria-nomeado-para-a-categoria-outstanding-product-5782 (accessed Oct. 18, 2020).

[2] "REAPER | Audio Production Without Limits." https://www.reaper.fm/ (accessed Aug. 02, 2020).

[3] "Pro Tools - Music Software - Avid." https://www.avid.com/pro-tools (accessed Aug. 02, 2020).

[4] N. Fonseca, "3D PARTICLE SYSTEMS FOR AUDIO APPLICATIONS."

[5] "About — blender.org." https://www.blender.org/about/ (accessed Oct. 18, 2020).

[6] "Touring Machine Company » Blog Archive » Pitch, Roll, and Yaw." https://www.touringmachine.com/Articles/aircraft/6/ (accessed Aug. 01, 2020).

[7] "Automated GUI Testing - Squish GUI Tester - froglogic." https://www.froglogic.com/squish/ (accessed Aug. 01, 2020).

[8] C. Indravadanbhai Patel, D. Patel, C. Patel Smt Chandaben Mohanbhai, A. Patel, S. Chandaben Mohanbhai, and D. Patel Smt Chandaben Mohanbhai, "Optical Character Recognition by Open source OCR Tool Tesseract: A Case Study," *Artic. Int. J. Comput. Appl.*, vol. 55, no. 10, pp. 975–8887, 2012, doi: 10.5120/8794-2784.

[9] S. Singh Ȧ, R. Gadgil Ȧ, and A. Chudgor Ȧ Ȧ, "Automated Testing of Mobile Applications using Scripting Technique: A Study on Appium," 2014. Accessed: Aug. 01, 2020. [Online]. Available: http://inpressco.com/category/ijcet.

[10] P. Ramya, V. Sindhura, and P. V. Sagar, "Testing using selenium web driver," in *Proceedings of the 2017 2nd IEEE International Conference on Electrical, Computer and Communication Technologies, ICECCT 2017*, Nov. 2017, pp. 1–7, doi: 10.1109/ICECCT.2017.8117878.

[11] "Katalon | Simplify Web, API, Mobile, Desktop Automated Tests." https://www.katalon.com/ (accessed Aug. 01, 2020).

[12] V. Armenise, “Continuous delivery with Jenkins: Jenkins solutions to implement continuous delivery,” in *Proceedings - 3rd International Workshop on Release Engineering, RELENG 2015*, May 2015, pp. 24–27, doi: 10.1109/RELENG.2015.19.

[13] “Bamboo Continuous Integration and Deployment Build Server.” https://www.atlassian.com/software/bamboo (accessed Aug. 01, 2020).

[14] “Continuous Integration and Delivery - CircleCI.” https://circleci.com/ (accessed Aug. 01, 2020).

[15] “RaiMan’s SikuliX.” http://sikulix.com/ (accessed Aug. 01, 2020).

[16] Omg, “An OMG ® Unified Modeling Language ® Publication OMG ® Unified Modeling Language ® (OMG UML ®) OMG Document Number: Date,” 2009. Accessed: Sep. 06, 2020. [Online]. Available: https://www.omg.org/spec/UML/20161101/PrimitiveTypes.xmi.

[17] “Apache Ant.” https://ant.apache.org/ (accessed Aug. 02, 2020).

Apêndices

Apêndice A - Funcionalidades Cobertas Pelos Testes Criados

Categoria	**Número de testes**	**Estimativa de Funcionalidades Cobertas**
Audio Files	2	- Ativação/desativação de ficheiros de áudio importados
Audio Modifiers	47	- Aplicação de todos os *audio modifiers.* - Utilização dos parâmetros de cada *audio modifiers,* com diferentes valores. - Utilização de *particle groups* com diferente número de partículas. - Utilização de p*articles groups* com diferentes *start zones.*
Audio Tracks	34	- Utilização de a*udio tracks* com *start zones* em posições diferentes. - Utilização de todos os *movement modifiers.* - Utilização de *audio tracks* com diferentes tipos de *source* (Stereo, 5.1, 7.1 e Ambisonics) - Utilização de diferente número de partículas - Utilização de diferente número de partículas emitidas por segundo. - Combinação de *audio tracks* com *particle groups* e *particle emitters*.
Automation	313	- *Audio tracks* com diferentes tipos de *source (*stereo, *5.1, 7.1)* - Automação de valores dos parâmetros: • Volume • *Particle rate* • *Start Time* • *Start zone position* • *Radius* • *Yaw* • *Pitch* • *Roll*

		• *Azimuth* • *Elevation* • *Microphone angle* • *Microphone front-distance* • *Microphone capsule distance* - Aplicação de todos os *audio modifiers* e automação de cada parâmetro destes. - Seleção de diferentes *start zones.* - Utilização de a*udio tracks*, *particle groups* e *particle emitters.* - Utilização da navegação temporal. (Exemplo: Avançar 10 segundos na *timeline).* - Criação de *automation key points* de forma a criar variações de valores de parâmetros ao longo do tempo. - Utilização de diferentes *outputs* de microfones (Mono, Stereo).
Batch Processing	61	- Renderização e exportação de ficheiros em *batch.* - Utilização de diferentes números de partículas. - Utilização de diferentes números de emissão de partículas por segundo. - *Audio tracks* com diferentes tipos de *source* (*stereo*, 5.1, 7.1). - Seleção de canais a ser exportados. - Diferentes valores de sample rates (1000, 5000 20000, 192000). - Diferentes microfones (Mono, Stereo, 7.1).
Export	53	- Diferentes ordens de canais na exportação de áudio (SMPTE, Film, Pro Tools). - Exportação de cada canal para um ficheiro - Seleção dos canais a ser exportados. - Seleção de diferentes números de partículas. - Seleção de diferentes números de emissão de partículas por segundo. - Seleção de diferentes valores de *sample rates.* - Seleção de diferentes *movement modifers.*

		- Seleção de diferentes microfones (Mono, 5.1, 7.1). - Exportação com formatos de ficheiro diferentes (WAV, AIFF, FLAC).
Import CGI	25	- Importação de diferentes ficheiros de áudio - Importação de ficheiros FBX, modelos 3D e animações. - Seleção de objetos 3D presentes nos ficheiros FBX. - Associação de objetos 3D a *audio tracks*, *particles groups* ou *particle emitters.* - Diferentes valores de offset entre a posição do objeto na animação e a posição a ser considerada.
Import	10	- Importação de diferentes ficheiros de áudio. - Importação parcial de áudio com vários canais (apenas *left channel* ou *right channel*). - Importação de áudio com vários canais com diferentes *channel orders* (SMPTE e Film). - Importação de áudio com vários canais com conversão para *mono*. - Importação de áudio com vários canais em *streams* separadas. - Seleção de diferentes microfones (*Mono*, 5.1, 7.1, 7.1 3/4).
Microphone	56	- Seleção de diferentes números de partículas. - Seleção de diferentes valores de volume do microfone. - Microfone *mono* com várias configurações (*Omni, Figure-of-8, Cardioid* e *Hypercardioid*). - Seleção de *movement modifiers*. - Seleção de diferentes valores de *start time*. - Ativação/desativação de *stems*. - Microfone *stereo* com várias configurações (*Angle, Distance, Front Distance*) - Configuração personalizada de sistema de microfones.
Movement Modifiers	60	- Seleção de diferentes números de partículas. - Seleção de diferentes números de emissão de partículas por segundo. - Aplicação de todos os *movement modifiers* a *audio tracks, particle groups* e *particle emitters.* - Seleção de todos os parâmetros de *movement modifiers*.

Particle Emitter	20	- Uilização de *particle emitters*. - Combinação de *audio tracks*, *particle emitters* e *particle groups*. - Seleção de diferentes valores de emissão de partículas por segundo. - Seleção de diferentes valores de duração de emissão de partículas. - Exportação de *render* de áudio incompleto.
Particle Group	24	- Utilização de *particle groups.* - Seleção de diferentes números de partículas. - Seleção de diferentes *start zones.* - Seleção de diferentes valores de rotação de grupo de partículas (Yaw, Pitch e Roll). - Aplicação de vários *audio modifers.*
Particle Track	75	- Utilização de *particle groups.* - Seleção de diferentes números de partículas. - Utilização de todas as *start zones.* - Selecção de diferentes posições da *start zone*. - Selecção de diferentes valores de raio e altura da *start zone*.
Preferences	2	- Utilização de diferentes níveis de normalização de microfone.
Project Settings	22	- Ativação/desativação do parâmetro *air delay*. - Seleção de diferentes valores de velocidade do som. - Ativação/desativação do parâmetro *distance attenuation.* - Utilização de diferentes valores de *distance attenuation*. - Utilização de diferentes valores de *minimum distance* da *distance attenuation*. - Alteração do parâmetro “Within”, que define como a *distance attenuation* é processada para partículas com distancia ao microfone inferior à *minimum distance*. - Seleção de diferente *sample rates.* - Ativação/desativação do efeito *doppler*,
Project Templates	27	- Abertura de cada um dos *Project Templates* que estão incluídos no Sound Particles.

Random Distributions	146	- Seleção de todo o tipo de Random Distributions para os seguintes parâmetros: • Audio Modifiers -> Delay -> Random Delay -> Delay • Audio Modifiers -> Granular -> Granular Synthesis -> Grain Size • Audio Modifiers -> Time/Pitch -> Random Time Pitch -> Pitch • Movement Modifiers -> Straight Line -> X, Y, Z • Movement Modifiers -> Rotation -> Azimuth, Elevation, Radius - Utilização de todos os *movement modifiers*.
Render Actions	7	- Realização da acção "Clear Render", verificando que as *samples* do áudio exportado, antes e depois desta operação, são iguais. - Exportação de áudio através do modo "auto-render" (áudio produzindo clicando no botão "Play" em vez do botão "Render"). - Parar processamento do áudio e retomar o mesmo.
Resampling	10	- Alteração de *sample rate.* - Importação parcial de ficheiros de áudio, apenas left channel ou *right channel.* - Importação de ficheiros de áudio multicanal com conversão para *mono*. - Importação de ficheiros de áudio multicanal para *streams* separadas.
Time Editor	10	- Ativação do modo mute de uma das *tracks*. - Ativação do modo solo de uma das *tracks*. - Duplicação de *audio tracks*, *particle groups*, *particle emitters* e microfones. - Duplicação de *tracks* com modo solo ou mute ativo.
Track Presents	23	- Aplicação de *presets*, incluídos no Sound Particles, a *audio tracks, particle groups, particle emitters* e a microfones. - Gravação de *presets* personalizados. - Utilização de *presets* previamente criados.