FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Towards a Live Software Development Environment

Diogo da Silva Amaral

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Ademar Aguiar

Coorientador: Rui Nóbrega

27 de junho de 2018

Towards a Live Software Development Environment

Diogo da Silva Amaral

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Filipe Correia
Arguente: Luís Borges Gouveia
Orientador: Ademar Aguiar

---

27 de junho de 2018

Resumo

O constante aumento da complexidade e dimensão dos sistemas de software tornam a sua compreensão cada vez mais custosa e morosa, principalmente, quando essa compreensão é realizada por um programador distante no tempo ou espaço do seu autor original. Essa dificuldade constitui um entrave para os processos seguintes de manutenção, seja ela corretiva ou evolutiva, que não poderá ser realizada sem o completo entendimento do sistema.

Embora sejam usadas boas práticas no desenvolvimento de software, um sistema que tenha um crescimento considerável não conseguirá evitar o aumento da sua complexidade essencial, o que evidencia o problema dos sistemas poderem demorar muito tempo a serem compreendidos e evoluídos pelos utilizadores. São necessárias tarefas exaustivas de análise que se traduzem em elevados custos.

Este trabalho visa mitigar este problema permitindo que os utilizadores entrem literalmente no sistema de software de uma forma *live* utilizando realidade virtual. Foi desenvolvida uma ferramenta para sistemas Java que recebe informações quanto à análise estática e dinâmica do sistema de software, com recurso a ferramentas de *reverse engineering*. Cabe-lhe a tarefa de, em tempo real e durante a execução do sistema, permitir a sua visualização utilizando metáforas visuais. A interação com o sistema em plena execução é um fator fulcral, seguindo conceitos de *live programming* que permitem criar um *feedback* fluído entre o programa e o programador. Esta nova abordagem de *live software development* recorre à realidade virtual para a construção de um ambiente, através do qual é possível compreender o sistema e explorá-lo de uma forma interativa e imersiva.

Para a sua avaliação foram realizadas experiências controladas com utilizadores experientes e com casos de estudo de sistemas de software em Java.

Este contributo pretende inovar, na medida em que junta ao *liveness* a realidade virtual, representando uma mais valia para todos os engenheiros de software que lidam com grandes sistemas, mas também para iniciantes no mundo da programação, facilitando uma compreensão rápida dos sistemas de software.

Abstract

The constant increase in the complexity and size of software systems makes their understanding increasingly costly and time consuming, especially when this understanding is carried out by a programmer distant in the time or space of its original author. This difficulty constitutes an obstacle to the following maintenance processes, whether corrective or evolutionary, that can not be performed without the complete understanding of the system.

Although good practices in software development are used, a system that has considerable growth will not be able to avoid increasing its essential complexity, which shows that the systems may take a long time to be understood and evolved by users. Comprehensive analysis tasks are required which translates into high costs.

This work aims to mitigate this problem by allowing users to literally enter the software system in a live way using virtual reality. A tool was developed for Java systems that receive information about the static and dynamic analysis of the software system, using reverse engineering tools. The tool must, in real time and during the execution of the system, allow its visualization using visual metaphors. The interaction with the system in full execution is a key factor, following concepts of live programming that create a fluid feedback between the program and the programmer. This new approach of live software development resort to the virtual reality for the construction of an environment, through which it is possible to understand the system and visit it in an interactive and immersive way.

For the evaluation was carried out controlled experiments with experienced users and case studies of software systems in Java.

This contribution intends to innovate, adding to the liveness the virtual reality, representing an added value to all software engineers dealing with large systems, but also for beginners in the programming world, facilitating a quick understanding of software systems.

à memória da minha Mãe, Maria Fernanda

Agradecimentos

Antes de mais, gostaria de deixar um agradecimento à Faculdade de Engenharia da Universidade do Porto, nas pessoas de todos os docentes e não docentes que tive o privilégio de conhecer e aprender.

Um agradecimento especial ao meu orientador Professor Ademar Aguiar e ao meu coorientador Professor Rui Nóbrega pela partilha de conhecimento e de experiências, disponibilidade e, acima de tudo, sentido de humor.

À minha família, em especial à minha Mãe e ao meu Pai, que sempre foram uma fonte de suporte, coragem e união. Obrigado por terem acreditado em mim e pelo constante apoio às minhas decisões.

Aos meus amigos e colegas que me acompanharam durante os últimos cinco longos anos, em especial aos meus parceiros de laboratório, muito obrigado pela amizade, motivação, entreajuda e cafés.

Por fim, mas acima de tudo, à minha companheira Sara, comigo em todos os momentos, agradeço-te por tudo aquilo que partilhamos e por todo o apoio, força e coragem que me ofereceste.

Diogo da Silva Amaral

"*...o pensamento é impossível sem uma imagem.*"

Aristóteles

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

2D	Duas Dimensões
3D	Três Dimensões
RV	Realidade Virtual
UML	*Unified Modeling Language*
IDE	*Integrated Development Environments*
HMD	*Head-Mounted Display*
GUI	*Graphical User Interface*
API	*Application Programming Interface*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
URL	*Uniform Resource Locator*

Capítulo 1

Introdução

Grande parte do software criado hoje em dia é construído de forma incremental, a partir de um protótipo que evolui gradualmente através da adição de novas funcionalidades e detalhes. Com o avanço deste processo, o sistema escala e a produtividade é dificultada pelas tarefas de compreensão.

Os sistemas de software podem atingir elevada complexidade, muito devido à sua dimensão, chegando a milhões de linhas de código [AD07a]. Os engenheiros de software, nas tarefas de manutenção, seja evolutiva ou corretiva, quando acrescentam funcionalidades devem, em primeiro lugar, compreender o sistema [PBG03, VNP17b]. Uma tarefa de compreensão que pode apresentar muitos desafios, tais como, a escalabilidade e a complexidade [Wet10, MTRK14, EPP15].

Esta dificuldade pode ser reduzida aplicando a ideia de *liveness*, isto é, a capacidade de modificar um programa enquanto este se encontra em execução [Tan13]. Em conjunto com *live programming*, que permite ao utilizador receber imediato *feedback* das alterações realizadas ao código, o programa reexecuta continuamente à medida que é editado.

A visualização e a interação com o software num ambiente de realidade virtual podem aumentar a capacidade de compreensão recorrendo a metáforas visuais baseadas no mundo real, ao representar o software num contexto mais familiar que facilmente pode ser identificado pelo programador [VNP17b].

No ambiente virtual, com total imersão, será possível uma aproximação da realidade, criando uma simulação de um mundo real ou virtual no qual o utilizador pode mergulhar, tocar e sentir os objetos com a presença virtual nesse mundo 3D [SS17].

1.1 Contexto

A ficção científica há muito que nos habituou com uma programação fácil e acessível a todos, inclusive com experiências de imediato *feedback* na resposta a edições de código. Por exemplo, em 2008, no filme Iron Man[1], Tony Stark utiliza um ambiente holográfico interativo que lhe permite construir a sua armadura no momento em que a desenha [McD17].

Com o desenvolvimento da tecnologia, dos recursos computacionais e do interesse e aceitação pela população, abrem-se portas para projetos que podem ter um impacto significativo na engenharia de software.

Atualmente, é possível ter um computador do tamanho de um cartão de crédito[2] executar qualquer sistema operativo. Adicionalmente, ferramentas e plataformas que mudam a forma como visualizamos informação podem ser construídas em papel ou adquiridas a baixo custo[3].

Os computadores apresentam cada vez mais recursos e conseguem realizar tarefas que num passado próximo eram computacionalmente impossíveis. Exigir a um sistema de software que execute continuamente mesmo durante a edição de código dentro de um ambiente imersivo de realidade virtual, não é uma tarefa impossível. Prova disso são ferramentas recentes já criadas que se aproximam a esta ideia [FKH17, MGAN17, VNP17b].

Neste sentido, esta dissertação aborda o projeto de *live software development environment*, pretendendo implementar um ambiente virtual imersivo para a visualização e interação com sistemas de software.

Inserida num grupo de investigação em *live software development*, a dissertação pretende reduzir o esforço destinado a tarefas de compreensão e manutenção de sistemas. Como tal, os sistemas serão analisados, extraindo informação estática e dinâmica, guardando esses dados num repositório. Por sua vez, o ambiente virtual comunicará com o repositório construindo o sistema, recorrendo a metáforas visuais para uma maior familiarização do utilizador com o sistema. Este trabalho visa sobretudo sistemas de software Java, mas pode ser aplicado a outros tipos de sistemas orientados a objetos. Outras áreas, tais como *cloud computing* e *IoT*, encontram-se também a tentar aplicar a ideia de um ambiente virtual, no âmbito do grupo de investigação.

1.2 Motivação

O software é complexo e de grande dimensão, evolui e é intangível [Wet10].

[1] Iron Man IMDb, https://www.imdb.com/title/tt0371746/, Último acesso em: 05/11/2017
[2] Raspberry Pi, https://www.raspberrypi.org/, Último Acesso em: 07/11/2017
[3] Google Cardboard, https://vr.google.com/cardboard/, Último acesso em: 07/11/2017

Investigadores da área de engenharia de software enfrentam dificuldades na compreensão de software. Este problema, no momento do desenvolvimento, revela-se através do aumento do nível de esforço, do custo e do tempo despendido [DN14, PBG03, BWDL08, Wet10, KF17]. A necessidade de compreender e até evoluir um sistema de software rapidamente reforça a procura de novas soluções.

As representações visuais de software ajudaram a resolver este problema durante largos anos [MLM01]. No entanto, é necessário atualizar esta implementação, adaptando-a a sistemas mais exigentes.

Conseguir reduzir o esforço no ciclo de edição, compilação e teste, de forma a ser possível editar o código no mesmo momento que está a ser executado, apresenta-se como uma proposta interessante para resolver este problema, com vantagens significativas [McD16, Gol04b].

A aplicação de um *live software development environment* com interação facultada pela realidade virtual enquadra-se na solução a este problema porque permite criar um ciclo fluído de *feedback* totalmente imersivo, aumentando a produtividade. O objetivo da aplicação passa por mitigar estes problemas anteriormente apresentados.

Desta forma, utilizando a realidade virtual, será possível tornar a programação *live*, examinando visualizações com conteúdo espacial e temporal, para melhorar a descoberta da estrutura de software, aspetos do seu comportamento, recursos e eventuais *bugs*.

1.3 Problema

Esta dissertação procura explorar se **um ambiente de desenvolvimento que inclua funcionalidades de *liveness* e realidade virtual, ajuda na compreensão e na manutenção de sistemas de software**.

Com este intuito, são investigadas soluções de trabalhos relacionados dentro da comunidade científica que se aproximam ou que se apliquem a este contexto, tais como, conceitos, algoritmos ou ferramentas desenvolvidas. A presença de uma vasta literatura sobre os temas que são analisados nesta dissertação, demonstra o interesse na procura de soluções para estes problemas.

A Secção 3.3 explora com mais destaque a tese defendida, identificando a questão de investigação e respetivas hipóteses.

1.4 Objetivos

O objetivo desta investigação consiste em reduzir o esforço necessário para compreender e evoluir um sistema de software.

Neste sentido, é desenvolvido e apresentado no âmbito desta dissertação um motor de ambiente virtual. A designação de motor surge pelo motivo da ferramenta ter sido implementada na plataforma Unity3D[4], conhecida por motor de jogos. Este motor será alimentado por dados de um repositório e dará origem a um ambiente virtual.

[4]Unity3D, https://unity3d.com/pt, Último Acesso em: 20/06/2018

O ambiente virtual deve suportar um *feedback* imediato, onde alterações ocorridas na estrutura do sistema de software em análise sejam repercutidas visualmente. Adicionalmente, as ações do sistema, como invocações de métodos que aconteçam, devem ser explícitas ao utilizador para uma fácil análise e compreensão.

Neste sentido, o *live* deve sair reforçado, minimizando a latência das alterações no sistema de software e a correspondente visualização.

A implementação de metáforas visuais e espaciais com contextos familiares em 3D e com o cuidado na disposição dos objetos virtuais devem facilitar a resposta ao objetivo levantado. Adicionalmente e como mais valia, a aplicação de realidade virtual trará ao utilizador sensações reais, como a interação com objetos ou o movimento espacial, sem este sair do mundo virtual.

No final do desenvolvimento, o projeto deverá permitir a aplicação à visualização de sistemas orientados a objetos e deixar uma base sólida de funcionalidades que possam ser implementadas no futuro.

1.5 Estrutura do Relatório

Neste presente capítulo é apresentado o contexto, motivação, problema e objetivos da dissertação. O documento tem ainda mais 5 capítulos:

- Capítulo 2, Revisão Bibliográfica - revê as áreas mais significativas ao estudo de *live software development* explorando as principais tarefas no desenvolvimento de software, linguagens que representam visualmente a informação, visualização do software, programação com imediato *feedback* e imersão num ambiente virtual. É ainda realizada uma revisão das tecnologias que mais contribuíram, com contextualizações históricas e aplicações a casos reais.

- Capítulo 3, *Live Software Development* - contextualiza o projeto em desenvolvimento e fundamenta o problema de investigação, acrescentando os requisitos necessários, uma análise à arquitetura da abordagem proposta e dois casos de estudo.

- Capítulo 4, Ambiente Virtual - explora e justifica a estratégia de implementação desenvolvida, com apresentação de conceitos e algoritmos. A interface é também explorada, bem como, um pequeno paralelo com a revisão bibliográfica.

- Capítulo 5, Avaliação - definição e análise dos testes de utilizador em experiências controladas de forma a compreender a aceitação e a usabilidade da ferramenta.

- Capítulo 6, Conclusões e Trabalho Futuro - apresentam-se as conclusões do autor, bem como pormenores e funcionalidades que podem ser consideradas num desenvolvimento posterior deste projeto.

Capítulo 2

Revisão Bibliográfica

Este capítulo aborda e destaca os tópicos mais relevantes nas cinco áreas selecionadas que se consideram mais relevantes para a investigação proposta.

Através de definições, contextualizações históricas e abordagens realizadas pela comunidade científica é efetuada uma reflexão teórica para cada área.

Os exemplos de aplicação exploram as tecnologias e trabalhos relacionados já realizados ou em produção, focando em analogias que possam ser pertinentes para este projeto.

No final do capítulo é apresentado um resumo dos temas abordados e ainda uma análise de várias ferramentas.

2.1 Engenharia de Software

> "Software development is difficult because software is complex, the software production process is complex and understanding of software systems is a challenge."
>
> ——— [PBG03]
>
> "O desenvolvimento de software é difícil porque o software é complexo, o processo de produção de software é complexo e a compreensão dos sistemas de software é um desafio."

As linguagens de alto nível evoluíram de forma constante ao longo dos anos permitindo um desenvolvimento de software mais acessível. Fortran, C, Pascal, Smalltalk, Lisp, C++ e Java são um pequeno conjunto de linguagens que conseguiram ser bem-sucedidas devido ao seu suporte para abstrações que tornam o processo de programação mais natural e com a possibilidade de ser mapeado, por um compilador ou interpretador, para um código executável mais eficiente. Se os programadores conseguissem manipular de forma direta abstrações de alto nível e ver os resultados dessa ação imediatamente, então, menos tempo seria dedicado à mecânica e mais às atividades intelectuais [Gol04a].

Com recurso ao guia SWEBOK [SBF14], é possível definir e separar o estudo da engenharia de software em vários subtópicos.

Requisitos de software. Esta área de conhecimento debruça-se na elicitação, análise, especificação e validação dos requisitos de software, bem como a gestão dos requisitos durante todo o ciclo de vida do produto desenvolvido. Esta tarefa pode considerar-se crítica na medida que a viabilidade dos projetos é vulnerável à qualidade desta área.

Design. Visto do lado do processo, *design* é uma atividade do ciclo de vida do software onde os requisitos são analisados de forma a produzir uma descrição da estrutura interna. O *design* do software descreve a arquitetura, a sua organização em componentes e respetivas interfaces. Nesta fase, os engenheiros de software produzem modelos de solução para posterior implementação.

Construção. O termo construção de software refere-se à criação detalhada de software funcional através de uma fusão de codificação, verificação, testes unitários, testes de integração e *debug*.

Os fundamentos para a construção refletem a intenção de reduzir complexidade, antecipação à mudança, verificação e reutilização.

Manutenção. Os esforços no desenvolvimento de software concentram-se na entrega de um produto que satisfaça os requisitos apresentados pelos utilizadores. Consequentemente, o produto de software deverá mudar ou evoluir. No decorrer do desenvolvimento do projeto, as imperfeições e erros são descobertos, os ambientes de operação mudam e novas necessidades dos utilizadores surgem. A fase de manutenção é uma parte integral do ciclo de vida do software apesar da pouca atenção que recebe inicialmente. O paradigma dos projetos abertos, *open source*, voltou a atenção para a questão da manutenção dos artefactos do software desenvolvidos por outros programadores. Desta forma, o objetivo é modificar software existente, preservando a sua integridade.

Os objetivos incidem em melhorar a eficiência e a qualidade de um projeto de desenvolvimento e reduzir o seu custo.

Neste seguimento, é também importante realçar o papel da arquitetura do software. Este campo de estudo, que se preocupa especificamente com o *design* e com os níveis de arquitetura, deve decidir perante alternativas de *design* e descrever propriedades de alto nível de sistemas complexos. O seu objetivo é conceber o sistema de software a construir [Agu03].

Em suma, todas as etapas que levam à construção do software são relevantes e poderão ser significativas no momento da compreensão.

2.2 Linguagens Visuais

O contexto das linguagens visuais, densamente estudado nas últimas três décadas, teve como pioneiro S.K. Chang [CIL86] que previu a dificuldade em definir estas linguagens, considerando que o termo tem um significado diferente para cada pessoa [ESW17].

De facto, ainda hoje, a comunidade de investigadores nesta área não encontra consenso numa definição exata. Como tal, sendo imprescindível tirar uma conclusão quanto à definição deste tópico, Erwig *et al.* [ESW17] estudaram, durante vinte anos, artigos da comunidade científica de linguagens visuais e após uma extensa análise concluíram que estas são linguagens cuja estrutura sintática pode ser classificada em *Graph*, *Partition* ou *Icon*. Na sua generalidade, consistem numa linguagem formal com sintaxe visual e semântica [PDB17].

As linguagens visuais têm o objetivo de ajudarem novos programadores a expressarem com maior facilidade os seus algoritmos e ainda para os programadores mais experientes serem mais produtivos, conseguindo simplificar a organização do programa com representações visuais [Tan90].

> "Visual representations promise the possibility of more natural and rapid understanding by human programmers and users of the workings of programs than is possible with conventional textual representations of programs."
>
> ——— [Tan90]
>
> "Representações visuais prometem a possibilidade de uma compreensão mais natural e rápida por parte dos programadores humanos e utilizadores sobre o funcionamento de programas do que é possível com representações textuais convencionais."

O software é construído através de direta manipulação das primitivas e dos operadores da linguagem gráfica. Por sua vez, essas linguagens foram construídas usando uma variedade de paradigmas, como por exemplo, o *dataflow* [Gol04a].

2.2.1 *Liveness*

No seguimento da pesquisa e desenvolvimento de linguagens visuais, Steven Tanimoto [Tan90] propôs a linguagem visual VIVA, Visualization of Vision Algorithms, em 1990, para o processamento de imagem, sublinhando a importância do *feedback* do sistema em relação ao utilizador. O objetivo do seu trabalho consistiu em fornecer uma ferramenta de ensino para estudantes e ainda servir como uma base de software que possa facilitar o trabalho de plataformas de pesquisa em interações humano-computador. VIVA foi projetada para fornecer uma maior qualidade de *feedback* ao programador durante a construção do programa [CLR96, Hil92, Tan13].

É notório que grande parte dos trabalhos desenvolvidos neste período, em particular por Tanimoto, tinham como objetivo permitir uma melhor compreensão dos sistemas, construindo imagens mentais dos programas e das suas execuções, principalmente para estudantes [Tan03].

A importância do trabalho desenvolvido por Tanimoto, através da linguagem visual VIVA, sobressai por ser o primeiro a definir o termo *liveness* [Tan90], como um atributo das linguagens visuais, referindo-se à capacidade de modificar um programa enquanto este se encontra em execução [BAW98, Tan13]. *Liveness* é a característica que permite a edição de código enquanto o programa permanece em constante execução e conduz até à definição de *live programming*, que será examinado no Capítulo 2.3.

A utilização de *liveness* em ambientes de programação é uma forma potencialmente valiosa de obter *feedback* imediato. Como tal, apresentam-se algumas motivações para o seu uso [Tan13]:

- Diminuição da latência entre uma ação de programação e a reação no sistema, aquilo que o utilizador observa do seu efeito na execução do programa.
- Experiência de visualização pela audiência é controlada pelas ações do programador.
- Simplificação da tarefa de atribuição de responsabilidades quando ações de programação provocam novos comportamentos, como um *bug*.
- Apoio à aprendizagem e investigação.

Tanimoto classificou os sistemas de programação visual quanto ao grau com que apresentam *live feedback* ao programador. A escala definida pelo *liveness* foi proposta com 4 níveis e categoriza o imediatismo do *feedback* semântico que é fornecido automaticamente durante a programação [Tan90].

No nível 1, o nível mais baixo de *liveness*, não existe qualquer *feedback* semântico ao utilizador sobre o programa, apenas uma representação visual deste como forma de ajudar o utilizador a documentar ou entender o programa. Tratando-se de um nível informativo.

No nível 2, o utilizador consegue obter *feedback* semântico sobre uma parte do programa, mas não é fornecido automaticamente. Existe uma representação visual que consiste na especificação real para o computador sobre como a computação deve ser realizada. Se isso for um fluxograma, então este deve ser executável e conter os detalhes suficientes para que o programa seja completamente especificado. Neste caso, o nível considera-se informativo e significativo.

No nível 3, o *feedback* semântico é fornecido, automaticamente, no momento em que o utilizador executar uma edição no programa, todos os valores afetados no ecrã são automaticamente atualizados. Neste nível, uma ação por parte do utilizador desencadeia uma reação por parte do sistema. Já não é necessário que o utilizador envie, de forma explicita, o fluxograma para a execução. Esta ação permite assegurar a consistência do estado do ecrã e do estado do sistema, quando o utilizador é o único a poder editar o sistema. Trata-se de um nível que acrescenta a responsividade aos pontos já presentes.

Por fim, no nível 4, considerado o máximo possível de *liveness* na proposta inicial de Tanimoto, reflete-se um sistema onde as representações visuais são contempladas pelo nível 3 e acrescenta outros eventos, onde o sistema atualiza constantemente e de forma concordante com o estado do programa. Por exemplo, sinais de relógio do sistema e cliques do rato ao longo do tempo, garantindo que todos os dados em exibição refletem com precisão o estado atual do sistema. O computador não irá esperar, mas sim manter o programa em execução, alterando o comportamento do programa à medida que as modificações acontecem. É o nível que acrescenta o *live* à sua definição [BAW98, CLR96, Tan90, Tan13, KRB18].

Conseguir implementar métodos que suportem *liveness* até ao seu quarto nível pode-se revelar vantajoso na medida em que o programador terá um aumento do *feedback* recebido. Um sistema de software que consiga suportar por completo *liveness* será o meio para o programador editar um programa enquanto este permanece em constante execução, sem interrupções e permanentemente atualizado.

Em 2013, com a universalidade do *live programming*, em análise no capítulo 2.3, a sua comum incorporação em *Integrated Development Environments*, IDEs, e em ferramentas de aprendizagem, os níveis de *liveness* foram estendidos, acrescentado mais dois [Tan13].

No novo nível 5, o computador não executa apenas o programa e responde, mas também prediz a próxima ação do programador, criando um conjunto de múltiplas alternativas e executando uma ou mais versões que previu recorrendo, por exemplo, a máquinas virtuais separadas. O ambiente de programação faz um pequeno planeamento para descobrir a próxima versão do programa que o programador possa ter interesse em selecionar. É o nível de *liveness* taticamente preditivo.

O nível 6 acrescenta a possibilidade de uma maior inteligência na inferência das intenções do programador. Não só fará uma simples previsão tática, como no nível 5, mas também previsões estratégicas. Essa previsão seria capaz de cobrir o comportamento desejado de largas unidades de software. O sistema conseguirá sintetizar o comportamento do programa, rapidamente, através da análise de outros programas ou de uma grande base de conhecimento.

A Figura 2.1 reflete este esquema de 6 níveis de *liveness*, após atualização.

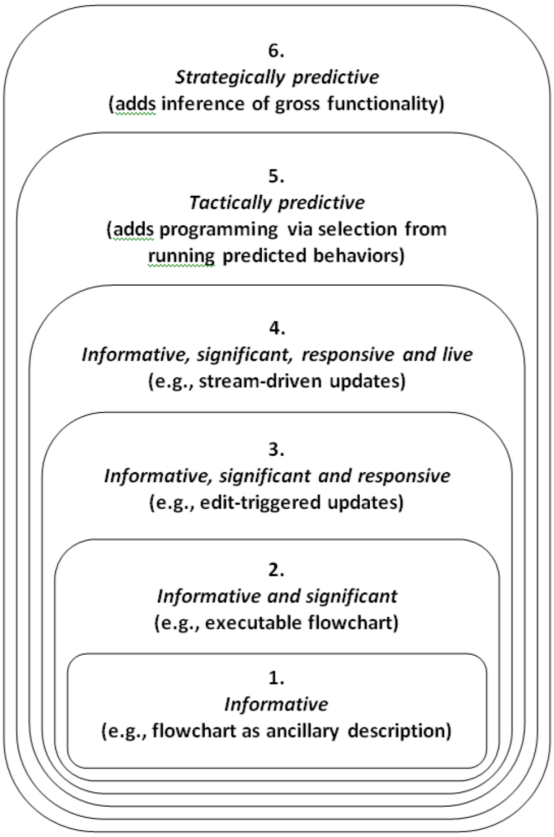


Figura 2.1: Níveis de *liveness* em sistemas de programação visual definidos por Tanimoto [Tan13].

> "Higher levels of liveness would allow for additional kinds of feedback to the user at the descriptive, significant, responsive, or live levels."
>
> ——— [Tan90]
>
> "Níveis mais altos de *liveness* permitiriam tipos adicionais de *feedback* para o utilizador nos níveis descritivo, significativo, responsivo ou *live*."

Daniel Hils [Hil92], um ano após a publicação da linguagem criada por Tanimoto, investigou 15 linguagens visuais orientadas à programação, entre as quais se encontra a linguagem VIVA. Através da construção de *data flows* mostrou as potencialidades destas ferramentas à programação visual, agrupando as linguagens quanto ao seu domínio de aplicação [ESW17].

Principais Críticas

O conceito de *liveness*, no seu nível máximo, dificilmente se aplicaria a todos os casos e ambientes de programação. No entanto, apresenta-se como potencialmente importante para vários tipos de programação [Tan13].

As críticas mais comuns apresentam-se em situações em que o programa executa em alguns milissegundos e termina ou se o programa for executado por um longo período de tempo e a

edição que for realizada estiver numa parte do programa onde a execução não passará novamente. A utilização de *liveness* nestes casos aparenta ser desnecessária [Tan13].

No entanto, com a introdução de um ciclo repetidor, até ser terminado pelo utilizador, em ambas as situações, é praticável aplicar *liveness* e interagir com o sistema. Desta forma, será possível executar o ponto modificado e avaliar o seu correto funcionamento [Tan13].

2.2.2 Discussão

Linguagens visuais permitem uma maior facilidade na manipulação da informação, aumentando a produtividade e simplificando a organização do problema. Pelo menos, foi esse o intuito que levou à densa investigação em linguagens visuais e à implementação de vários ambientes visuais de desenvolvimento, como VIVA.

Liveness representa a capacidade de modificar um programa enquanto este se encontra em execução e os seus níveis classificam a quantidade de *feedback* que o programa é capaz de transmitir ao utilizador. Atualmente, várias ferramentas e *frameworks* integram imediato *feedback*, principalmente na *web* [KRB18].

As linguagens visuais, assim como *liveness*, tentam resolver um problema semelhante, o de tornar a programação mais fácil, facilitando a compreensão daquilo que o programa está a fazer ou daquilo que deveria estar a fazer.

2.3 *Live Programming*

A trivial atividade de um programador resume-se a escrever um trecho de código, compilá-lo e finalmente observar e testar o seu comportamento em *runtime*. Esta atividade, por sua vez, pode estar inserida num ciclo de repetições, levando o programador a repeti-la até conseguir implementar o pretendido.

Se fosse possível visualizar a forma como o programa se comporta, a produtividade durante a edição de código e *debugging* poderia melhorar de forma significativa [BFdH+13].

Live programming cria imediato *live feedback* do comportamento do programa, em tempo de execução, enquanto o código fonte é alterado [KRB18, LL13, McD07]. Ou seja, é possível mudanças no código fonte dos sistemas de software em plena execução sem necessitar de reiniciar a execução depois das alterações [SF16].

Isto permite diminuir o tempo gasto neste processo e garante que os programadores têm um pleno conhecimento das alterações realizadas e o seu impacto na execução do sistema de software [LL13].

Um sistema que suporte *live programming* por completo irá permitir que o programador edite o programa enquanto este está em execução, sem qualquer interrupção percetível [Tan13].

Alguns investigadores consideram que *live programming* é uma melhoria ao *debug* pela possibilidade de fazer a edição do código fonte sem interromper a constante execução do programa. No entanto, alterações ao programa são difíceis de concretizar e quando acontecem, normalmente,

a execução está suspensa, recorrendo a *breakpoints* e a *single-stepping*. Estas alterações durante o modo *debug* consistem, na maioria, em trocas nos valores dos dados e não em modificações de código [Tan13].

> "Live programming makes programming easier by re-executing a program continuously during editing."
>
> ——— [McD13]
>
> "A programação *live* torna a programação mais fácil por reexecutar um programa continuamente durante a edição."

Chris Hancock [Han03] considera que *live programming* devia ir além do *feedback* contínuo e assemelhar-se a uma mangueira de água. Numa situação destas é possível atingir o alvo direcionando a mangueira para cima ou para baixo. Como existe um fluxo contínuo de água a sair pela mangueira, qualquer reação é repercutida aproximando-se ou afastando-se do alvo. Esta analogia permite concluir que pequenas alterações na entrada de código levam a pequenas mudanças na saída de execução e assim manter um maior controlo do processo [McD13, McD16].

Principais Desafios

O maior desafio apontado à utilização de *live programming* refere questões ao nível da performance. A capacidade dos sistemas em que os recursos podem não ser suficientes para conseguir uma edição e uma execução a acontecer em paralelo, acrescentando ainda uma ação de compilação-ligação-carregamento (*compile-link-load*) desencadeada por uma edição, que pode acontecer tão rapidamente que nem seja percetível pelo utilizador. Neste caso, apesar dos recursos computacionais dos anos 80 ou 90 terem desincentivado o uso de *live programming*, os computadores atualmente conseguem lidar com maior agilidade estas questões [Tan13].

Relacionado com a semântica do programa, a edição pode levantar inconsistências entre a versão em execução e a versão atualizada do programa ou modificações concorrentes. Nestes casos, devem ser criados mecanismos de deteção e até correção, antes da utilização de *live programming*. Por exemplo, sistemas como o Eclipse *for* Java[1], permitem a edição *live* do programa em execução. Para lidar com esta questão, o sistema possui ferramentas de retroceder (*undoing*), sendo que qualquer erro semântico é facilmente corrigido [Tan13].

2.3.1 Linguagens *Live*

As linguagens dinâmicas promovem um uso fácil e prático da escrita de código focado em programação de alto nível, agilizando o ciclo de edição-compilação-teste. As linguagens *live* conseguem superar as linguagens dinâmicas, aumentando a facilidade na escrita [McD07].

[1] Eclipse IDE for Java Developers, https://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/keplersr1, Último Acesso em: 08/02/2018

Uma linguagem *live* suporta *live programming* e fornece aos programadores contínuo e responsivo *feedback* sobre os efeitos das suas alterações no código. O objetivo é permitir ao programador escrever menos código e capacitá-lo com *feedback* semântico responsivo, permitindo poupar tempo em serviços como a compilação do código [McD07].

Por sua vez, as linguagens visuais, abordadas na Secção 2.2, foram projetadas com o intuito de aumentar a simplicidade de uso, em relação às linguagens dinâmicas. Como tal, baseiam-se em modelos de programação simples, na maioria declarativos, enfatizando a reutilização de componentes do código. Estes modelos simples também permitem a realização de *live programming* graças ao *feedback* imediato e responsivo que é transmitido aos programadores [McD07]. A vantagem em relação às linguagens dinâmicas resumem-se à capacidade de escrever código mais rapidamente [Pet95].

Neste sentido, uma linguagem *live* pode ser considerada uma linguagem textual ou uma linguagem visual, desde que suporte *live programming* através de um modelo de programação simples [McD07].

2.3.2 *Hot Swapping*

Hot swapping é uma técnica usada por compiladores e sistemas em tempo de execução para trocar novas funções e valores num programa enquanto este está em plena execução [Cza13].

Live programming está relacionado com o conceito de *hot swapping* do código, onde ambos permitem que o código de um programa em execução seja atualizado sem perder o seu estado e contexto. No entanto, *live programming* preocupa-se com o desenvolvimento do sistema de software, enquanto que o *hot swapping* do código preocupa-se com a atualização dos programas já implantados. Ainda, o primeiro concentra-se em conexões navegáveis entre código e execução, enquanto o último concentra-se em tempo e robustez, respetivamente [BFdH+13].

A existência de *hot swapping* em linguagens dinâmicas permite substituir o código para que a sua reexecução ocorra de acordo com a nova versão do código em oposição à versão antiga. Ambientes como LISP, Smalltalk e Erlang suportam *hot swapping*, onde o código do programa pode ser atualizado sem reiniciar [McD07]. Na linguagem Java, *hot swapping* admite a redefinição da classe previamente carregada e é suportado pela JVM Tool Interface [VBAM09a, VBAM09b].

Comparando com *live programming*, o sistema *hot swapping* suportado por linguagens dinâmicas é pouco consistente. Nomeadamente, o efeito da edição de código no comportamento do programa não é demonstrado até o código ser reexecutado. Como exemplo, no ambiente Smalltalk editando a declaração "w fill: red"para "w fill: blue", a cor não mudará imediatamente. Isto acontece porque o *hot swapping* em linguagens dinâmicas substitui o código para que a sua reexecução ocorra de acordo com a nova versão do código em oposição à versão antiga [McD07].

2.3.3 IDEs

Na sua maioria, os IDEs têm a capacidade de inspecionar uma computação e modificá-la. Adicionar-lhes *liveness* é uma forma de melhorar [Tan13].

Esta característica, por exemplo, no Eclipse para programação em Java, apresenta uma variedade de recursos que interagem com o código, permitindo deixar o programador mais informado sobre o estado do sistema de software [Tan13].

A capacidade de fornecer contínuo e responsivo *feedback* sobre a estrutura léxica e sintática do código está presente em muitos IDEs. No entanto, grande parte das linguagens visuais *live*, como VIVA, linguagem proposta por Tanimoto [Tan90] e abordada no Capítulo 2.2, vão além disso, fornecendo *live feedback* sobre como o programa é executado à medida que o código é editado [McD13].

Integrated Development Environments para linguagens como LISP, Smalltalk, Java e C# suportam funcionalidade de *"fix-and-continue"*, onde o programador é capaz de modificar o código sem reiniciar o processo de *debug*. Infelizmente, esta funcionalidade raramente é capaz de fornecer um *feedback* responsivo, como conseguido recorrendo ao *live programming* [BFdH+13].

IDEs como Eclipse, IntelliJ e Netbeans e editores como Emacs, Vim e Atom fornecem rápido *feedback* quando o programa muda. Esta resposta pode ser apresentada num formato de *highlighting* sintático, erros e avisos, entre outros. Os programadores contam com a resposta praticamente instantânea para encontrar e resolver rapidamente os problemas [KEV16]. No entanto, os IDEs não eliminam o ciclo edição-compilação-execução no desenvolvimento de um programa [Gol04a], apesar de já conseguirem apresentar altos graus de *liveness* [KRB18].

2.3.4 Aplicações de *Live Programming*

Os sistemas de *live programming* apresentam grande influência no mundo da programação e a sua aplicação leva ao florescer de novas ideias.

Programação por Exemplo

Ambientes de *live programming* permitem atualizações do programa em tempo de execução para as mudanças realizadas no código-fonte. Ao separar a renderização do tratamento de eventos, o *output* da aplicação pode ser atualizado em reação às mudanças de código para fornecer *feedback* visual ao programador. Desta forma, Schuster e Flanagan [SF16] introduzem o conceito de *live programming by example*, como uma forma de alterar o código de um programa em execução por manipulação direta da interface do utilizador. O conceito surge da junção de *live programming* e de programação por exemplo, *programming by example*, uma abordagem já estudada para tornar a programação mais simples, principalmente para não programadores, onde é apresentado ao sistema exemplos dos dados que o programa deve processar e usar esses exemplos durante o desenvolvimento do programa [Mye86].

Desenvolvimento de Linguagens

A aplicação de *live programming* ao desenvolvimento de linguagens de programação é também um tópico investigado. O objetivo é facultar rápido *feedback* ao programador da linguagem no momento em que acontecem mudanças nessa linguagem, permitindo a experimentação com o *design* e desenvolvimento da linguagem. Uma linguagem é especificada e a partir daí podem

existir implementações derivadas da linguagem. Quando uma especificação é alterada, os programadores querem obter uma resposta rápida sobre as mudanças na especificação da linguagem, mas também para todos os programas que a usam. Com esse *feedback* praticamente instantâneo, os programadores podem explorar rapidamente as consequências das mudanças introduzidas e reagir de acordo. Assim, obtém-se um novo tópico denominado *live language development* que aborda, principalmente, o *liveness* dos ambientes de desenvolvimento de linguagens [KEV16].

Spreadsheets

Edwards *et al.* [ECW16] apresenta como único caminho a seguir a revisão das etapas do ambiente original de *live programming*: as *spreadsheets*. As *spreadsheets* ajudam, principalmente, não programadores a resolverem problemas de pequena escala. Caso seja aplicada a mesma estratégia, é possível oferecer uma experiência totalmente *live* e simplificada de programação que seja realmente útil. O modelo das *spreadsheets* pode ser rapidamente aprendido por não programadores o que permitirá resolver vários problemas.

Neste sentido, um largo número de pessoas surgem familiarizadas com o tema de *live programming* no contexto de *spreadsheets*. Nesta situação, os dados e fórmulas podem ser editados e o efeito dessas edições pode ser imediatamente percecionado. Para além das *spreadsheets*, linguagens baseadas principalmente em modelos de programação declarativos, incluindo muitas linguagens visuais, já conseguem fornecer experiência de *live programming*, apesar de serem limitadas na expressividade em programas mais complexos [BFdH+13].

As funcionalidades das *spreadsheets* suportam o nível 3 de *liveness* [BAW98], níveis explorados por Tanimoto.

2.3.5 *Live Software Development*

> "The advantage of a fully live software development environment is that the amount and rate of feedback to the programmer is maximized."
>
> ——— [Tan90]
>
> "A vantagem de um ambiente de desenvolvimento de software totalmente *live* é que a quantidade e a taxa de *feedback* para o programador são maximizadas."

O tema *live software development*, desejado por Tanimoto [Tan90], foi estudado e implementado por Kenneth Goldman [Gol04a] num determinado contexto. Goldman investigou o desenvolvimento de software de forma *live* utilizando uma implementação de classes dinâmicas em Java que não requerem modificação da máquina virtual da linguagem.

No desenvolvimento de software *live* orientado a objetos, surgiu a necessidade de aceder e modificar as definições das classes. Linguagens como Java fornecem suporte de acesso a informações do tipo em tempo de execução, mas a definição da classe é fixa em tempo de compilação, ou seja, não existem meios para mudar o seu estado [Gol04b].

A abordagem foi considerar classes verdadeiramente dinâmicas, que podem ser modificadas em tempo de execução de forma a ser possível a alteração da sua interface e implementação. O

processo consistiu em criar versões paralelas unificadas das classes Java imutáveis e das classes dinâmicas mutáveis.

Neste seguimento, os objetivos do trabalho de Goldman [Gol04b] foram conseguir:

- Classes dinâmicas: a representação interna, interface e implementação da classe podem ser modificada dinamicamente em tempo de execução sem recompilação.
- *Liveness*: modificações das classes dinâmicas afetam as instâncias de forma imediata.
- Transparência: classes dinâmicas têm acesso total às classes compiladas e suas instâncias.
- Interoperabilidade: instâncias de classes dinâmicas e classes compiladas devem referir-se mutuamente.
- Integridade hierárquica: classes dinâmicas devem ter subclasses de classes compiladas e devem sobrescrever os seus métodos.
- Polimorfismo: instâncias de classes compiladas devem chamar métodos em instâncias de classes dinâmicas polimorficamente.
- Portabilidade: a linguagem de programação subjacente e o seu sistema de tempo de execução não são modificados.

Como tal, Goldman investigou como suportar a noção de classe dinamicamente modificável, cujos membros (campos, métodos e construtores) podem ser adicionados, removidos e modificados em tempo de execução. No seu estudo "Live Software Development with dynamic classes"[Gol04b], Goldman caracteriza *dynamic classes* como classes Java, exceto no ponto em que as definições da classe podem ser alteradas em tempo de execução, sendo aqui os seus efeitos imediatos nas instâncias existentes. A Figura 2.2 exemplifica o meio para modificar a definição de uma classe em plena execução do programa.

O seu ambiente de desenvolvimento, definido como JPie [Gol03], *Java programmer's interactive environment*, foi implementado num projeto na Washington University in St. Louis iniciado em 1999. Trata-se de um ambiente de programação interativo projetado para tornar o poder do desenvolvimento de software em Java acessível a um público mais amplo e que suporte a construção de *live software* através da manipulação direta de representações gráficas de programas [Bra04].

JPie integra desenvolvimento de *live software* na medida em que todos os aspetos do comportamento do programa podem ser modificados enquanto o mesmo está em execução, eliminando os ciclos edição-compilação-teste. Quando uma classe é modificada no ambiente JPie, todos os objetos são afetados pela mudança. Como o editor gráfico suporta manipulação direta das unidades semânticas do programa, o seu ambiente de execução reage imediatamente às modificações causadas, durante a execução do programa. Não se trata de uma nova linguagem de programação, mas sim uma abordagem num ambiente de programação em que esse ambiente está intimamente consciente da estrutura do software em desenvolvimento, suportando um grande nível de interatividade e aproveitando o poder de uma linguagem orientada a objetos [Gol04a].

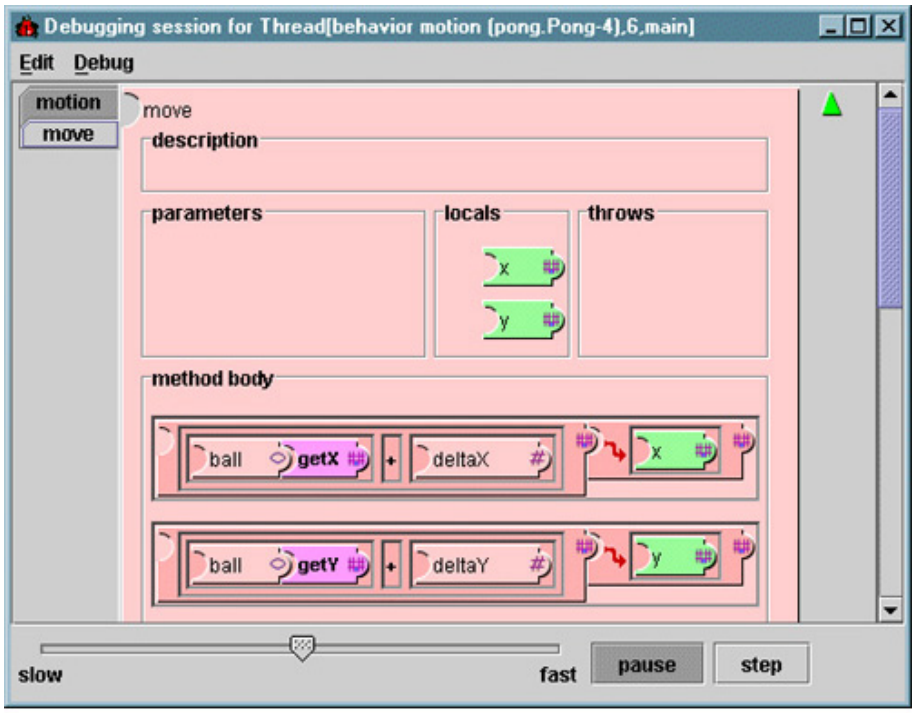


Figura 2.2: Modificação da definição de classe Java durante execução do programa no ambiente JPie[2].

De forma a tornar o desenvolvimento de software imediato e tangível, a ferramenta JPie fornece representações gráficas de abstrações das linguagens de programação. Grande parte das operações, como declaração e uso de variáveis e métodos, são realizadas através de *drag-and-drop*, o que previne a ocorrência de erros de sintaxe [Gol04a].

Usualmente, IDEs tradicionais incluem uma interface gráfica, GUI, que permitem visualizar componentes gráficos rapidamente. Em oposição, o ambiente JPie fornece uma representação visual de todo o programa, incluindo não só a GUI como também a definição das classes. Ou seja, o programador nunca precisa de escrever código [Gol04a].

Java é uma linguagem importante no ensino da programação. Dadas as suas características foi usada por Goldman porque fornece acesso detalhado a informação do tipo em tempo de execução, permitindo a implementação das classes dinâmicas e, consequentemente, ao *live software development*. O principal intuito da ferramenta JPie foi facilitar o ensino da programação [Gol03, Gol04a].

> "(Java) provides clean type-safe support for standard object-oriented programming techniques and abstractions and simplifies programming by providing a garbage collected heap and a comprehensive exception-handling mechanism."
>
> ——— [Gol04a]

[2]JPie, https://jpie.cse.wustl.edu/menus/pictures_menu.htm, Último Acesso em: 10/12/2017

> "(Java) fornece um suporte seguro para as técnicas e abstrações de programação orientada a objetos e simplifica a programação fornecendo um *garbage collected heap* e um mecanismo abrangente de tratamento de exceção."

Goldman apresentou as classes dinâmicas como um meio para suportar a modificação do programa como *live* durante o desenvolvimento de software e ainda apelou ao poder do desenvolvimento de software orientado a objetos para os programadores experientes e para os estudantes que iniciam os seus passos no estudo da ciência da computação.

Live software development surgiu para permitir o aumento da produtividade através de uma abordagem mais fluída no desenvolvimento do software. É assim possível fazer alterações no software em plena execução e sem a necessidade de o terminar ou interromper [Gol04a].

2.3.6 Discussão

A maior parte da "ação"de programar acontece na cabeça do programador, no raciocínio e abstrações. *Live programming* impõe-se como uma alternativa, evoluindo o processo desde a edição do código até à execução, criando uma experiência mais fluída, onde o comportamento do programa é observado imediatamente quando a edição ocorre [McD17].

O principal intuito é a capacidade de transmitir imediato *feedback* face a todas as alterações provocadas pelo programador.

Da evolução de *live programming* surgem diversas aplicações do seu contexto, respondendo a problemas e influenciando projetos. Como por exemplo, os IDEs estão cada vez mais preparados para lidar com situações de *feedback* instantâneo.

A junção de *live programming* e *liveness* revela-se interessante para a indústria, onde várias empresas já assumiram projetos com estas vertentes, e também no mundo académico com várias conferências internacionais e *workshops* [KRB18].

Goldman apresentou classes dinâmicas como um mecanismo de suporte ao desenvolvimento de software de forma *live* onde a programação é realizada através de *drag-and-drop*, evitando erros sintáticos e facilitando a tarefa de programar, podendo desta forma atingir um maior número de programadores, incluindo os que possuem menor experiência. JPie capacita a realização de *live programming* e adiciona uma interface de representação visual de todo o programa, sendo o resultado um mecanismo interessante para aplicação no ensino da programação.

2.4 Visualização de software

Parte do senso comum que "uma imagem vale mais que mil palavras". Conseguir transmitir informação através de imagens é uma característica da visualização de software que permite transformar o software em algo mais tangível, através de representações desse sistema.

> "We define Software Visualization as the use of the crafts of typography, graphic design, animation, and cinematography with modern human-computer interaction technology to facilitate both the human understanding and effective use of computer

> software."
>
> ——— [PBS93]
>
> "Definimos Visualização do Software como o uso do artesanato de tipografia, design gráfico, animação e cinematografia com a moderna tecnologia de interação humano-computador para facilitar, tanto a compreensão por humanos, quanto o uso eficaz de software de computador."

Visualização de software é um subdomínio de *reverse engineering* e a definição atribuída por Price *et al.* [PBS93], apesar de antiga, datada de 1993, continua a ser usada por investigadores na área [Wet10, NS10, PdQ06].

Diehl [Die07] considera que a área de visualização de software está interessada na visualização da estrutura, comportamento e evolução do software.

- Estrutura: as visualizações concentram-se nas partes e relações estáticas do sistema, podendo ser extraídas diretamente do código fonte, sem a necessidade de executar o programa.
- Comportamento: o seu intuito é relativo à informação dinâmica que pode ser extraída da execução do programa.
- Evolução: visualizações focam-se nas mudanças que o sistema de software está sujeito ao longo da sua vida útil.

São consideradas a análise estática, a análise dinâmica e as modificações implementadas na sequência de evoluções.

Por outro lado, as ferramentas usadas na visualização devem abordar três fases: aquisição, análise e visualização de dados. [Die07]

Esta dissertação seguiu, como principal exemplo para a área de visualização de software, a tese de doutoramento de Wettel [Wet10], pelo extenso e importante trabalho desenvolvido nesta área. A investigação levada por Wettel *et al.* apenas se concentra na visualização da estrutura e na evolução dos sistemas de software, ignorando o comportamento, ou seja, a informação que pode ser extraída durante a execução do programa.

O objetivo da aplicação de visualização de software remete à capacidade de ajudar na compreensão dos sistemas de software e de melhorar a produtividade do processo de desenvolvimento desses.

> "Visualization is more than a method of computing. Visualization is the process of transforming information into a visual form, enabling users to observe the information. The resulting visual display enables the scientist or engineer to perceive visually features which are hidden in the data but nevertheless are needed for data exploration and analysis."
>
> ——— [Ger94]
>
> "Visualização é mais que um método de computação. Visualização é o processo de

transformar informação numa forma visual, permitindo que os utilizadores observem as informações. O resultado da visualização permite que o cientista ou engenheiro perceba funcionalidades visualmente ocultas nos dados, mas que, no entanto, são necessárias para a exploração e análise de dados."

2.4.1 Contextualização Histórica

Os primeiros passos na fundação de visualização de software, anterior a 1980, foram dados recorrendo a diagramas. Estas estruturas foram intensamente investigadas por serem uma forma de compreensão dos programas. Mais precisamente, Haibt [Hai59], em 1959, criou um sistema que permitia o desenho de *flowcharts* automaticamente, a partir da linguagem Fortran ou linguagens *Assembly*, como demonstra a Figura 2.3. Três anos depois, Knuth [Knu63] desenvolveu um sistema semelhante que integrava a documentação com o código fonte e podia gerar *flowcharts* automaticamente, como demonstra a Figura 2.4 [PBS93, Wet10].

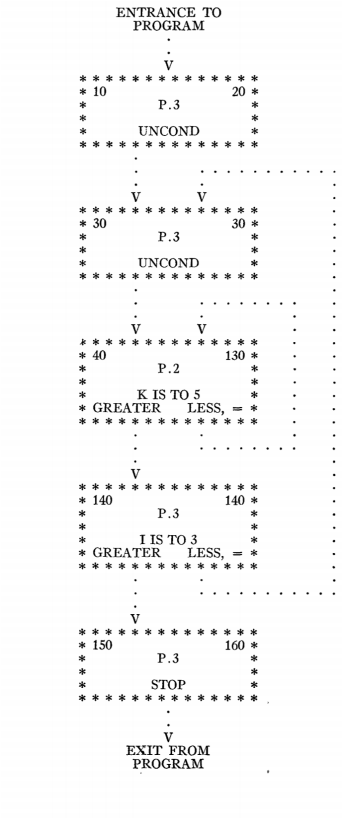


Figura 2.3: *Flowchart* obtido automaticamente pelo sistema desenvolvido por Haibt [Hai59].

Já na década de 1980, grande parte das investigações em visualização de software foram direcionadas para o comportamento do programa (*behavior*), usado, principalmente, para fins educacionais. O exemplo mais popular de uma animação do comportamento do programa foi o filme

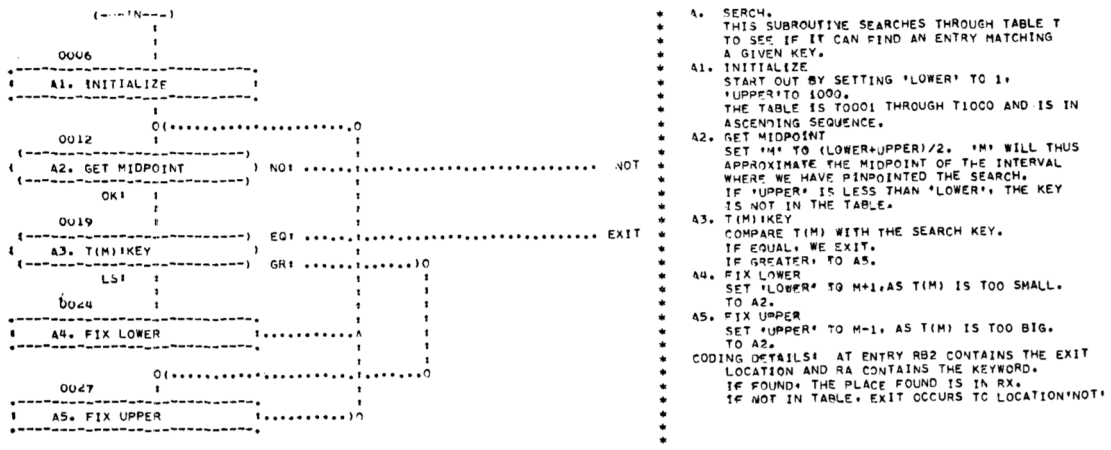


Figura 2.4: Exemplo de *flowchart* e de *flow outline* para pesquisa binária desenvolvido por Knuth [Knu63].

de Baecker [Bae98], de 1981, "Sorting Out Sorting"[3], que permitiu fornecer um auxílio visual que ajudava a entender alguns dos algoritmos de classificação mais importantes, como *quick-sort*, *bubble-sort* ou *shell-sort*.

A introdução de visualização em 3D recorrendo a realidade virtual foi iniciada por Reiss em 1995 [Rei95] e Young e Munro em 1998 [YM98]. O aspeto principal foi a exploração de representações de software para a compreensão dos programas em realidade virtual.

Em 2001, Bassil e Keller [BK01] fizeram um levantamento da utilidade de um conjunto de técnicas e ferramentas recentes de visualização de software que se tornaram disponíveis para o suporte das atividades de análise, modelação, teste, *debug* e manutenção. Neste inquérito, as representações 3D, o *layout* e as técnicas de realidade virtual apresentaram o maior valor de inutilidade e de falta de conhecimento do tema ou não aplicação ao caso. Isto deve-se meramente ao facto de estes aspetos funcionais serem pouco comuns nas ferramentas de visualização de software na época, sugerindo assim um elevado grau de incompreensão. De notar que 63% dos participantes concordaram que a ferramenta de visualização era de fácil utilização e 68% afirmaram obter resultados mais vantajosos na sua utilização.

Ainda nesta pesquisa, Bassil e Keller identificaram os principais benefícios dessas ferramentas, tais como, poupança em tempo e dinheiro, melhor compreensão do software, aumento da produtividade e qualidade, gestão da complexidade e procura de erros [Die07].

No ano seguinte, um inquérito [Kos02] questionou mais de uma centena de investigadores em manutenção de software, *reengineering* e *reverse engineering*, concluindo que 40% dos questionados consideram a visualização de software absolutamente necessária no seu trabalho e outros 42% consideram importante, mas não crítico [Die07].

[3]Sorting Out Sorting IMDb, www.imdb.com/title/tt0210301/, Último Acesso em: 15/01/2018

2.4.2 Representação e Visualização

O conceito de visualização deve ser dividido em dois níveis distintos, representações e visualizações. As representações preocupam-se com a representação gráfica de um único componente. As visualizações são uma coleção ou configuração de representações individuais e outras informações que compõem um componente de nível superior. No entanto, uma representação pode ser uma visualização, em contextos diferentes [YM98].

A representação deve transmitir o máximo de informação mantendo uma baixa complexidade visual. A sua aparência deve distinguir diferentes componentes com contraste e assemelhar componentes idênticas. Esta formação deve estar preparada para gerar visualizações automáticas tentando manter este processo flexível [YM98].

A visualização deve transmitir o máximo de informação com o mínimo de complexidade visual. Deve existir um *layout* compreensível e a navegação deve ser fácil e intuitiva, com o mínimo de desorientação. O sistema deve estar preparado para permitir uma visualização simplificada para utilizadores iniciados e outra mais extensa para os mais experientes [YM98].

2.4.3 Visualização Estática e Dinâmica de Software

Os aspetos estáticos do software representam informação que pode ser obtida sem o programa estar em execução, mas que é válida para todas as possíveis execuções [VNP17b].

O ponto de partida da visualização estática consiste, normalmente, no código fonte ou no código máquina do programa. Algumas visualizações baseiam-se apenas na análise sintática ou na análise do fluxo de dados [Die07].

A maioria das abordagens propostas e investigadas ao longo dos anos abordam apenas aspetos estáticos do software [VNP17b]. Diagramas como *flowcharts* são um exemplo de visualização estática e o ponto de partida em vários estudos [Mye86].

Os aspetos dinâmicos são resultado da análise dinâmica da execução do programa e a informação obtida é válida apenas para essa execução do programa [VNP17b].

O método principal de aquisição de dados consiste na instrumentação do código fonte. Em animações é comum a aplicação deste método, identificando eventos que sejam relevantes. Podem e devem ser utilizados filtros para controlar e reduzir a quantidade de informação disponível dos eventos durante a execução [Die07]. A aplicação de um sistema de *pause* seria pertinente.

As visualizações conseguidas dinamicamente baseiam-se na acumulação, projeção espacial ou animação usando objetos gráficos simples, como caixas, bem como gráficos computacionais 3D, criando um ambiente virtual mais complexo. Os dados são adquiridos em tempo de execução, levando a um acesso direto à memória do programa [Die07].

A visualização dinâmica é cada vez mais possível de atingir graças aos avanços dos recursos computacionais, sendo que o maior desafio nesta visualização de informação é a escalabilidade. Os sistemas são cada vez mais complexos e de grande dimensão [Wet10].

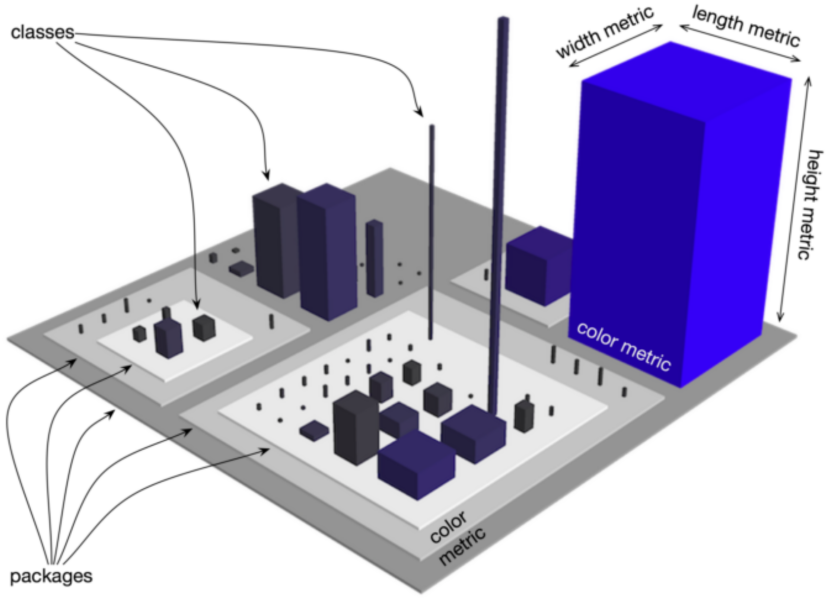


Figura 2.5: Exemplo demonstrativo da metáfora da cidade aplicada na ferramenta CodeCity [Wet10].

2.4.4 Metáforas Visuais

Uma metáfora visual define o mapeamento de um modelo de um programa - um nível baixo de abstração - para uma imagem - um elevado nível de abstração - especificando o tipo de visualização [PBG03]. As metáforas visuais introduzem conceitos familiares e mais naturais para os utilizadores, fornecendo um bom ponto de partida para obter uma maior compreensão da visualização [YM98, AD07b].

Das metáforas visuais utilizadas na visualização de software, possivelmente a mais popular é a metáfora da cidade, que foi explorada no passado por vários investigadores [PBG03, Wet10, VNP17b, AD07b, KM00, PPDSF12, LC07, FKH15, WWF+13].

Esta tipologia de representação de software foi também a explorada por Wettel *et al.* [Wet10]. O processo foi iniciado com o desenho das linguagens visuais, tema explorado no Capítulo 2.2, com o intuito de fazer o mapeamento entre o domínio de origem (a cidade) e o domínio de destino (o sistema de software). O sistema é assim representado como uma cidade, sendo os pacotes do sistema como distritos da cidade e as classes como edifícios. De forma a conseguir obter mais informação da análise da cidade, esta é enriquecida com métricas de software, ou seja, as propriedades físicas dos artefactos da cidade (cor e dimensões) refletem um conjunto de propriedades mensuráveis dos artefactos do software. A Figura 2.5 demonstra esta representação.

A metáfora é aplicada por Wettel *et al.* em três contextos distintos: compreensão do programa, análise da evolução do software e na avaliação da qualidade do projeto. Aplicando esta abordagem a contextos de sistemas de software reais, Wettel *et al.* implementaram a ferramenta CodeCity baseada na metáfora da cidade.

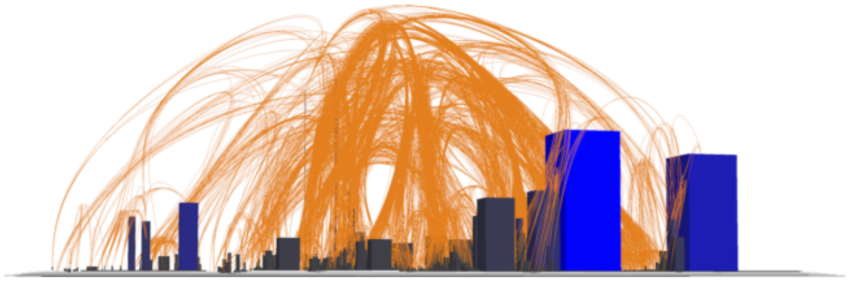


Figura 2.6: Visualização de *bundled edges* da experiência da ferramenta CodeCity [Wet10].

As relações entre classes e a sua comunicação são ainda um desafio em visualização de software. Um par de classes podem partilhar mais de uma dezena de diferentes relações, por exemplo, definições de herança de classe, invocação de métodos, acessos a atributos, etc. *Holten's bundled edges* [Hol06] é uma técnica que permite agilizar este problema ao seguir rotas semelhantes e assim reduzir a complexidade. Aplicando esta solução de *bundled edges* até ao 3D, é possível conseguir várias perspetivas do ambiente gerado e assim compreender melhor as ligações. A Figura 2.6 demonstra as *bundled edges*, num grande aglomerado de linhas, das invocações, numa perspetiva de linha do horizonte, aplicada ao ArgoUML[4] como caso de estudo. Com isto pode ser levantado um problema de escalabilidade, devido à grande quantidade de informação que é manipulada. Felizmente, o constante aumento do poder computacional existente permite que esta visualização seja mais exequível [Wet10].

Wettel *et al.*, na sua investigação com a ferramenta CodeCity, concluiu que a metáfora da cidade facilita a criação de eficientes visualizações do software. Nesse sentido, deve ser dado especial destaque ao algoritmo de alocação dos objetos, que contribuiu para essa eficiente visualização. Com CodeCity é proposto um *layout rectangle packing* para lidar com elementos de diferentes dimensões na sua alocação, formando estruturas retangulares. O algoritmo está interessado em responder a três situações:

1. não desperdiçar em demasia a estrutura da cidade,
2. refletir as relações estabelecidas e
3. ter em conta as dimensões e proporções dos edifícios.

Numa última análise, foi verificado se a experiência dos utilizadores tinha influência nos benefícios que podiam ser retirados da ferramenta. As conclusões são interessantes, na medida em que os utilizadores iniciais, ou seja, menos experientes, conseguem um desempenho mais consistente em termos de exatidão do que utilizadores experientes. Neste caso, não existe uma curva de aprendizagem. Os utilizadores conseguem atingir bons resultados, sem ser necessário muito treino. Por outro lado, em termos de tempo de conclusão, esta abordagem revela-se vantajosa em particular para os profissionais da indústria [Wet10].

[4]Welcome to ArgoUML, http://argouml.tigris.org/, Último Acesso em: 15/01/2018

A análise da imagem representada pela metáfora permite obter informação quanto aos aspetos estáticos e dinâmicos do programa. Do ponto de vista estático, o tamanho, a cor e a estrutura dos edifícios podem facilitar a identificação de áreas de código que precisam de *refactoring*, por exemplo. Em oposição, dinamicamente, animações podem ser criadas, como a introdução de movimentos que se deslocam entre dois métodos, realçando o *trace* entre eles [PBG03].

2.4.5 Aplicações de Visualização de Software

Em 2012, Petrillo *et al.* [PPDSF12] investigou 52 ferramentas de visualização de software analisando a sua situação atual.

É notória a quantidade de trabalhos de investigação que conduziram a aplicações nesta área, realçando o interesse pela utilização de metáforas visuais e de visualização em 3D. CodeCity, referido anteriormente é um exemplo desses.

3D UML

Unified Modeling Language (UML) é uma linguagem usada para a modelação de sistemas de software, onde, tal como a visualização de software, transmite informação através de representações visuais. É uma linguagem baseada em diagramas usada para descrever os aspetos estáticos, por exemplo, diagramas de classes, diagramas de pacotes ou diagramas de componentes, e os aspetos dinâmicos, por exemplo, diagramas de sequência [Wet10]. A Tabela 2.1 relembra estes pormenores com a diferenciação dos vários diagramas.

Tabela 2.1: Diferentes tipos de diagramas UML de acordo com a versão 2.5.1 [Gro17].

Tipo	**Diagrama**
Estrutura	Classe, Objeto, Componente, Estrutura Composta, Pacote, Implementação, Perfil.
Comportamento	Caso de Uso, Atividade, Máquina de Estado.
Interação	Sequência, Comunicação, Temporização, Visão Geral de Interação.

O UML tornou-se uma linguagem comum ao fornecer ferramentas que permitem criar representações visuais do software interativamente. Gogolla *et al.* [GRR99, RG00] juntaram os aspetos estáticos e dinâmicos numa única vista, estendendo o UML para a terceira dimensão. Neste projeto, denominado 3D UML, é possível demonstrar o comportamento no contexto dos aspetos estruturais, em vez de criar múltiplos diagramas para mostrar isso mesmo. Uma visualização a três dimensões é importante por conseguir representar cenários complexos. Isto permite ao utilizador percorrer o modelo e examiná-lo de diferentes pontos de vista [Die07].

O primeiro passo dado por Gogolla *et al.* nos diagramas de classes, objetos e sequência adotando uma metáfora natural de primeiro plano permitiu desenvolver o modelo 3D. Estas implementações foram criadas através de *Virtual Reality Modeling Language* [Die07]. A Figura 2.7 representa um diagrama de sequência em 3D onde duas esferas se deslocam do emissor até ao recetor. A lista seguinte esclarece quanto aos diagramas implementados.

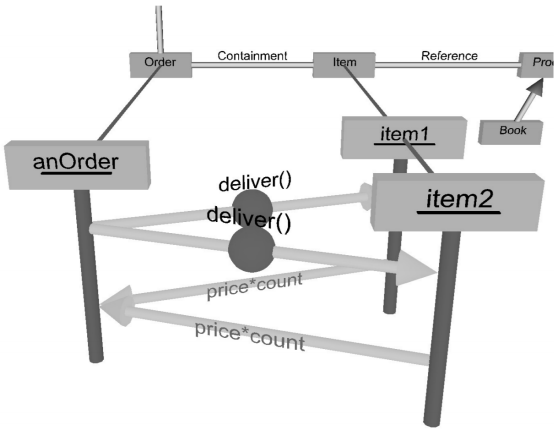


Figura 2.7: Diagrama de sequência em 3D com animação ao longo das linhas de vida [RG00].

- Diagrama de Classes: classes de maior relevância são desenhadas em primeiro plano, destacando-se. Na necessidade de acesso a classes do segundo plano, um clique na mesma move-a para um plano de destaque de forma a ser examinada pelo utilizador. Isto permite ter várias perspetivas do mesmo diagrama. Existe uma animação suave que acompanha todas estas alterações.

- Diagrama de Objetos: podem ser utilizadas várias formas para representar os objetos. Todos os objetos pertencentes à mesma classe têm formas ou cores semelhantes, de modo a uma rápida identificação. São exibidas ligações entre os objetos de diferentes classes.

- Diagramas de Sequência: as animações introduzidas nestes diagramas mostram, através de bolas, a troca de mensagens entre o remetente e o recetor. O problema da sobreposição de setas para mensagens enviadas simultaneamente e de setas que cruzam linhas de vida também pode ser resolvido pelo *layout* 3D. Infelizmente, projetar essas visualizações 3D em monitores 2D leva a outros tipos de sobreposições e cruzamentos.

A vantagem de utilização da versão em três dimensões revela-se na análise dos diagramas, conseguindo vários pontos de vista na modificando da perspetiva. Esta vantagem aumenta na compreensão de diagramas de sistemas complexos [GRR99].

UML-City

Uma versão do UML em 3D desenvolvida por Lange e Chaudron [LC07] resultou em diagramas enriquecidos com métricas de software, obtendo conjuntos de visualizações baseadas na metáfora da cidade. Este projeto denominado UML-City conseguiu reduzir o esforço necessário em tarefas de compreensão [Wet10]. A Figura 2.8 demonstra um exemplo de utilização desta ferramenta.

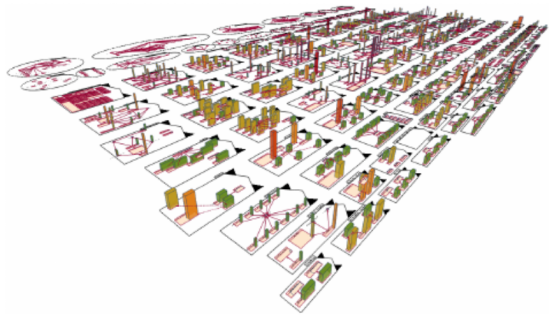


Figura 2.8: UML-City: Diagramas UML enriquecidos com métricas do software [LC07].

De facto, a utilização de UML consegue uma abstração do sistema de software, mas pode revelar dificuldade na tarefa de compreensão de grandes sistemas em monitores devido à representação da informação ser apenas em duas dimensões [MLM01]. Recorrendo a uma visualização 3D e às suas vantagens na visualização é possível mitigar estas dificuldades.

2.4.6 Discussão

A visualização de software é uma técnica usada em diversas ferramentas de manutenção de software [YM98]. Destaca-se pela capacidade de sintetizar grande quantidade de informação [VNP17b].

O principal objetivo da visualização é transmitir informação. Esta informação deve ser transmitida de uma forma compreensível, eficaz e fácil de lembrar. No entanto, apenas uma pequena parte dos sistemas de visualização de software existentes suportam o desenvolvimento de grandes sistemas e com equipas numerosas, o que é comum na indústria de software [Die07].

A aplicação de metáforas visuais enriquecidas com métricas de software permite criar um ambiente mais familiar ao utilizador, facilitando a compreensão do mesmo. A metáfora da cidade foi a mais abordada neste documento pelo seu destaque em inúmeras investigações da área.

Na visualização 3D devem ser considerados alguns tópicos, como a representação gráfica, a quantidade de informação visual a mostrar, a navegação, correlações entre a visualização e o sistema a visualizar, automação e interatividade. A utilização de ambientes imersivos na visualização revela-se uma importante vantagem [YM98]. A sua aplicação real é analisada no Capítulo 2.5, abordando um conjunto de aplicações ao tema.

2.5 Realidade Virtual

A realidade virtual (RV) é uma técnica usada para a criação de simulações do mundo real ou virtual, aplicando a teoria da imersão num espaço virtual 3D, no qual a visão estereoscópica, o sentido da audição, o sentido do tato e o sentido do olfato são muito semelhantes aos do mundo real, dentro de uma determinada área [SS17]. No capítulo anterior, no âmbito de visualização

de software, já foram explorados ambientes em 3D. No entanto, não existia qualquer imersão no sistema, nem recurso aos restantes sentidos do corpo humano, para além da visão.

Segundo a escala *reality–virtuality continuum*, neste capítulo, a variação aproxima-se continuamente do ambiente completamente virtual [MTUK95].

A realidade virtual é capaz de mudar a forma como muitos aspetos do quotidiano são experimentados e usados, por exemplo, através de sistemas de interação háptica [CNdSR16], e em tarefas simples como jogos, páginas web, filmes, entre outros [SS17]. Mesmo distantes, utilizadores espalhados pelo mundo podem juntar-se num ambiente virtual para comunicarem [VNP17b].

Desde o século XIX, quando Charles Wheatstone[5] propôs a ideia da estereoscopia, que permitiu alcançar a sensação de relevo na fusão de duas imagens, grandes mudanças aconteceram [SS17].

Atualmente, presenciamos fortes investimentos nesta área. Empresas como o Facebook que adquiriu a Oculus, a Valve que tem trabalho em colaboração com a HTC e a iniciativa de levar o sistema a qualquer pessoa movida pelo Google Cardboard, demonstra que a realidade virtual é uma área com futuro e que deve ser trabalhada e explorada [KRB18, SS17].

Um rápido avanço no campo da realidade virtual permitiu um crescimento no desenvolvimento de vários dispositivos, como computador poderosos e *smartphones* com *displays* de alta densidade e recursos de gráficos 3D.

A abordagem seguida pela RV permitiu melhorar a experiência de utilização. Monitores em 2D controlados por ratos não conseguem criar uma experiência tão imersiva e natural [FKH15].

A modelação de RV refere-se à modelação de objetos 3D, à sua interação com o mundo virtual e real, ao efeito da interação nos objetos, etc., de forma a simular os mesmos [SS17].

2.5.1 Realidade Virtual aplicada à Engenharia de Software

> "VR has opened the door for software engineers to create systems that increase efficiency and enable previously impossible experiences."
>
> ——— [EPP15]
>
> "A RV abriu a porta para engenheiros de software criarem sistemas que aumentem a eficiência e possibilitem experiências antes impossíveis."

A realidade virtual em conjunto com a engenharia de software aplicadas ao *live coding* e à revisão de código apresentam benefícios para o programador, colocando de parte a mera interação através de um teclado e de um rato. RV permite que os engenheiros de software entrem nos ambientes que anulam o *delay* entre a ação do programador e a visualização do resultado dessa sua ação. Este rápido *feedback* já foi implementado para ecrãs bidimensionais [BAW98], mas a utilização de RV permite estender esta funcionalidade para visualizações a três dimensões. [EPP15]

[5] Sir Charles Wheatstone, http://www.stereoscopy.com/faq/wheatstone.html, Último Acesso em: 28/12/2017

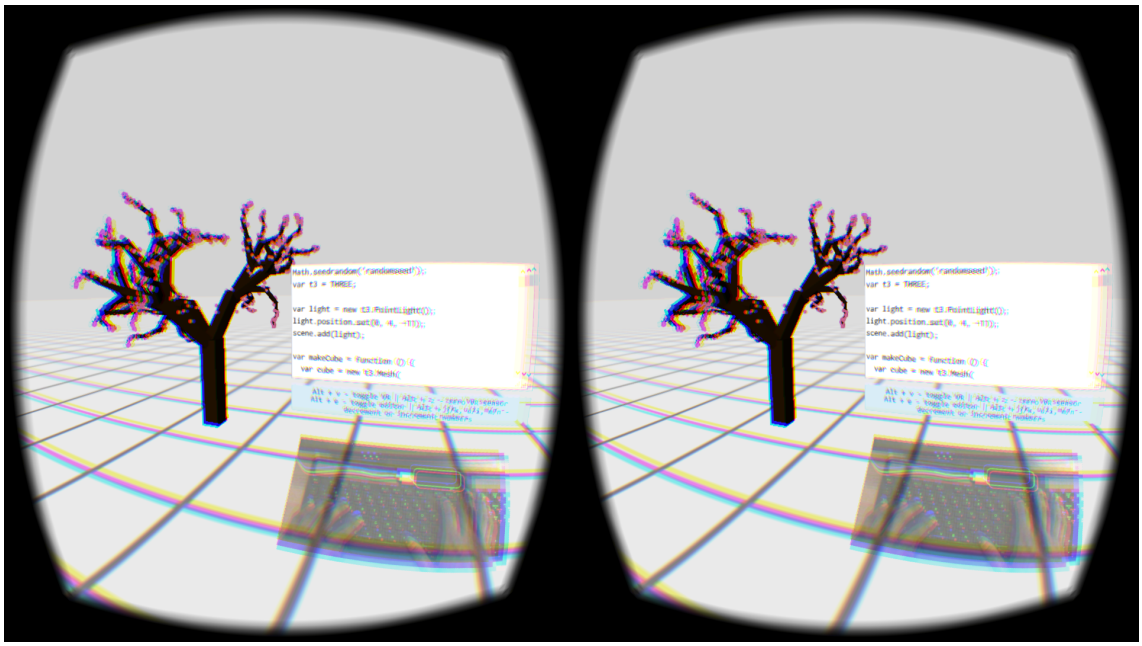


Figura 2.9: Escrita de código no mundo virtual utilizando o teclado físico [Pei17].

Elliott *et al.* [EPP15] desenvolveram soluções que respondem às atividades *live coding* e de revisão de código, com o desenvolvimento das ferramentas RiftSketch e Immersion, respetivamente.

Live Coding

Num ambiente de *live coding* para RV, os utilizadores manipulam a cena 3D num ambiente de realidade virtual. À medida que o código é modificado, o mundo é atualizado instantaneamente [VNP17b].

Como exemplo, a ferramenta designada de RiftSketch apresenta um simples editor de texto posicionado à frente do utilizador no mundo virtual. A capacidade de atualização *live* e de animação da cena permite ao utilizador manipular o estado da cena 3D no código, criando comportamentos para os objetos na cena. A animação acrescenta uma sensação de imersão, melhorando qualquer imagem em 2D. A ferramenta RiftSketch é poderosa relativamente ao *feedback*. A escrita do código e os seus efeitos no ambiente virtual permitem uma rápida experiência, permitindo a perceção imediata de erros, com possível correção, sem recorrer ao passo da recompilação. Estes benefícios tornam-se mais evidentes numa cena 3D [VNP17b].

Para permitir a escrita de código no ambiente virtual, estando o utilizador completamente imerso no sistema, aplicou-se uma abordagem que naturaliza esse processo. Uma câmara montada no HMD para o teclado físico e respetiva projeção no ambiente virtual habilita o utilizador a escrever texto no ambiente virtual recorrendo ao teclado. Revela-se utilidade nesta funcionalidade para a introdução de comentários, anotações ou para próprio refinamento do código. A Figura 2.9 demonstra a agilização deste processo na escrita de código num ambiente virtual [EPP15].

Revisão de Código

Ainda no contexto da investigação anterior, com a constante necessidade de revisão do código por parte dos engenheiros, a ferramenta Immersion fornece um ambiente imersivo para esse fim. Recorrendo ao poder cognitivo do espaço, a ideia é agrupar fragmentos em diferentes secções no chão do mundo virtual [VNP17b]. As secções são pacotes do sistema e a sua cor indica a quantidade de modificações que ocorreram nesse pacote, facilitando a revisão.

A colaboração remota permite que vários programadores se juntem num espaço virtual habilitado para o *live coding* o que poderá levar a encontrar a solução mais rapidamente. Os pensamentos e ideias de cada utilizador podem ser inseridos como anotações na secção correspondente no sistema.

Ao utilizar a realidade virtual para a visualização do software, é possível aproveitar um maior conjunto de perceções dos utilizadores e ainda a capacidade de navegar através do ambiente virtual [MLM01].

2.5.2 Aplicações de Realidade Virtual

Atualmente, ainda são poucas as ferramentas de visualização de software que usam a realidade virtual para tarefas de compreensão [MGAN17].

No entanto, ao longo dos últimos anos, várias investigações foram realizadas com a consciência que a aplicação de realidade virtual seria vantajosa no processo de compreensão do código de um sistema de software.

Software World

Na introdução de soluções 3D em visualização de software, principalmente nas últimas duas décadas, destaca-se o trabalho de Knight e Munro [KM00] que descreveram e implementaram uma metáfora visual da cidade em 3D para realidade virtual, que representa uma análise estática dos sistemas de software Java, denominada de Software World [FKH17, Wet10].

Nesta visualização é possível encontrar o sistema representado pelo mundo, ficheiros de código como cidades, classes são distritos e métodos são edifícios. As propriedades físicas, como a cor dos edifícios e das janelas destes representam conceitos do sistema, neste caso o facto de serem públicos ou privados, como demonstra a Figura 2.10 [BWDL08, Wet10, PBG03].

No entanto, foi apontada falta de discussão sobre a escalabilidade do sistema Software World. O mapeamento visual também sofreu críticas porque sistemas com elevado número de classes levaria a cidades irrealistas [WL07].

A implementação Software World assemelha-se à apresentada por Wettel *et al.*, CodeCity, abordada no Capítulo 2.4. Nota-se que Software World é limitado para sistemas Java, enquanto que CodeCity é baseado num meta-modelo independente de linguagem que permite a aplicação em sistemas que utilizem várias linguagens. Em oposição, Software World pretende aplicar realidade virtual na sua visualização.

Figura 2.10: Metáfora Software World que representa o ficheiro de código como uma cidade [KM00].

ExplorViz

A ferramenta ExplorViz explora as visualizações do software recorrendo à metáfora da cidade usando realidade virtual e interações baseadas em gestos. Este último aspeto representa uma importante abordagem. Utilizando uma câmara de reconhecimento de gestos, como Microsoft Kinect v2, é realizado um rastreamento do corpo. O direto mapeamento das mãos no ambiente virtual é utilizado em detrimento dos controladores físicos. Isto deve-se a experiências em que os utilizadores com os controladores, após algumas utilizações e em movimentos contínuos, tentam manipular o modelo como se estivessem a utilizar uma abordagem de mapeamento direto [FKH15].

Através dos gestos das mãos é possível fazer translação, rotação, *zoom* e seleção dos objetos. Através de um salto é efetuado *reset*, voltando à posição inicial [FKH15].

ExplorViz fornece uma abordagem de visualização do software hierárquica e multinível com o intuito de melhorar a compreensão do sistema [FKH17].

A ferramenta utiliza técnicas de análise dinâmica, fornecendo uma visualização *live* do rastreamento da comunicação em grandes sistemas [FZ17].

CityVR

Já em 2017 surgiu a ferramenta CityVR [MGAN17] que permite uma visualização interativa, implementando a técnica da metáfora da cidade através de realidade virtual num ambiente 3D imersivo (I3D) para aumentar o envolvimento do programador em tarefas de compreensão de software.

CityVR recorre a um HMD e destaca-se da implementação de Software World por não utilizar um monitor como meio para exibir a visualização, tecnologia limitada disponível na altura. Em relação à ferramenta CodeCity, esta oferece interações baseadas numa configuração do ecrã do computador, enquanto na ferramenta CityVR os programadores interagem com visualizações através de movimentações pelo espaço físico disponível e selecionando classes usando controladores manuais para obter mais informações sobre estas, como experimentado na Figura 2.11

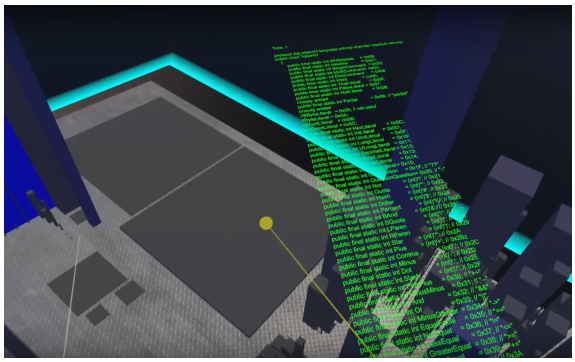


Figura 2.11: Experiência de utilização da ferramenta CityVR utilizando os controladores de mãos.

[MGAN17].

CityVR permite aos programadores obterem uma visão geral do sistema de software enquanto estão imersos nesse mesmo sistema. Os utilizadores interagem com o sistema usando os seus braços ou até todo o próprio corpo. A visualização da cidade representante do software é escalada para a mesma dimensão do espaço real disponível para o utilizador usar, sendo que a distância caminhada pelo utilizar na realidade é a mesma no mundo virtual. Esta navegação requer algum treino prévio e habituação. A busca pela interação mais física, caminhando, movendo o corpo e saltando, permite retirar aos programadores longas horas passadas sentados nas cadeiras nas tarefas de compreensão do código [MGAN17].

O trabalho desenvolvido em CityVR teve como maiores pilares e objetivos as técnicas de seleção, formas de interação e meio ambiente. Com isto foi possível discutir questões de *design* de arquitetura e observar a experimentação em ambiente controlado da ferramenta desenvolvida.

VR City

A ferramenta VR City pretende visualizar três tipos de aspetos com grande nível de detalhe: aspetos estáticos, aspetos dinâmicos e a evolução. Para tal, as métricas de software devem ser visualizadas para cada método, mantendo a representação das classes como edifícios, recorrendo à metáfora da cidade. Neste caso, os edifícios podem ter uma grande variedade de formas o que permite acelerar a identificação das interfaces, classes e métodos abstratos, *code smells*, entre outros, por parte dos utilizadores, criando uma visão geral sobre todo o sistema [VNP17b].

Nesta versão da metáfora da cidade, as classes são representadas por edifícios e os métodos são representados por andares. Cada andar consiste num número de blocos (cubos) igual ao valor da métrica de software escolhida, por exemplo, linhas de código do respetivo método.

Recorrendo ao *room-scale tracking* do utilizador num ambiente real com equipamento que o permita, é possível caminhar livremente na área de jogo, com os movimentos reais refletidos no ambiente virtual. No entanto, esta área é tipicamente pequena, dificultando a tarefa de percorrer grandes cidades no ambiente virtual. Neste seguimento, foi adicionada a funcionalidade de teleporte dos utilizadores, facilitando a deslocação no ambiente virtual [VNP17b].

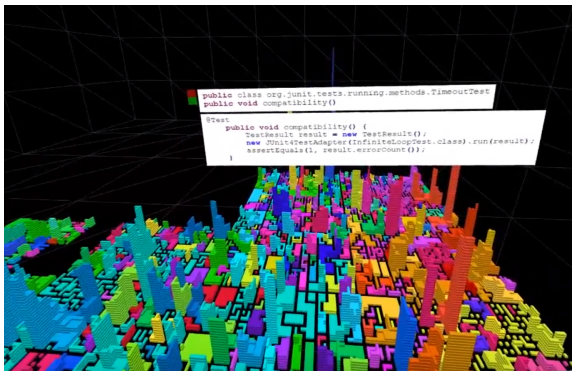


Figura 2.12: Experiência de utilização da ferramenta VR City com destaque da visualização do código [VNP17a].

A visualização da evolução é conseguida recorrendo a uma animação que demonstra quais as classes e métodos que sofreram alterações numa sequência de *commits* previamente fornecida à ferramenta. A evolução do sistema ao longo do tempo demonstra como o código ou a cidade, relativamente à metáfora visual, é afetada pelos programadores. Cada *commit* é visualizado, por ordem, especificando o seu autor e o local da edição. Esta animação pode ser interrompida para análise das modificações com mais detalhe [VNP17b].

A irregularidade dos edifícios, devido ao algoritmo de adição dos blocos, e a visualização do código durante a experiência virtual é refletida na Figura 2.12. A estrutura do software é representada por um grafo onde as classes são nós e as suas dependências são ligações. O algoritmo de *layout* deriva de uma técnica de visualização *space-filling*[6]. A cidade resultante sustenta uma construção 3D definida por curvas de Hilbert[7]. Graças a este *layout*, sendo uma mais valia, a identificação de *code smells* é beneficiada.

Esta ferramenta permite aos utilizadores explorar sistemas de software orientados a objetos em ambiente de realidade virtual de uma forma intuitiva na navegação e interação com os objetos da cidade e na navegação com o código fonte [VNP17b].

A avaliação da ferramenta é efetuada utilizando os sistemas JUnit 4 e JHotDraw 7.0.6 como casos de estudo [VNP17b].

EvoSpaces

O projeto denominado EvoSpaces [AD07a] explora a aplicação da metáfora visual da cidade representando a arquitetura e métricas dos sistemas de software complexos num mundo virtual 3D.

Esta visualização adapta ficheiros e classes como edifícios, distribuídos por distritos, em 3D. Os métodos são definidos como pessoas a trabalhar dentro dos edifícios. O sistema permite navegação, na forma de voo ou caminhar pelo sistema, e interações [AD07b]. O modo de interação

[6] *Space-filling curve*, https://en.wikipedia.org/wiki/Space-filling_curve, Último Acesso em: 20/01/2018

[7] *Hilbert curve*, https://en.wikipedia.org/wiki/Hilbert_curve, Último Acesso em: 20/01/2018

consiste na alteração do modelo de visualização, por exemplo dos objetos [AD07a].

A análise dinâmica demonstra a comunicação entre classes e ficheiros durante a execução, um *trace*, definido no programa como modo noite sobre a cidade. O modo dia corresponde à análise estática do código [FKH17, AD07b].

O código fonte do Mozilla foi usado como caso de estudo, pela complexidade do sistema, conseguindo uma técnica 3D para abordar a tarefa de compreensão deste sistema [AD07a].

2.5.3 Discussão

A manutenção e o desenvolvimento de software dependem da análise e compreensão da estrutura e comportamento do sistema em estudo. Tanto a análise como a compreensão são processos que consomem esforço e requerem uma grande quantidade de recursos. Estes custos podem ser reduzidos usando ferramentas que suportem formas mais eficientes de visualizar o software [VNP17b].

Como tal, a realidade virtual apresenta-se com uma área de investigação na resolução destes problemas. A imersão num ambiente virtual 3D adicionada à aplicação da metáfora visual da cidade tornam possível reduzir o esforço na visualização e compreensão dos sistemas. Os exemplos analisados exploram, em larga escala, a implementação da metáfora da cidade, pelas suas características já abordadas e demonstram essa redução do esforço.

Os ambientes virtuais imersivos requerem habituação para a sua utilização. No entanto, após várias iterações, o utilizador consegue absorver a técnica necessária para aumentar a velocidade e a capacidade de utilização do ambiente.

Realidade virtual apresenta-se como uma área em investigação e com forte investimento que tem sido explorada no contexto do desenvolvimento de software, com importante intuito na compreensão desses sistemas de software.

2.6 Sumário

Com o objetivo de analisar a implementação de um ambiente de *live software development* foi realizada uma investigação da literatura, destacando as áreas de engenharia de software, linguagens visuais, visualização de software, *live programming* e realidade virtual.

As tecnologias ocupam um papel importante nesta revisão. Como tal, a tabela 2.2 foi construída para análise das ferramentas em destaque ao longo desta dissertação. Quanto às funcionalidades pesquisadas, foi dado destaque aos seguintes pontos:

- Metáfora da Cidade: utilização desta metáfora na ferramenta;
- Linguagem Alvo: linguagem do código fonte do sistema, como *input* à ferramenta;
- Linguagem do Visualizador: linguagem de implementação da ferramenta;
- Análise Estática: aplicação de análise estática na ferramenta;

- Análise Dinâmica: aplicação de análise dinâmica na ferramenta;
- 3D: implementação de um ambiente de três dimensões;
- VR: uso de tecnologias de realidade virtual;
- *Plugin IDE*: possibilidade de integração com IDE.

Espaços em branco identificam pontos em que não foram encontradas informações.

Assim, as ferramentas analisadas foram: FileVis [MGAN17, PPDSF12, TC09], EvoSpaces [PPDSF12, AD07b, AD07a, LGD09], UML-City [LC07], CodeCity [Wet10, WL07, PPDSF12], ExplorViz [FKH15, FKH17, FZ17], CityVR [MGAN17] e VRCity [VNP17b].

Tabela 2.2: Análise das tecnologias de visualização de software.

			Funcionalidades							
Nome	**Autor(es)**	**Ano Última Versão Disponível**	**Metáfora da Cidade**	**Linguagem Alvo**	**Linguagem do Visualizador**	**Análise Estática**	**Análise Dinâmica**	**3D**	**VR**	***Plugin IDE***
FileVis	Peter Young Malcolm Munro	1998	Não	C	-	Sim	Não	Sim	Não	Não
EvoSpaces	Sazzadul Alam Philippe Dugerdil	2009	Sim	Java C++	Java	Sim	Sim	Sim	Não	Sim
UML-City	Christian Lange Michel Chaudron	2007	Sim	-	-	Sim	Não	Sim	Não	Não
CodeCity	Richard Wettel	2008	Sim	Meta- -linguagem	VisualWorks Smalltalk	Sim	Não	Sim	Não	Não
ExplorViz	Florian Fittkau Alexander Krause Wilhelm Hasselbring	2016	Sim	-	Java	Sim	Sim	Sim	Sim	Sim
CityVR	Leonel Merino Mohammad Ghafari Craig Anslow Oscar Nierstrasz	2017	Sim	-	C#	Sim	Não	Sim	Sim	-
VRCity	Juraj Vincur Pavol Navrat Ivan Polasek	2017	Sim	Java	-	Sim	Sim	Sim	Sim	Não

A partir da análise à tabela 2.2 é possível destacar algumas similaridades entre as ferramentas. A metáfora visual e a visualização em 3D apenas não foram aplicadas no caso da ferramenta FileVis, pela sua antiguidade. A sua análise deveu-se à importância que teve no desenvolvimento desta área. As linguagens utilizadas, tanto para o código de *input* como para a implementação da ferramenta, apresentam uma pequena inclinação para a linguagem Java, em ambos os casos. A análise estática está presente em todas as abordagens estudadas. No entanto, a funcionalidade de análise dinâmica está disponível em menos de metade das ferramentas em escrutínio, devido à sua dificuldade de implementação. Existe interesse dos investigadores em aplicar a tecnologia como *plugin* de um IDE, sendo o Eclipse o escolhido na maioria dos casos. Por fim, a aplicação da funcionalidade de realidade virtual é muito recente, estando apenas presente nos casos mais atuais.

Algumas ferramentas apresentam interações com o sistema, apesar de não fazerem nenhuma modificação. Como interações destaca-se a seleção de objetos e movimentação pelo espaço virtual.

Em suma, é possível concluir, que existe interesse por parte da comunidade científica em conseguir visualizar e analisar um sistema de software com menos esforço, recorrendo a ferramentas

que facilitem esta tarefa. Seja para compreensão ou evolução do sistema, recorrer a métodos de visualização com realidade virtual, aliados a um desenvolvimento de software *live*, revela-se uma mais valia com bastante potencial num mundo onde os sistemas são cada vez mais complexos.

Capítulo 3

Live Software Development

Este capítulo explora o problema e respetiva proposta de abordagem para esta investigação.

Inicialmente, é realizada uma contextualização à investigação em curso de *live software development* particularizando para esta dissertação, sendo a sua contribuição alvo o motor de ambiente virtual. Esta denominação foi atribuída à ferramenta desenvolvida por esta utilizar a plataforma de desenvolvimento Unity3D, conhecida por ser um motor de jogos.

A tese defendida é explorada com a apresentação da questão de investigação e as possíveis hipóteses resultantes.

Segue-se a apresentação dos requisitos funcionais e da arquitetura do motor com o auxílio de vários diagramas.

Por fim, a abordagem seguida nesta dissertação foi avaliada e implementada com o auxílio de dois casos de estudo que serviram como experiência à prática seguida.

3.1 Contextualização

Este projeto está inserido num grupo de investigação de *live software development*, na área de Engenharia de Software da Faculdade de Engenharia da Universidade do Porto, FEUP.

Neste contexto, dois projetos de dissertação foram realizados de forma paralela e complementando-se a um certo ponto: nos dados. À presente dissertação, foi realizada uma outra investigação de mestrado denominada "A Software Repository for Live Software Development"[Dom18], com o objetivo de apresentar um repositório que armazena um conjunto de dados provenientes da análise estática e dinâmica de sistemas de software em linguagem Java, através de ferramentas de *reverse engineering*. Um *plug-in* adicionado ao Eclipse fará toda a análise e envio dos dados cuja única interferência com os projetos do *workspace* é a adição de bibliotecas de AspectJ. A informação guardada no repositório é fulcral para alimentar o motor de geração do ambiente virtual, o projeto desenvolvido nesta dissertação. A Figura 3.1 esquematiza a abordagem seguida no processo de obtenção da análise estática e dinâmica do código Java, até à execução do motor que gerará o ambiente virtual.

Com efeito, ambos os projetos de dissertação reforçam a componente do *live*, na medida em que a análise gerada na execução do sistema de software deverá repercutir-se imediatamente no ambiente virtual, maximizando o *feedback* que é fornecido ao utilizador e explorando o conceito de *liveness*.

Por sua vez, o ambiente virtual apresenta-se como o responsável pela visualização do conteúdo, estático e dinâmico, e pela interação com essa informação. O utilizador é assim capaz de observar o código fonte com representações 3D e num ambiente totalmente imersivo. As representações são geradas recorrendo à metáfora visual da cidade, criando um ambiente que procura familiarizar conceitos de uma cidade através de métricas do software.

O recurso à realidade virtual para a visualização do conteúdo 3D revela-se indispensável na resposta à imersão desejada e também pela necessidade dos controladores do dispositivo de RV para permitir a referida interação. É possível a utilização de vários dispositivos, tais como os Oculus Rift[1] ou os HTC Vive[2]. Um simples monitor pode ser utilizado para uma visualização superficial do ambiente virtual.

O resultado obtido pela ferramenta será continuado em novas investigações que permitirão o maior desenvolvimento da ferramenta com adição de novas funcionalidades, mantendo o foco na compreensão de sistemas. A aplicação ao ensino em áreas como Arquitetura de Sistemas de Software ou em projetos destinados à *cloud* e ao *IoT* são fatores considerados.

3.2 Objetivos

O projeto apresentado tem como objetivo reduzir o esforço na compreensão e na manutenção de sistemas de software.

[1] Oculus Rift, `https://www.oculus.com/rift`, Último Acesso em: 08/06/2018
[2] HTC Vive, `https://www.vive.com/eu/`, Último Acesso em: 08/06/2018

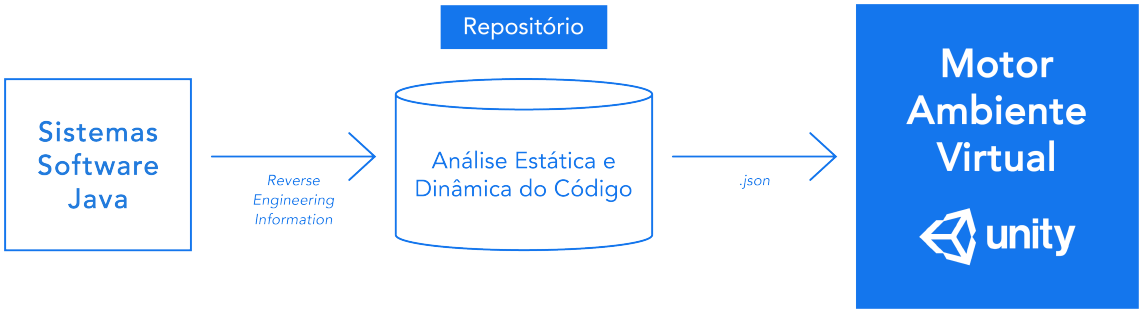


Figura 3.1: Processo utilizado para a criação de um ambiente virtual de *live software development*.

Neste sentido, esta dissertação aborda um conjunto de pontos que devem ser considerados de forma a ser possível atingir o resultado proposto.

- Aplicação da ideia do *live* que permite ao utilizador receber *feedback* instantâneo sobre as ações do sistema de software.
- Desenvolvimento de uma API de comunicação com o repositório para uma correta transferência dos dados.
- Visualização de um ambiente virtual com informações representadas de uma forma estruturada, familiar e intuitiva, recorrendo a dispositivos de realidade virtual ou a monitores convencionais.
- Interação com o ambiente virtual acedendo a toda a informação disponível e recorrendo à imersão e aos controladores da realidade virtual.
- Avaliação do ambiente virtual recorrendo a casos de estudo e a uma experiência controlada, de forma a analisar a experiência de utilização com programadores na ambientação ao protótipo e na resolução de problemas.

Estes aspetos conduzem a uma proposta de redução de esforço e têm o intuito de apresentar o caminho seguido no desenvolvimento deste projeto.

Por fim, reforçar-se que a junção do *live feedback* com a realidade virtual apresenta um papel de relevância, constituindo o foco desta dissertação.

3.3 Questão de Investigação e Hipóteses

A declaração seguinte apresenta a tese referente a esta dissertação, enunciada anteriormente no Capítulo 1.

- O uso de um ambiente virtual que permita não só visualização como também interação reduz o esforço necessário em tarefas de compreensão e de manutenção de sistemas de software.

O significado da declaração deve ser claro e homogéneo, por isso, os seus termos mais relevantes devem ser justificados.

O ambiente virtual é constituído por conteúdo 3D através de metáforas visuais e espaciais com o objetivo de criar uma maior familiarização com aquilo que é representado. A imersão deve ser completa, sem interferência de fatores exteriores.

A visualização representa o código fonte do software de forma exata e em tempo real, minimizando o atraso entre as operações do utilizador no IDE e a resposta do ambiente. A interação através de controladores de realidade virtual permite acrescentar mais informação à visualização.

Entende-se como tarefa de compreensão o ato de perceber e explorar o funcionamento, a organização ou o próprio código de um sistema de software. As tarefas de manutenção, que podem ser corretivas ou evolutivas, visando detetar problemas, erros ou descobrindo novas funcionalidades.

Por fim, o esforço traduz-se na redução do tempo e dos custos.

O público alvo remete a programadores, principalmente aqueles que não são os autores originais do sistema de software em análise.

Neste sentido, apresenta-se a questão de investigação (Q1) desenvolvida nesta dissertação.

Q1 : Será que um ambiente de desenvolvimento que inclua funcionalidades de *liveness* e realidade virtual, ajuda na compreensão e na manutenção de sistemas de software?

A Tabela 3.1 identifica as hipóteses subjacentes à questão de investigação apresentando uma opção nula, onde o impacto do ambiente virtual não acrescenta qualquer vantagem, e uma opção alternativa, com resultados positivos.

Tabela 3.1: Hipóteses nula e alternativa.

Hipótese Nula	Hipótese Alternativa
$H0$: A ferramenta não tem impacto positivo na redução do esforço na compreensão e manutenção de um sistema de software.	$H1$: A ferramenta tem impacto positivo na redução do esforço na compreensão e manutenção de um sistema de software.

Em suma, é apresentada apenas uma questão de investigação que engloba o objetivo fulcral da investigação, com duas hipóteses respetivas.

3.4 Requisitos

O ambiente virtual possui apenas um ator, o programador.

A Figura 3.2 apresenta o diagrama de casos de uso para o programador, exibindo as interações possíveis do ator com o ambiente, e cujos casos são especificados de seguida pelas unidades funcionais que representam.

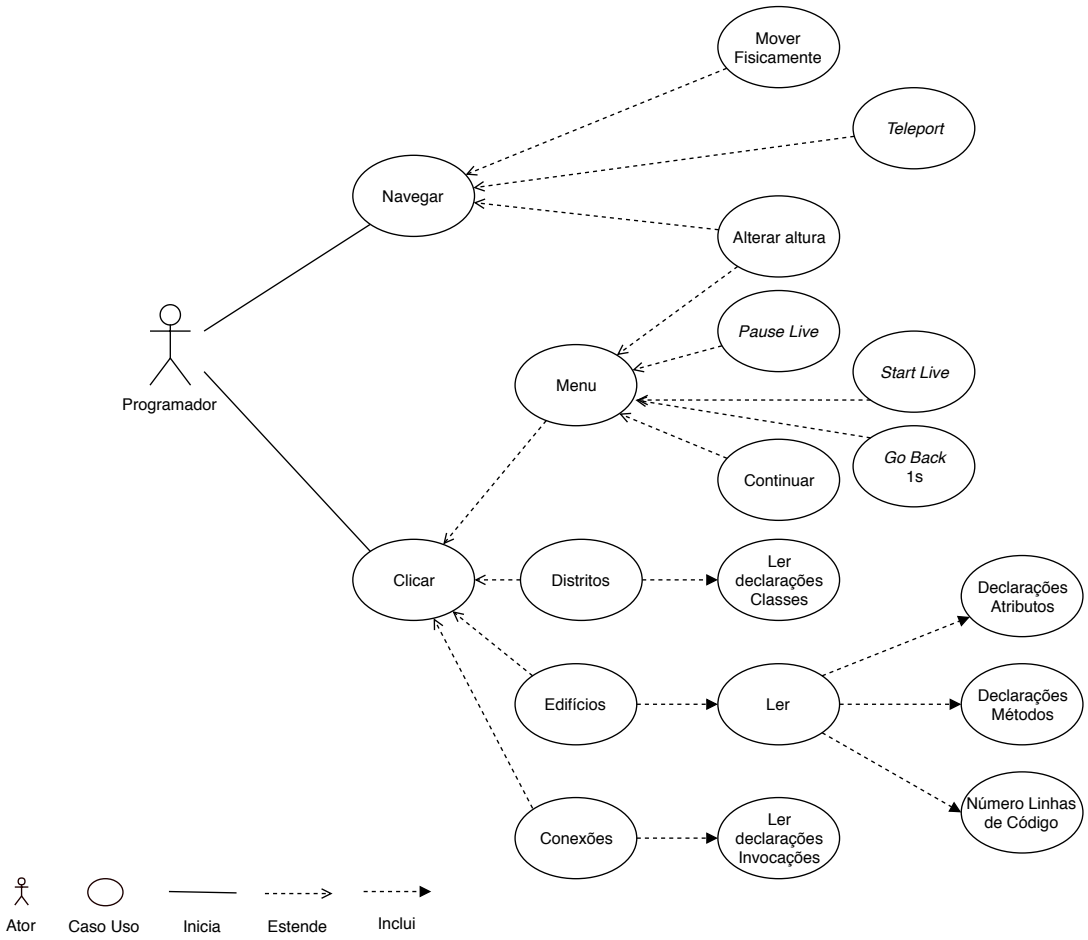


Figura 3.2: Diagrama de casos de uso.

1. **Navegar**. O programador deverá ser capaz de percorrer todo o espaço do ambiente virtual.
   - 1.1. **Mover Fisicamente**. O programador deverá ser capaz de se mover no mundo real e ver repercutida a ação no mundo virtual.
   - 1.2. ***Teleport***. O programador deverá ser capaz de se mover de um lugar de destino diferente do de origem no ambiente virtual apenas com um clique.
   - 1.3. **Alterar Altura**. O programador deverá ser capaz de alterar a coordenada *y* da sua posição, correspondendo à altura, com um botão, no ambiente virtual.
2. **Clicar**. O programador deverá ser capaz de utilizar os comandos disponíveis nos controladores de RV.
   - 2.1. **Menu**. O programador deverá ser capaz de interagir com um conjunto de botões no ambiente virtual.
     - 2.1.1. ***Go Back* 1s**. O programador deverá ser capaz de voltar atrás do tempo 1 segundo, obrigando o ambiente virtual a repetir tudo o que aconteceu nesse período de tempo.

2.1.2. ***Pause Live***. O programador deverá ser capaz de pausar a visualização dos eventos recebidos no ambiente virtual.

2.1.3. ***Start Live***. O programador deverá ser capaz de iniciar a visualização dos eventos recebidos no ambiente virtual.

2.1.4. **Continuar**. O programador deverá ser capaz de continuar a visualização dos eventos recebidos no ambiente virtual desde o momento que pausou.

2.2. **Distritos**. O programador deverá ser capaz de ler as declarações de classes de um pacote se clicar no distrito que o representa.

2.3. **Edifícios**. O programador deverá ser capaz de ler as declarações de atributos e de métodos e ainda o número de linhas de código de uma classe se clicar no edifício que a representa.

2.4. **Conexões**. O programador deverá ser capaz de ler invocações de métodos se clicar nos arcos que as representa.

Assim, o programador tem à sua disposição um conjunto de ferramentas para navegar no ambiente e explorá-lo o quanto possível recorrendo aos comandos disponíveis.

3.5 Arquitetura

A arquitetura do software desenvolvido revela-se imprescindível na apresentação da estrutura do sistema implementado.

A Figura 3.3 exibe, em alto nível, a arquitetura do ambiente virtual desenvolvido com um diagrama de arquitetura. O *MainController* representa o motor de geração do ambiente virtual e os pacotes *Components*, *Data* e *UI* os principais e mais influentes módulos à estrutura e implementação. Explorando o processo, após ser iniciada a ferramenta, a classe *Importer* acede ao repositório e tenta obter a análise estática do projeto e, em seguida, abre um meio de comunicação para a receção da análise dinâmica. Este mecanismo de comunicação é explicado no ponto seguinte. Consecutivamente, o motor faz a extração dos dados recebidos que foram colocados em cache, numa estrutura definida pelo motor. Toda a informação de entrada é transformada em formato JSON. Caso esta informação exista, é transformada e carregada para as componentes respetivas, gerando os objetos edifícios, distritos e conexões. Por fim, a informação das componentes é instanciada criando o ambiente virtual acessível ao utilizador em adição aos mecanismos que permitam a sua interação, ou seja, habilitando os menus e a navegação. A qualquer momento o motor pode ter necessidade de atualizar algum componente se for notificado que novos dados foram gerados no repositório e importados para a ferramenta.

Comunicação com Servidor

O motor de geração do ambiente virtual é alimentado pelos dados disponibilizados no repositório alojado num servidor. Como tal, a comunicação com esse servidor revela-se de extrema importância para a execução do ambiente.

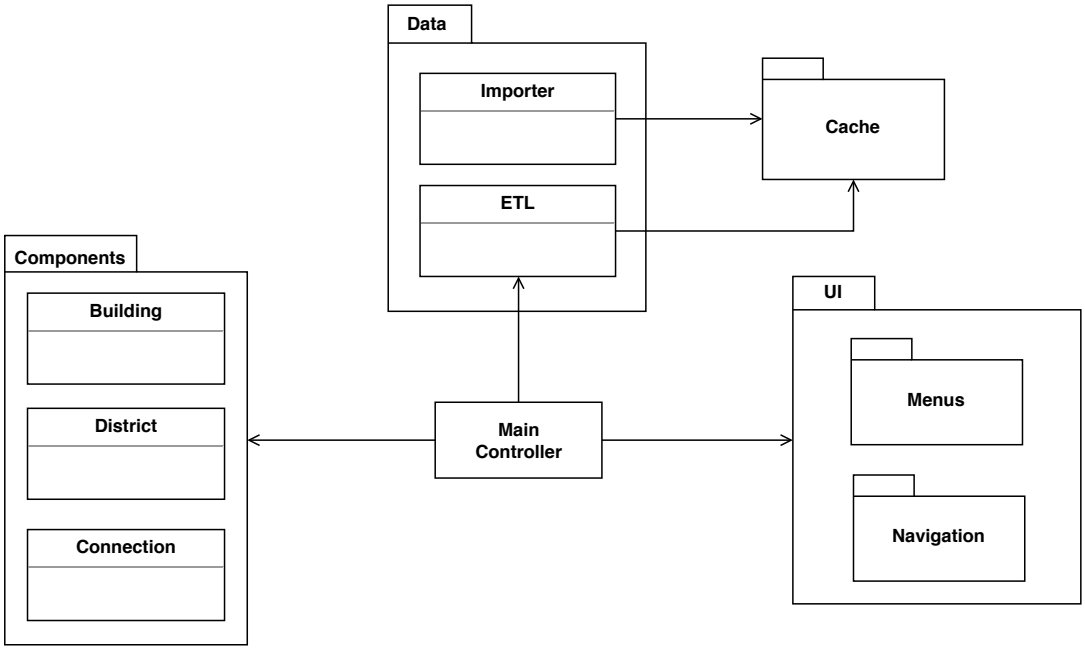


Figura 3.3: Diagrama de arquitetura do ambiente virtual.

Esta comunicação é realizada de duas formas distintas, separando a análise estática da dinâmica. O diagrama de colaboração da Figura 3.4 exemplifica o processo comunicativo com o servidor através de um exemplo, separando as duas análises realizadas pelo motor.

Análise Estática

Concretamente, a ferramenta ao ser iniciada deverá importar todos os dados necessários do repositório para a instanciação do ambiente estático através de um pedido HTTP ao servidor. Este pedido é endereçado a um URL onde estão localizados todos os dados extraídos estaticamente do sistema de software Java. Se o pedido tiver sucesso, retorna um *array* de *bytes* com a informação que é processada para um formato JSON, criando uma estrutura que ficará guardada em cache para ser usada pelo motor. Este processo representa o mecanismo de obtenção da análise estática e o Código 3.1 apresenta um exemplo da formatação JSON para a estruturação da informação recebida.

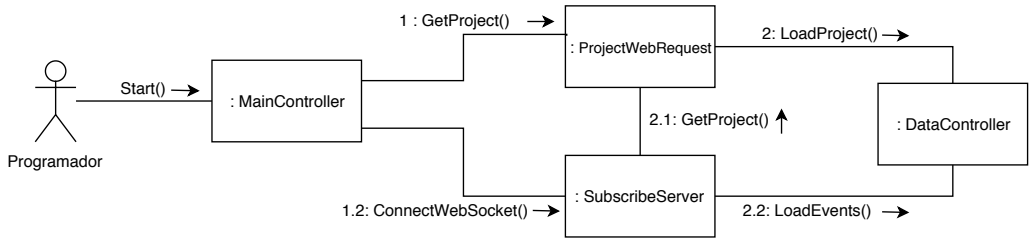


Figura 3.4: Diagrama de colaboração com exemplo de comunicação com servidor.

```
{
   "allProjectData":[
      {
         "num_packages": 1,
         "project_name": "Maze",
         "packages":[
           {
              "package_name":"maze.cli",
              "class_count":1,
              "i_classes":[
                 {
                    "class_name":"Interface",
                    "class_hash":"1f1cc557b0ed181f8fds84tyt0bc74ceed73e7
                       bbf4278cebf4ba0e43d0f3ff465fda9c",
                    "lines_of_code":99,
                    "i_methods":[
                       {
                          "argument_count":1,
                          "method_name":"main",
                          "method_count":1,
                          "arguments":[
                             {
                                "argumentName":"args",
                                "attribute_type":"String[]"
                             }
                          ],
                          "returnattribute_type":"void"
                       }
                    ],
                    "attribute_count":1,
                    "class_attributes":[
                       {
                          "attribute_name":"Move",
                          "attribute_type":"int"
                       }
                    ]
                 }
              ]
           }
         ]
      }
   ]
}
```

Código 3.1: Formato JSON para informação estática

Análise Dinâmica

Após a ferramenta obter com sucesso os dados da análise estática do repositório, esta subscreve o servidor criando uma linha de comunicação através de *web sockets*. A ferramenta fica assim a aguardar qualquer tipo de mensagem sobre atualizações que possam acontecer ao sistema de software em análise, posteriores à sua inicialização.

As mensagens podem assumir um de dois tipos: notificações de alterações na estrutura ou invocações realizadas.

Uma notificação avisa o motor que existiu alguma alteração na estrutura estática do sistema e é necessário repetir o pedido HTTP ao servidor, como referido anteriormente. O Código 3.2 representa um exemplo da estrutura recebida em JSON, alertando de operações que ocorreram e que modificaram a estrutura construída no momento da inicialização.

Por outro lado, caso seja enviado um evento representativo de uma invocação ocorrida, essa informação é processada para um formato JSON, criando uma estrutura que ficará guardada em cache para ser usada pelo motor. Um processo semelhante ao já apresentado. O Código 3.3 apresenta a formatação JSON para a estruturação da informação recebida representando as invocações

```
{
   "fetch_structure":"Interface",
   "operation":"create",
   "package_id":"integer",
   "project_id":"integer",
   "class_id":"integer"
}
```

Código 3.2: Formato JSON para notificações

```
{
   "id":1,
   "origin_hash":"1f1cc557b0ed181f8fds84tyt0bc74ceed73e7bbf4278cebf4ba0e43
      d0f3ff465fda9c",
   "destination_hash":"124074653a6adc65e7f3207d12455713daa855b3eb2c99e16d8
      5f8fdab3c670f",
   "timestamp":"2018-05-24T13:45:33.236Z",
   "signature":"boolean maze.cli.Interface.Move()",
   "project_name":"junit",
   "event_arguments":[]
}
```

Código 3.3: Formato JSON para informação dinâmica

A necessidade de utilização de uma cache reflete-se no mecanismo de serialização da plataforma de desenvolvimento do motor utilizada, aumentando a eficácia na interação com serviços *web*.

A estrutura com formato JSON apresentada nos 3 casos reflete uma API previamente estabelecida entre o motor e o repositório, de forma a gerir a transferência da informação.

Neste sentido, a premissa de forte interesse a ambas as investigações supramencionadas, o *live*, deve ser favorecida por uma rápida e eficaz comunicação entre o motor e o repositório.

3.6 Casos de Estudo

Com o objetivo de avaliar a proposta implementada e testar a aplicação do ambiente virtual em casos reais foram selecionados dois casos de estudo, de dimensões diferentes, ambos desenvolvidos sobre a linguagem Java.

3.6.1 Maze

O Maze é um projeto desenvolvido anualmente na unidade curricular de Laboratório de Programação Orientada a Objetos do Mestrado Integrado em Engenharia Informática e Computação da FEUP.

O sistema Maze tem a duração de meio semestre para a sua implementação, normalmente, em grupos de 2 elementos. O objetivo é construir um jogo em formato de labirinto em que o jogador tem que encontrar armas que lhe permitam vencer um ou vários dragões que também estão no labirinto.

A versão usada como caso de estudo foi desenvolvida em 2014, ano em que os investigadores deste projeto concluíram a unidade curricular. Ao longo dos anos o enunciado do projeto sofreu algumas alterações, mas é espectável que os projetos recentes também possam ser usados porque respeitam o único requisito: implementação em linguagem Java.

Este projeto foi escolhido para caso de estudo devido à sua dimensão. Sendo apenas um projeto de meio semestre, o seu tamanho é reduzido, facilitando testes mais pormenorizados e úteis no início da implementação do ambiente virtual.

A Figura 3.5 retrata o sistema Maze de uma posição superior, permitindo uma visualização total de todos os seus constituintes principais, os pacotes e classes, ou seja, os distritos e os edifícios, respetivamente. Este mapa da cidade reflete a sua postura permitindo uma clara perceção da distribuição do espaço.

A Figura 3.6 permite uma visualização das alturas dos edifícios, conseguindo percecionar o número de métodos de cada classe.

Este caso de estudo foi ainda selecionado para ser utilizado numa experiência controlada, como apresentado no Capítulo 5.

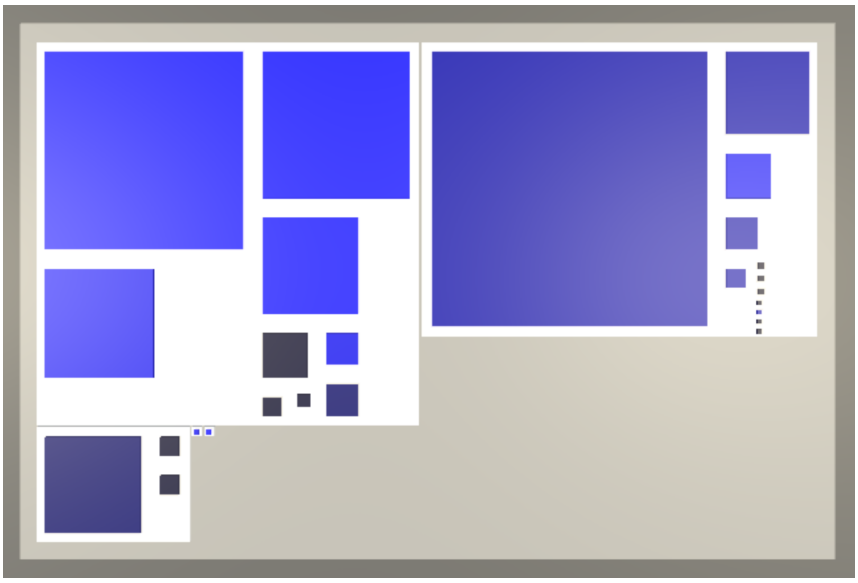


Figura 3.5: Vista superior do Maze com separação de pacotes ao mesmo nível.

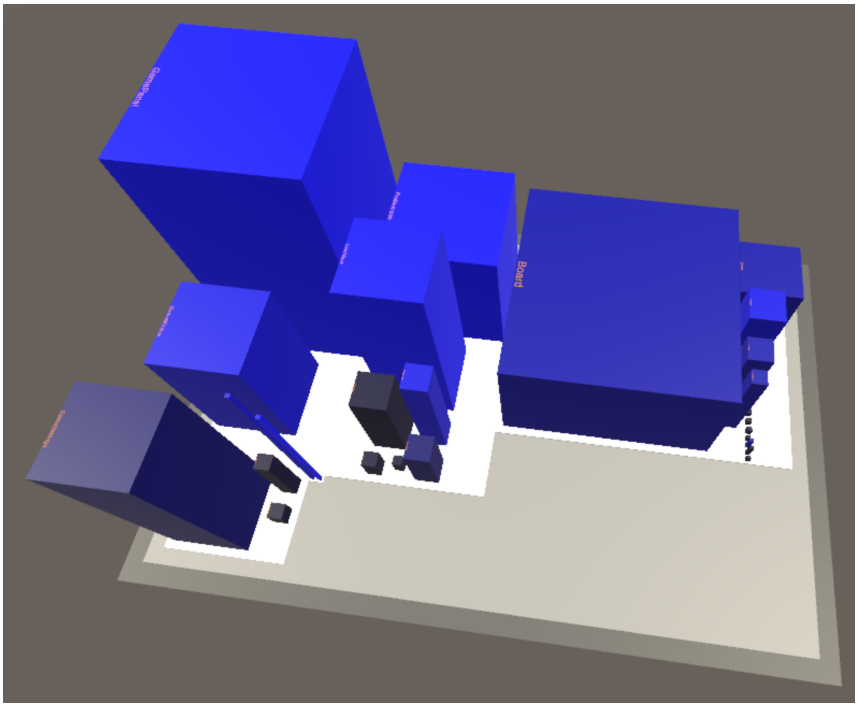


Figura 3.6: Vista diagonal superior do Maze.

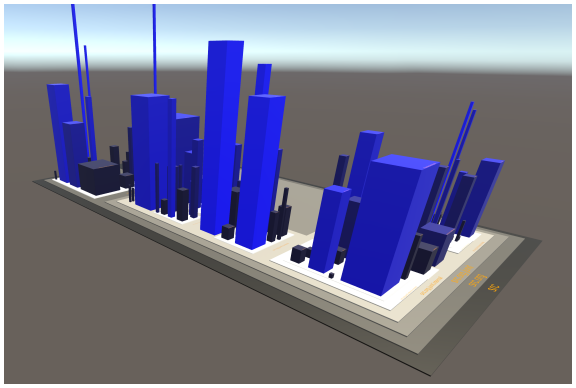
3.6.2 jUnit

jUnit é uma *framework open-source* em Java diversas vezes utilizada na literatura como caso de estudo [Agu03, VNP17b].

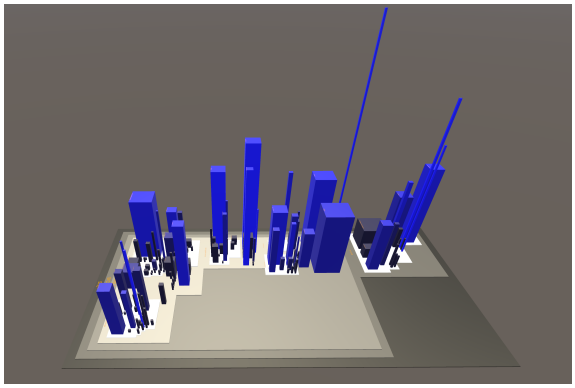
A sua aplicação a esta dissertação adequa-se pela necessidade de provar a capacidade do ambiente virtual para lidar com um sistema de média dimensão e complexidade, como é o caso desta *framework*. Como tal, revela-se como o principal meio para validar o trabalho desenvolvido e daí assumir a sua escalabilidade. Comparativamente com o Maze, o jUnit assume uma dimensão e uma complexidade consideravelmente superiores.

O sistema utilizado foi o jUnit 4 e cujo código fonte foi exportado do GitHub[3]. Os pacotes de teste do jUnit 4 não foram incluídos.

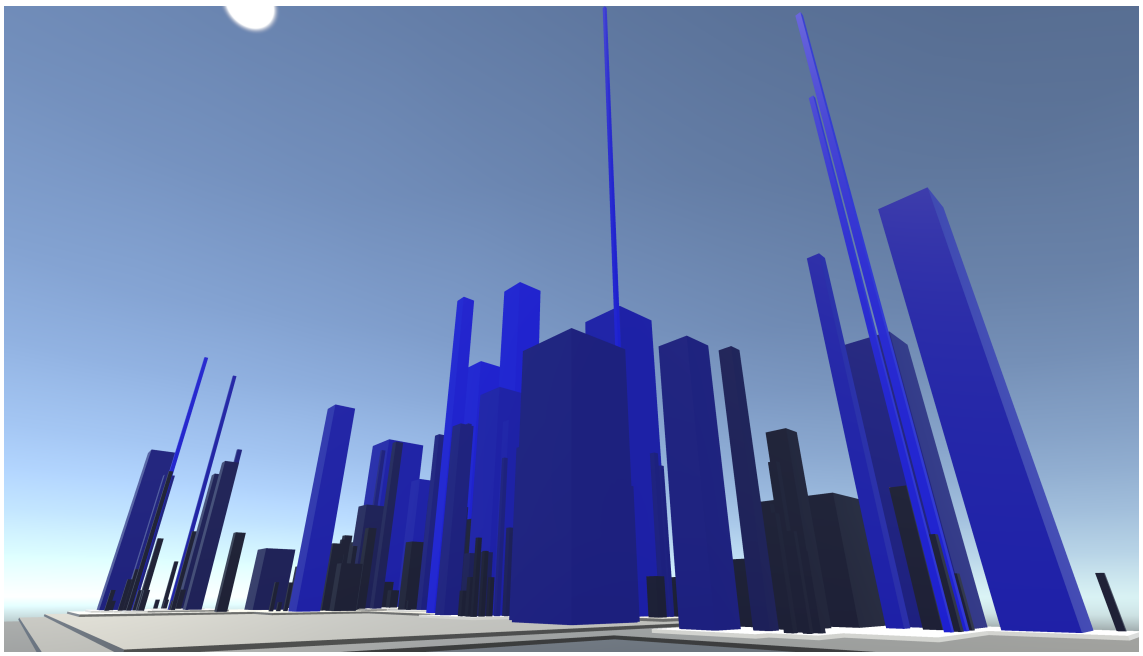
A Figura 3.7 retrata a visualização do sistema em estudo em várias perspetivas e pontos de referência.



(a) Vista diagonal do jUnit.



(b) Vista lateral do jUnit.



(c) Vista no solo do jUnit.

Figura 3.7: Perspetivas da visualização do caso de estudo jUnit.

[3]jUnit4, https://github.com/junit-team/junit4, Último Acesso em: 10/06/2018

3.7 Sumário

Este capítulo contextualizou o projeto em desenvolvimento pelo grupo de *live software development* realçando a origem dos dados que alimentam o motor do ambiente virtual e como foi projetada essa comunicação.

Foi identificado com mais pormenor o problema que se pretende resolver, a questão de investigação e as duas possíveis hipóteses. A arquitetura surge neste seguimento de forma a responder ao problema.

O uso dos dois casos de estudo definidos nos pontos anteriores pretendeu, por um lado, facilitar uma avaliação rápida no desenvolvimento de funcionalidade, no caso do Maze, e, por outro lado, verificar a capacidade do ambiente virtual reconhecer os dados de um sistema de maior dimensão e visualizá-los de forma correta, no caso do jUnit. Em ambos os casos, o ambiente virtual deve responder à questão de investigação.

O capítulo seguinte descreve a implementação realizada para o desenvolvimento do motor de realidade virtual.

Capítulo 4

Ambiente Virtual

O projeto desenvolvido no decurso desta dissertação concentra-se no ambiente virtual que é gerado e mantido em atualização por um motor que utiliza dados provenientes de um repositório alocado num servidor.

Este capítulo explica as estratégias de implementação dos principais componentes do projeto, desde as tecnologias, até aos algoritmos relevantes e a interface de utilizador.

A abordagem seguida na implementação da visualização e na implementação da interação, com um sistema de software, focou a capacidade da metáfora visual responder ao primeiro aspeto e da realidade virtual resolver o segundo.

A metáfora da cidade, já explorada na literatura, foi adaptada a esta dissertação de forma a transmitir ao utilizador aspetos que se revelam importantes na resposta ao problema de investigação. Nesse sentido, os utilizadores conseguem observar e interagir com maior detalhe num ambiente de realidade virtual utilizando os Oculus Rift, dispositivo usado para responder à necessidade de RV nesta dissertação.

Por fim, o *liveness* surge como o meio para aplicar o imediatismo no *feedback* ao utilizador.

4.1 Tecnologias

O motor de ambiente virtual foi desenvolvido recorrendo ao motor de jogos Unity3D, na sua versão gratuita. O Unity3D é um motor de jogo usado para desenvolver ambientes 2D e 3D, com funcionalidades de *drag-and-drop* e com uma vasta API disponível. O Unity assegura compatibilidade com vários dispositivos e plataformas.

A linguagem de programação utilizada foi C# por permitir o *scripting* da API e por ser a mais utilizada dentro da comunidade Unity. A Figura 4.1 exibe uma experiência de utilização do Unity3D.

É recomendado que o ambiente virtual seja utilizado em computadores que tenham a capacidade de executar plataformas de realidade virtual, para uma melhor experiência de utilização. É ainda aconselhado que a ligação à rede *internet* seja forte, de forma a não criar atrasos na comunicação com o servidor.

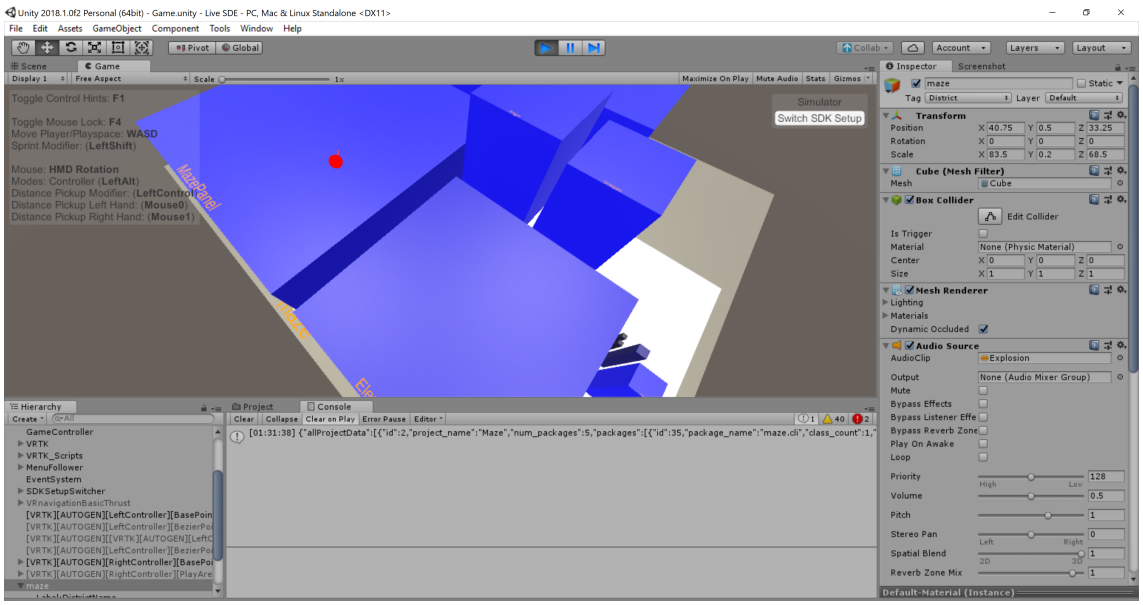


Figura 4.1: Exemplo de utilização do motor de jogos Unity3D na implementação do ambiente virtual.

4.2 Comunicação com Servidor

A obtenção de dados é um dos principais pontos da execução do motor de jogo. Como explicado na arquitetura desta comunicação, Secção 3.5, são realizados pedidos HTTP e é aberto um canal de comunicação através de *web sockets*.

Os pedidos HTTP são executados recorrendo ao módulo UnityWebRequest que é usado para comunicar com servidores *web*. Pedidos POST ao servidor permitem a obtenção de toda informação da análise estática ao projeto.

Por outro lado, o motor do ambiente virtual deve subscrever um canal de comunicação em Web Sockets e para tal foi utilizado o pacote WebSocketSharp[1], que consiste numa implementação cliente/servidor preparada para integração com o Unity. Esta implementação permite deixar o motor numa continua ação de escuta ao servidor, preparado para agir assim que receber algum tipo de evento, minimizando atrasos.

4.3 Visualização da Informação

A visualização da informação é o ponto chave que levará à compreensão do sistema. Para isso são utilizadas metáforas visuais e espaciais, que familiarizam o espaço gerado com conceitos e com uma organização que facilitam a compreensão exigida ao utilizador nas suas tarefas.

4.3.1 Metáfora da Cidade

A metáfora da cidade foi aplicada a este projeto pela forte presença em investigações relacionadas com visualização do software, como discutido na Secção 2.4.4, e pela sua capacidade de criar um ambiente familiar ao introduzir conceitos de uma cidade numa representação de um sistema de software.

O mapeamento realizado entre o que é representado e o que é visualizado é expresso na Tabela 4.1, onde é feito uso do mecanismo de pacotes, fornecido pelas linguagens orientadas a objetos, para uma melhor organização da cidade.

Neste sentido, um projeto é representado por uma cidade, os pacotes são representados por distritos contidos na cidade, as classes são edifícios que estão inseridos em algum distrito e, por fim, as invocações de métodos são conexões, em formato de arco ou linha vertical.

Tabela 4.1: Relação entre representação e visualização do sistema de software.

Representação	**Visualização**
Projeto	Cidade
Pacote	Distrito
Classe	Edifício
Invocação	Arco

[1] WebSocketSharp, https://github.com/sta/websocket-sharp, Último Acesso em: 15/06/2018

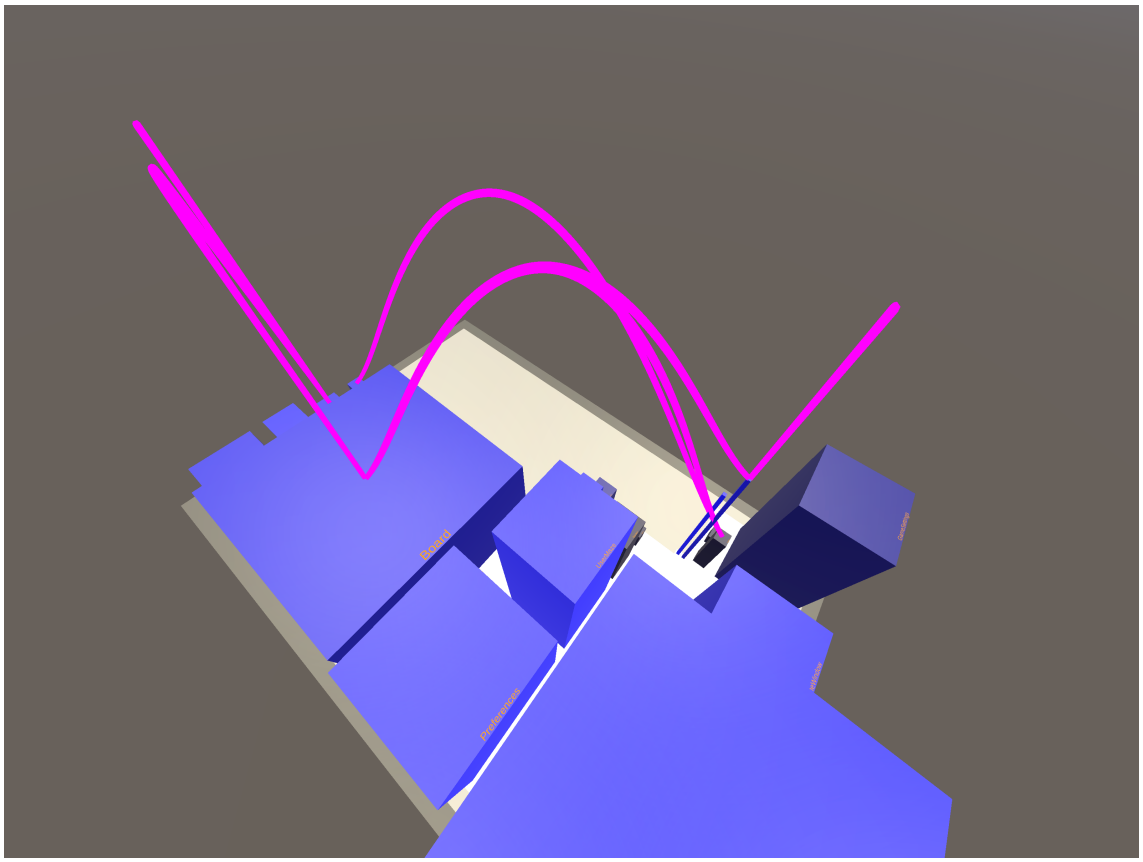


Figura 4.2: Fotografia tirada ao ambiente virtual num momento de várias invocações a acontecerem.

A Figura 4.2 explicita a identificação dos pacotes e das classes, ou seja, dos distritos e dos edifícios, respetivamente, num sistema exemplo. Os arcos surgem como uma extensão dinâmica da visualização até agora referida. Representam invocações entre classes e são o mais forte elemento visual que corrobora a ideia do *live*.

A abordagem seguida pela metáfora visual assemelha-se à usada na literatura.

A Figura 4.3 expõe as métricas usadas para representar um distrito que é instanciado como um plano. Importa definir a sua área e a sua cor que terá tonalidades de cinza. Mais escuro será quanto maior for o seu nível na cidade, ou seja, quantos mais distritos tiver dentro.

A Figura 4.4 aborda a representação dos edifícios. Neste caso, a altura já é um fator a considerar e é definido pelo número de métodos da classe representada. A área da base é igual ao quadrado do número de atributos e a cor, em tons de azul, está relacionada com o número de linhas de código da classe. Edifícios com uma tonalidade mais brilhante representam classes com poucas linhas de código, enquanto que uma tonalidade mais escura representa um valor superior de linhas de código.

As cores para os dois casos anteriores assumem o formato RGB e são definidas com valores previamente estabelecidos. A Tabela 4.2 faz a correspondência entre o rácio do nível do pacote

e a sua tonalidade de cinzento. Este rácio é obtido pela Equação 4.1, utilizando o nível total de pacotes do sistema e o nível do pacote a colorir. A Tabela 4.3 faz a correspondência entre o rácio de linhas de código da classe e a sua tonalidade de azul. Este rácio é obtido pela Equação 4.2, utilizando o número de linhas de código da classe com o maior número deste atributo e o número de linhas da classe a colorir. A divisão de linhas sugerida vai de encontro aos casos de estudo utilizados, criando o efeito desejado.

Por fim, a Figura 4.5 representa as métricas relacionadas com as conexões. Uma conexão pode assumir um formato de linha vertical se uma classe invocar um método da mesma, pois o ponto de origem e destino é o mesmo. Nos outros casos, assume o formato de um arco conectando a classe de origem e a de destino. A altura é variável e está relacionada com as alturas assumidas pelos edifícios adjacentes e a cor está definida como um rosa brilhante, para uma melhor perceção da mesma em cena.

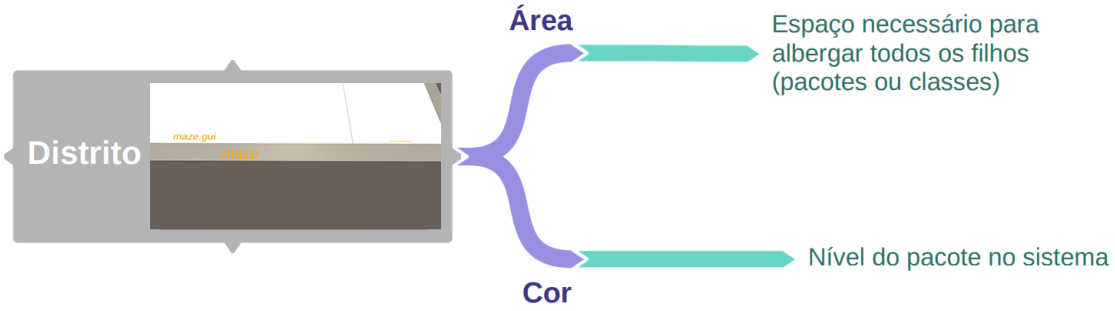


Figura 4.3: *Mindmap* das métricas usadas na visualização dos pacotes.

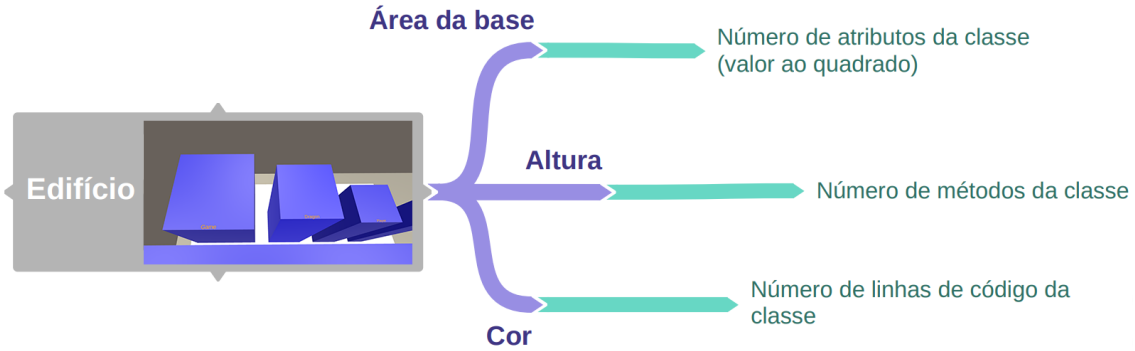


Figura 4.4: *Mindmap* das métricas usadas na visualização das classes.

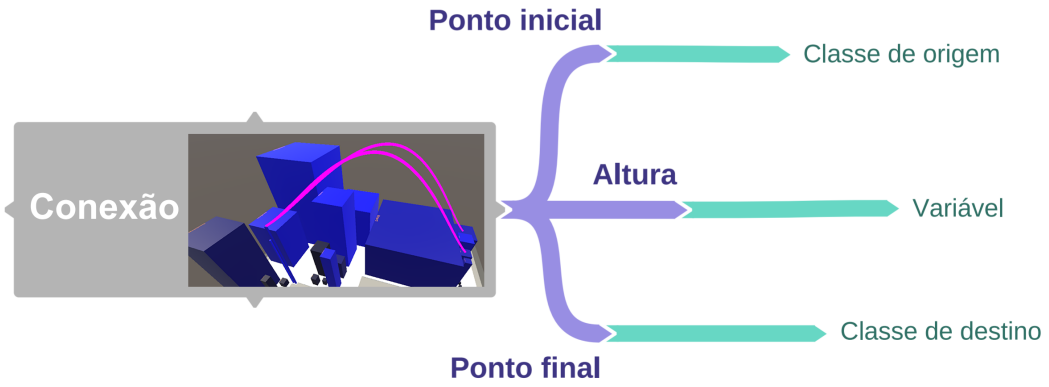


Figura 4.5: *Mindmap* das métricas usadas na visualização de invocações.

Tabela 4.2: Cor RGB atribuída ao distrito de acordo com o rácio correspondente ao seu nível.

Rácio Distritos	**Cor RGB**
<0.05	(255, 255, 255)
<0.1	(204, 204, 204)
<0.2	(166, 166, 166)
<0.3	(140, 140, 140)
<0.4	(128, 128, 128)
<0.5	(115, 115, 115)
<0.6	(89, 89, 89)
<0.7	(64, 64, 64)
<0.8	(38, 38, 38)
>= 0.9	(0, 0, 0)

Tabela 4.3: Cor RGB atribuída ao edifício de acordo com o rácio correspondente ao seu nível.

Rácio Edifícios	**Cor RGB**
<0.1	(0, 0, 255)
<0.25	(0, 0, 204)
<0.5	(0, 0, 153)
<0.75	(0, 0, 102)
>= 0.75	(0, 0, 51)

$$RácioDistritos = \frac{NívelTotaldePacotes - NíveldoPacote}{NívelTotaldePacotes} \tag{4.1}$$

$$RácioEdifícios = \frac{NúmeroMaxdeLinhas - NúmerodeLinhas}{NúmeroMaxdeLinhas} \tag{4.2}$$

Assim, a metáfora visual é visível na Figura 4.6, num exemplo que representa todos os aspetos que a constituem.

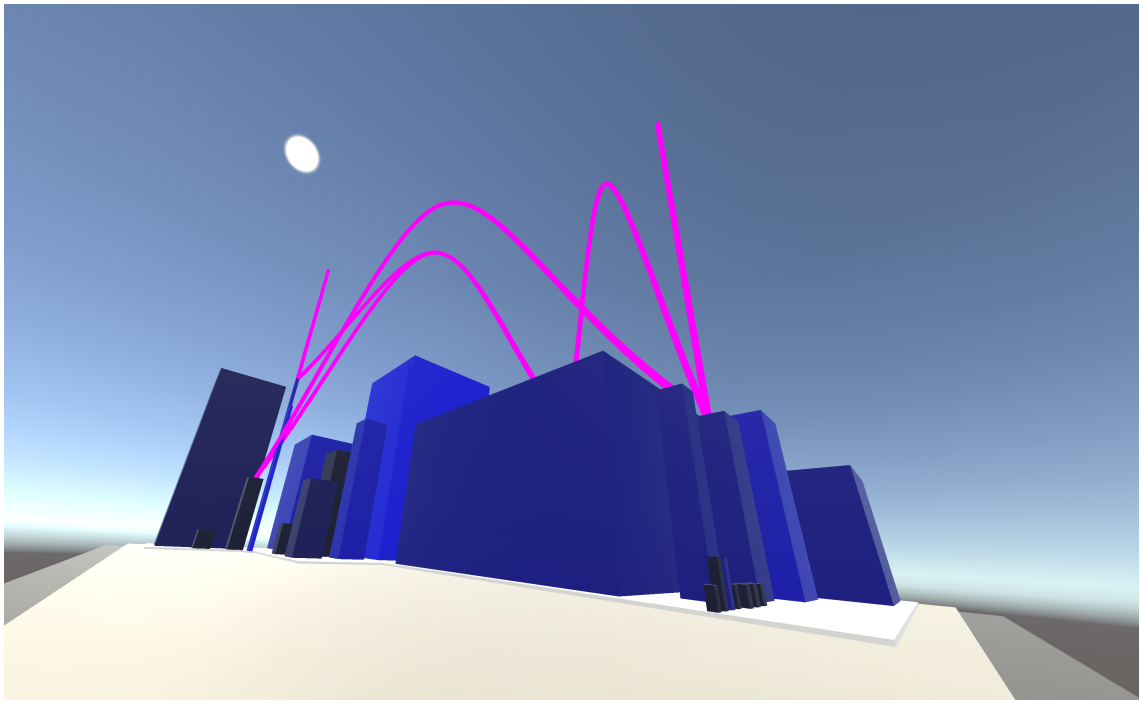


Figura 4.6: Exemplo de uma cidade com todas as representações visíveis.

4.3.2 Algoritmo de Alocação de Objetos

A metáfora visual, analisada no capítulo anterior, permitiu esclarecer como é realizada a representação de pacotes, classes e invocações. No entanto, nada foi dito quanto à alocação dos mesmos.

A técnica usada para a alocação de objetos, sejam distritos ou edifícios, para posterior instanciação, é considerada um dos fatores mais importantes pois auxiliarão na compreensão do software.

A alocação dos pacotes e respetivos edifícios seguem critérios previamente estabelecidos, de forma a manter o conteúdo espacial organizado e agrupado. O sistema de software assumirá uma relação hierárquica de pacotes e as suas classes certamente terão dimensões diferentes. Assim sendo, a sua representação virtual assumirá, na mesma forma, dimensões diferentes. O algoritmo de alocação deve respeitar os seguintes pontos:

- **Minimização do espaço necessário.** Os distritos devem agrupar os edifícios ou outros distritos, mantendo a relação hierárquica herdada do sistema. O espaço entre objetos deve ser mínimo, criando apenas um intervalo uniforme entre os edifícios de forma a permitir ao utilizador distanciar as componentes e até poder mover-se entre eles no ambiente virtual.

- **Manutenção de um espaço retangular.** Forçar a que a cidade assuma uma postura de distritos retangulares permitirá melhorar a perceção do espaço ocupado por esses e obrigar a manter uma área total reduzida.

- **Instanciação de elementos pela sua dimensão.** A alocação é realizada começando pelo objeto com maior área da base e instanciando-o no ponto mais próximo do centro do mundo. Com isto será possível aumentar o número de espaços ou ruas entre os edifícios do distrito à medida que vão sendo alocados.

Inicialmente, na implementação do algoritmo de alocação foi levantada a possibilidade de utilização de Tree-maps. No entanto, a técnica *rectangle packing layout*, proposta por Wettel *et al.* [Wet10], foi considerada mais vantajosa porque divide o espaço existente em blocos retangulares tendo em conta as proporções dos edifícios. A técnica de Tree-maps obriga a saber *a priori* quanto espaço será necessário para colocar um objeto.

Assim, tomando por base o algoritmo de Wettel *et al.* [Wet10], foi desenhada uma adaptação do mesmo. O Algoritmo 1 é chamado a alocar os edifícios de cada distrito. Por sua vez, um algoritmo semelhante será aplicado para a alocação de distritos que se encontram dentro de outros. São ainda considerados distritos que possuem edifícios e distritos filhos.

Como a alocação é realizada da folha para a raiz, serão instanciados os edifícios e apenas depois os distritos. Todos os objetos são instanciados com o centro no ponto de coordenadas (0,0,0) e posteriormente movidos para a sua respetiva posição. Com isto, é construída uma estrutura em árvore, após a instanciação, de forma a que a raiz seja responsável por todos os componentes da cidade, facilitando alterações geométricas, por exemplo.

Analisando o Algoritmo 1, o ciclo principal irá percorrer a lista de edifícios do distrito e procurar a melhor célula para o alocar. Na primeira iteração, a única célula disponível terá dimensão suficiente para albergar todos os edifícios da lista. Essa célula será removida da lista de células e novas células serão adicionadas à lista caso exista espaço restante. Os rácios são recalculados com o aumento do espaço ocupado pelo edifício alocado.

A função que procura a melhor célula é de relativa importância porque tomará a decisão de selecionar a posição do edifício. O Algoritmo 2 executa essa pesquisa e devolve a melhor célula. Para tal é considerado os rácios calculados no algoritmo anterior, bem como se a alocação do novo edifício expande os limites até então, ou seja, a área total ocupada pela lista de edifícios do pacote.

A redução do espaço ocupado foi considerado um fator fundamental, mas não o único a merecer cuidado. Sendo o objetivo construir uma visualização e uma interação mais favorável para o utilizador, o algoritmo possibilita ainda a seleção de células que permitam formar, no total do distrito, um quadrado, correspondendo, a mesma largura pelo mesmo comprimento. Ou seja, não existe qualquer obrigação que leve a instanciação a ocorrer o mais próxima possível de todos os objetos já criados, impedindo assim a criação de áreas de espaços vazios, sem edifícios.

Algoritmo 1 Adaptação da alocação *rectangle packing* para os edifícios.

Require: buildings are sorted by size descending

1: $freeCell \leftarrow$ cell at index (0,0) with sizes (X, Z) $\sum_i buildings[i].size$
2: $freeCells \leftarrow freeCell$
3: $ratioX, ratioZ \leftarrow 0$
4: **for** *building* in *buildings* **do**
5: **if** $freeCells$ not empty **then**
6: $bestCell \leftarrow$ FINDBESTCELL(freeCells, ratioX, ratioZ)
7: remove *bestCell* from $freeCells$
8: $ratioX \leftarrow$ MAX(ratioX, bestCell.X + building.SizeX)
9: $ratioZ \leftarrow$ MAX(ratioZ, bestCell.Z + building.SizeZ)
10: **if** building.SizeX < bestCell.SizeX **then**
11: $freeCell1 \leftarrow$ horizontal cut
12: add $freeCell1$ to $freeCells$
13: **end if**
14: **if** building.SizeZ < bestCell.SizeZ **then**
15: $freeCell2 \leftarrow$ vertical cut
16: add $freeCell2$ to $freeCells$
17: **end if**
18: **end if**
19: **end for**

Algoritmo 2 Função que procura a melhor célula para alocar um determinado edifício.

1: **function** FINDBESTCELL($freeCells$, $ratioX$, $ratioZ$) ▷ (ratioX, ratioZ) are the boundaries of the occupied area
2: $bestCell \leftarrow$ null
3: $bestRatio \leftarrow 0$
4: **for** *cell* in $freeCells$ **do**
5: **if** ($this.building.sizeX$, $this.building.sizeZ$) fits in *cell* **then**
6: $ratio \leftarrow$ total ratio if *cell* is used to place the element
7: **if** ($this.building.sizeX$, $this.building.sizeZ$) is boundary expander **then**
8: **if** $ratioX*ratioZ >= Max(x, ratioX) * Max(z, ratioZ)$ **then**
9: $bestCell \leftarrow cell$
10: **end if**
11: **else if** $|1 - ratio| < |1 - bestRatio|$ **then**
12: $bestCell \leftarrow cell$
13: $bestRatio \leftarrow best$
14: **end if**
15: **end if**
16: **end for**
17: **return** *bestCell*
18: **end function**

De forma a explorar este processo, foi gerado um exemplo simples que exemplifica a alocação de um conjunto de edifícios num pacote, Figura 4.7.

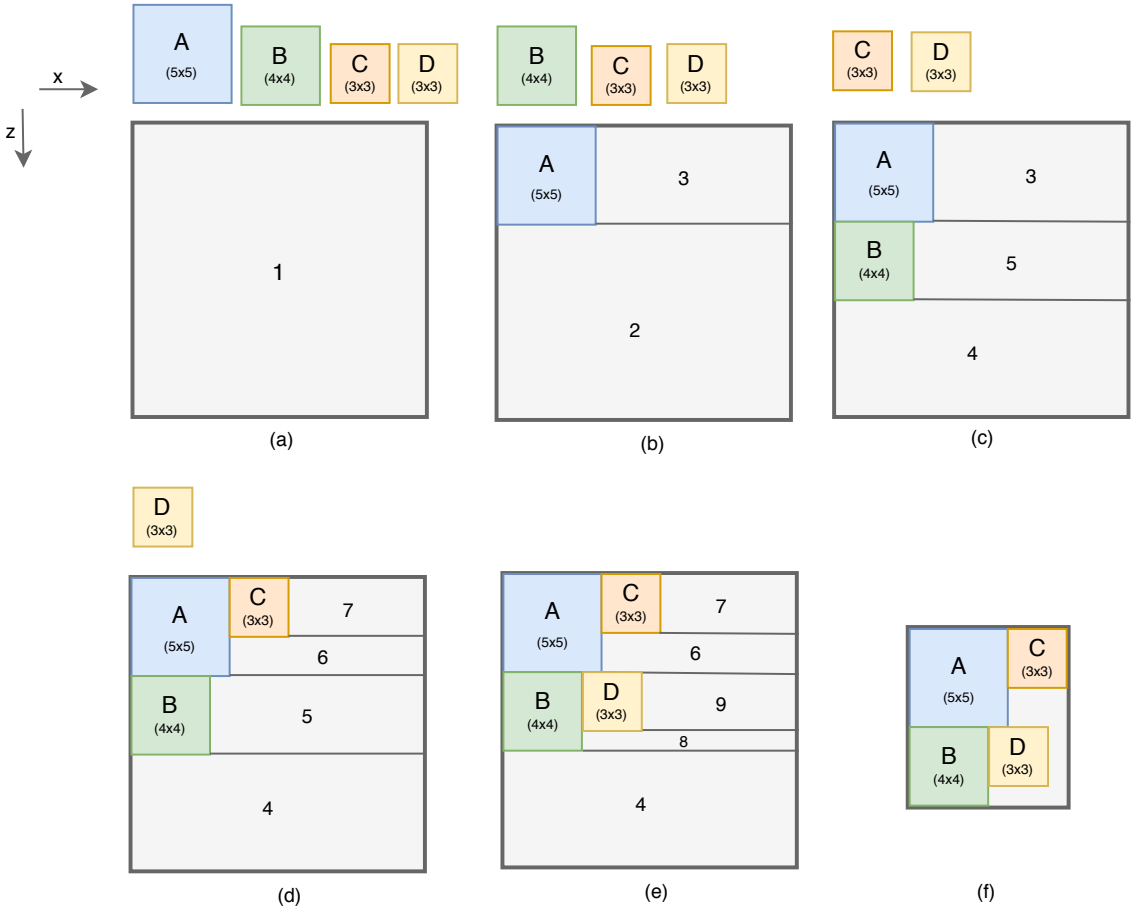


Figura 4.7: Exemplo ilustrativo do processo de alocação de edifícios.

Nesta figura, o espaço de alocação inicial (1) tem como largura o correspondente à soma das larguras dos objetos a alocar, neste caso, apenas quatro edifícios. Estes objetos já se encontram ordenados pela sua dimensão geométrica, respetivamente A, B, C e D. Começando pelo objeto de maior área (A), este será alocado no extremo mais próximo da origem, ou seja, no canto superior esquerdo do espaço (a). Esta alocação irá provocar um corte horizontal, dividindo o espaço restante em duas células retangulares, como visível em (b). O objeto (B) pode ser alocado em (2) ou em (3) porque a sua alocação irá sempre obrigar a um aumento igual do espaço de alocação, seja no eixo x ou no eixo z. Neste caso, o objeto é colocado o mais à esquerda, x = 0, criando um novo corte horizontal, como expresso em (c). O objeto (C) tem três células possíveis, mas é a (3) que mais favorece a construção de um quadrado de espaço total. Por fim, o objeto (D) pode ser alocado sem ser necessário o aumento do espaço de alocação, ou seja, a sua proporção. Finalizado o processo de alocação, o distrito assume a área total necessária para alocar todos os edifícios, (f).

No entanto, para facilitar a compreensão da visualização, os edifícios são separados por espaços, denominados de ruas. Neste sentido, os objetos edifícios na sua instanciação são reduzidos para o mínimo entre metade do seu tamanho ou em 1 unidade, no momento da sua alocação. Isto permite criar um espaço de contorno ao edifício cuja dimensão está dependente do seu tamanho e

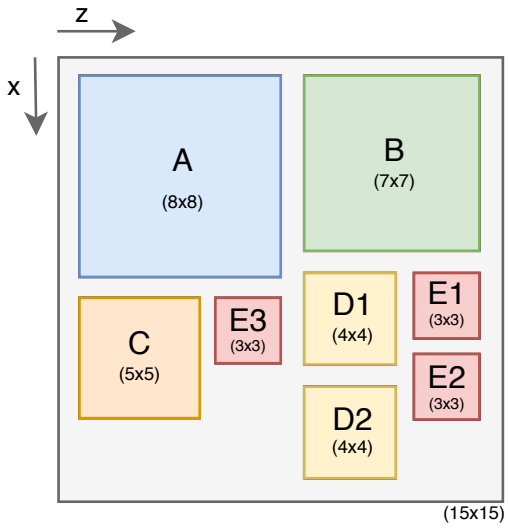


Figura 4.8: Exemplo ilustrativo de um conjunto de edifícios alocados.

é suficiente para permitir a construção de uma rua.

Num contexto com mais objetos, a Figura 4.8 revela a alocação de oito edifícios, instanciados pela ordem alfabética e, dentro desta, numérica. Atentando no objeto (E3), o último a ser processado, foi alocado numa área mais interior porque existia esse espaço e, principalmente, porque a proporção total do espaço já assumia a figura geométrica de um quadrado, aquilo que é desejado. Caso a proporção de um quadrado ainda não estivesse assumida no momento da instanciação do (E3) este procuraria uma posição que criasse ou aproximasse a uma proporção quadrada.

É ainda importante ressalvar que este último exemplo assume o eixo de coordenadas no Unity3D, ou seja, a postura assumida na visualização resultante. O processo de alocação explicado na Figura 4.7 assume uma postura de eixos invertida.

4.4 Compreensão da Informação

Com o objetivo de reduzir o esforço na compreensão do software, esta tarefa pode ser dividida numa compreensão conseguida pela simples visualização da cidade e outra em que o utilizador já interage com o sistema.

4.4.1 Compreensão pela Visualização

A tangibilidade criada com a metáfora da cidade permite assumir uma postura diferente para a compreensão de código, comparado com aquilo a que estamos habituados.

A visualização do software é uma facilitadora na compreensão da mesma se os mecanismos desenvolvidos conseguirem transmitir conteúdo apenas pela imagem. O ambiente virtual desenvolve uma imagem a três dimensões do sistema de software e não é apenas pela análise direta das métricas dos distritos e dos edifícios que se consegue obter informação, mesmo sem interação.

A visualização de *code smell* é um fator interessante porque o código fonte não consegue esconder a sua estrutura e situações como *large class* ou *feature envy* podem ser desvendadas com edifícios largos ou conexões repetitivas.

A aplicação a padrões de desenho também pode ser realizada de diversas formas. Por exemplo, através da identificação de classes interface, correspondendo a edifícios muito altos e finos, ou seja, com elevado número de métodos e um número reduzido de atributos.

Se o utilizador estiver posicionado num ponto superior ao ambiente virtual, terá acesso às componentes TextMeshPro[2], que colocam o nome da respetiva classe no topo esquerdo dos edifícios, facilitando a sua identificação sem necessitar de interação.

4.4.2 Compreensão pela Interação

Sendo um ambiente virtual *live*, as invocações que ocorrerem seriam impercetíveis, visto que, acontecem em milissegundos. De forma a ser possível a visualização e consecutiva análise, o motor gera as conexões no momento em que as recebe, no entanto, acrescenta-lhe 3 segundos de duração para que o utilizador tenha tempo necessário para se aperceber do que está a acontecer.

Adicionalmente, o utilizador tem ainda na sua posse outros fatores de controlo do tempo no menu do ambiente. Ou seja, o utilizar tem ao seu dispor os seguintes controlos:

- **Pausar** - O botão *Pause* bloqueia qualquer alteração no ambiente, seja com conexões ou com os distritos e pacotes. Este é o momento ideal para o utilizador fazer a sua análise pois terá total liberdade temporal.
  - ***Start Live*** - Após pausar, o utilizador poderá voltar ao *live*. Isto significa, volta para o tempo real, ignorando tudo o que possa ter acontecido no momento em que estava em pausa.
  - **Continuar** - Após pausar, o utilizador poderá ainda continuar com a execução no ponto seguinte ao que estava no momento que pausou a execução. Todos os eventos que o motor recebeu e não foram visualizados ficaram em cache e podem assim ser recuperados.
- **Recuar 1 segundo** - Apesar do atraso intencional criado nas alterações que ocorrem no ambiente, é possível que o utilizador perca algum pormenor ou que simplesmente deseje recuar no tempo. Esta funcionalidade permite exatamente isso, recuar um segundo. Após ser ativada, pede ao servidor os eventos que aconteceram no último segundo e volta a mostrá-los.

[2] TextMeshPro, https://assetstore.unity.com/packages/essentials/beta-projects/textmesh-pro-84126, Último Acesso em: 15/05/2018

No momento da conclusão desta dissertação, o ambiente virtual dispõe apenas dos botões acima referidos para o controlo temporal. No entanto, podem ser implementadas novas funcionalidades, incluindo novos botões, expandido as possibilidades de interação com o tempo.

4.5 Interface

A interação do utilizador com o ambiente virtual é possível utilizando apenas um monitor de um computador. No entanto, a visualização perde a sua imersividade e a interação torna-se praticamente impraticável pela não existência de controladores. Nesse sentido, aconselha-se o uso de dispositivos de realidade virtual com controladores de mãos e sensores, tais como HTC Vive ou Oculus Rift.

Na implementação dos conteúdos relacionados com RV foi utilizado o Virtual Reality Toolkit[3] que corresponde a uma biblioteca de funcionalidades preparadas para a plataforma Unity. O Toolkit suporta uma variedade de SDKs para execução, como o simulador de PC, HTC Vive, Oculus Rift, Daydream e Ximmerse.

4.5.1 Menu

O menu do ambiente virtual está fixo ao controlador esquerdo do *headset*, acompanhando sempre a sua posição. A interação com este deve acontecer através do apontador do controlador direito.

A Figura 4.9 apresenta os estados que o menu pode assumir. O texto informativo sobre Pacotes ou Classes e Invocações é habilitado ou desabilitado pelos botões físicos presentes no controlador direito, como será analisado na próxima secção. O *scroll* assumirá sempre o valor da posição y do visualizador, ou seja, a sua altura. O conjunto de botões apresentado poderá assumir dois aspetos. Os botões da Figura 4.9a estão disponíveis enquanto o ambiente está *live*, ou seja, a instanciar qualquer evento que receba. No entanto, se for acionado o botão *Pause*, os botões disponíveis passam a ser os da Figura 4.9b.

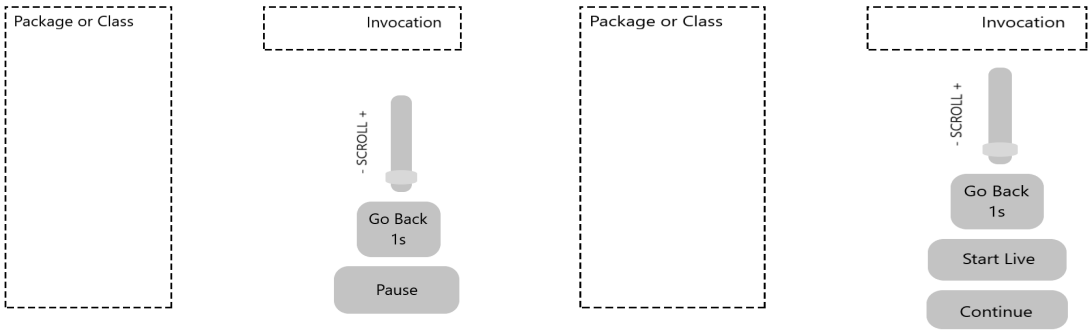
4.5.2 Manual de Utilização

O utilizador do ambiente virtual pode realizar diversas ações no momento da interação, como demonstrado no *Use Case* da Secção 3.4. Esta secção debruça-se na explicação de como podem ser realizadas essas ações.

Após o ambiente virtual ser iniciado, o utilizador iniciará a sua experiência de visualização. A Figura 4.10 expõe um exemplo da perceção no sistema gerada pelo Oculus Rift.

Os controladores de mãos assumem funcionalidades diferentes entre si. A Figura 4.11 exibe o dispositivo usado e a Figura 4.12 esquematiza as opções dos controladores.

[3] VRTK - Virtual Reality Toolkit, `https://vrtoolkit.readme.io/` Último Acesso em: 18/06/2018



(a) *Mockup* do menu com o *live* habilitado. (b) *Mockup* do menu com o *live* desabilitado.

Figura 4.9: *Mockups* das duas alternativas que o menu pode assumir.

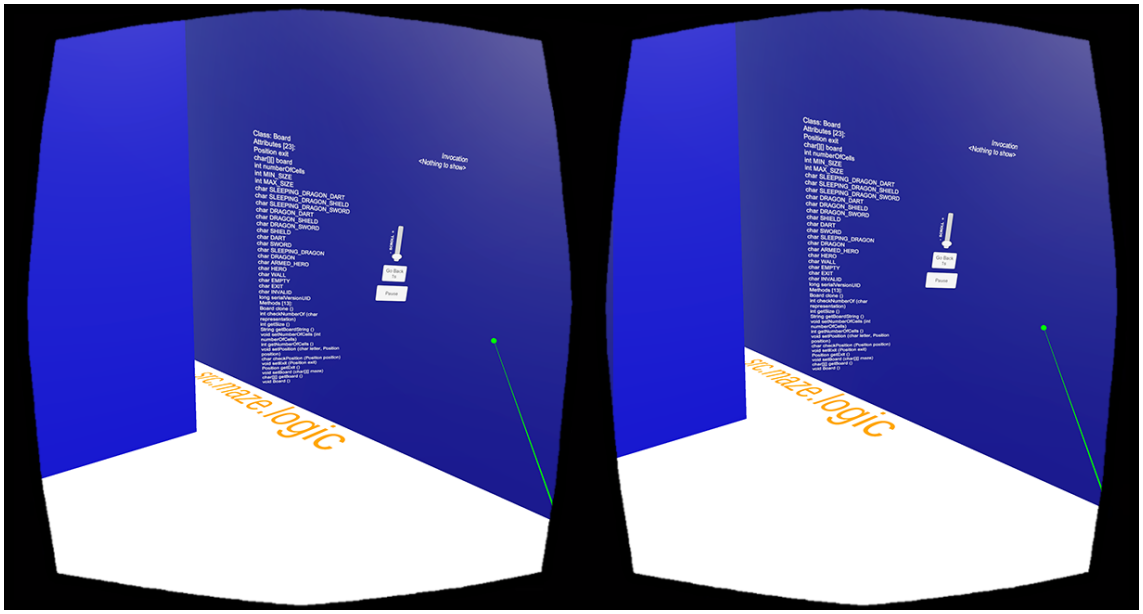


Figura 4.10: Esquema da visualização recorrendo a Oculus Rift.

Controlador Direita

O controlador de mão direita tem a função de seleção. Para isso basta premir o *touchpad* e apontar para o objeto que queremos analisar. Para interação com os botões ou com a barra de *scroll*, o utilizador deverá premir o *touchpad* e clicar no *trigger*. Por último, os botões A e B habilitam ou desabilitam a visualização do texto com informação dos distritos ou dos edifícios, bem como das conceções.

Controlador Esquerda

O controlador de mão esquerda será o plano de leitura da informação referida anteriormente. Ou seja, o texto acompanhará a mão. O *touchpad*, ao premir, possibilita a hipótese de fazer *teleport* pela cidade.

(a) *Headset* Oculus Rift. (b) Controladores do Oculus Rift.

Figura 4.11: Dispositivo Oculus Rift usado para a visualização e interação com o ambiente virtual.

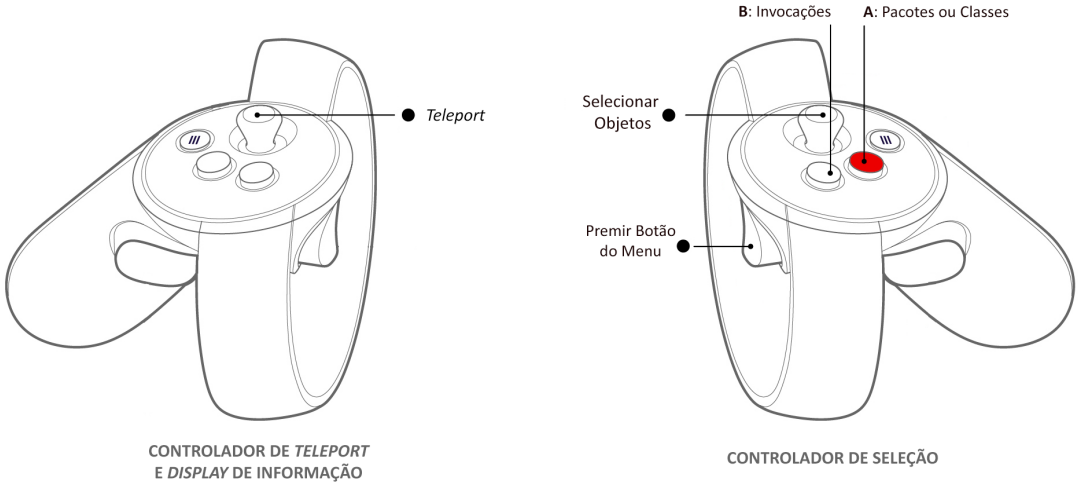


Figura 4.12: *Mockup* explicativa do funcionamento dos controladores do Oculus Rift.

4.6 Análise Comparativa

No final da Secção 2, foi elaborada uma tabela que comparava e analisava algumas tecnologias de visualização de informação, apresentadas de uma forma mais aprofundada na revisão da literatura.

CodeCity

O trabalho realizado por Wettel *et al.* [Wet10] foi uma das motivações a esta investigação em *live software development*. O seu trabalho foi denominado CodeCity.

Enquanto que CodeCity visualiza a estrutura estática do software, o ambiente virtual implementado nesta dissertação fornece uma visualização dinâmica da estrutura e ainda das invocações ocorridas. Ou seja, CodeCity apenas se concentra na visualização da estrutura e na evolução dos

sistemas de software, ignorando o comportamento, correspondendo à informação que pode ser extraída durante a execução do programa. A sua estratégia de alocação de objetos, distritos e edifícios, foi aplicada a este projeto por resolver os problemas do mesmo e pelo sucesso da sua aplicação já ter sido comprovado na tese de doutoramento de Wettel. CodeCity também não utiliza realidade virtual, ou seja, o seu conteúdo 3D é visualizado recorrendo a um monitor, onde a perda de imersão é evidente.

Em suma, o trabalho de Wettel consistiu numa importante base para a realização desta dissertação, sobretudo nos conceitos de aplicação da metáfora visual. No entanto, o motor de ambiente virtual introduz o *live*, que permite que a análise ocorra dinamicamente em concordância com a execução do sistema de software.

VRCity

Já em 2017, foi apresentado por Vincur *et al.* [VNP17b] o software de análise VRCity.

VRCity aplica uma versão modificada da metáfora visual a um ambiente 3D recorrendo à realidade virtual. Neste caso, a vantagem da visão estereoscópica é utilizada. As alterações ao código fonte e as invocações de métodos na execução do programa são analisadas por esta ferramenta, à semelhança da implementada nesta dissertação. No entanto, não acontecem de forma *live*, ou seja, os utilizadores apenas podem visualizar gravações de execuções do programa. Os utilizadores, podem ainda, ver como o software evolui em períodos de tempo selecionados. As animações são baseadas numa sequência de *commits* que são apresentados com o seu respetivo autor.

A estrutura do software é representada por um grafo e a posição dos objetos é definida seguindo a curva de Hilbert.

Comparativamente com o projeto desta dissertação, VRCity faz uso da realidade virtual para a visualização do conteúdo 3D e a análise dos aspetos dinâmicos é conseguida por aplicação de animações, sem recorrer ao *live feedback*.

4.7 Sumário

A abordagem seguida para a visualização da informação proveniente do software combinou as representações da metáfora visual com a realidade virtual, sem nunca descorar o *live* e a sua importância de permitir o rápido *feedback* ao utilizador.

O algoritmo de alocação dos objetos no espaço virtual revela-se importante na medida em que o sucesso das suas premissas favoreça as tarefas de compreensão e manutenção. Este deve criar uma cidade com uma postura que minimize o espaço virtual necessário, mantendo uma geometria a mais aproximada de um quadrado e instanciando os objetos pela sua área.

Este capítulo confirmou ainda que é possível retirar mais valias apenas recorrendo à análise visual e utilizando um monitor convencional. No entanto, utilizando RV, as interações passam a ser possíveis na sua totalidade, seja para mover o utilizador pela cidade, seja para conhecer ou controlar tudo o que está a acontecer, estático ou dinâmico, através de botões que lhe dão total controlo espacial e temporal.

As opções tomadas relativamente ao controlo da realidade virtual, em especial nos controladores do dispositivo, tiveram o intuito de criar a maior simplicidade, de forma a facilidade a utilização por programadores sem experiência com este tipo de tecnologia. No entanto, nunca foi descorada a pertinência das opções, pois considera-se que todas as funcionalidades implementadas são as suficientes para ponto de partida na investigação de *live software development*, em especial, as que permitem o controlo temporal.

Outras formas de interação com recurso à realidade virtual podem ser aplicadas de maneira a enriquecer o uso da ferramenta, tais como, a utilização de gestos pelo utilizador. Fazendo uso dos sensores dos dispositivos de RV, simples gestos como levantar os dois braços podem ser traduzidos em subir a posição do visualizador, ou seja, aumentar o valor da coordenada y, por exemplo.

Capítulo 5

Avaliação

A ferramenta desenvolvida e apresentada nesta dissertação tem como objetivo a redução do esforço na compreensão de um sistema de software. De forma a conseguir explorar e validar o desempenho da ferramenta no seu objetivo, a mesma foi sujeita a uma experiência controlada.

Nesse sentido, é escrutinada a abordagem seguida com a aplicação de uma metáfora visual num ambiente 3D recorrendo a realidade virtual.

Neste capítulo, é explorada a avaliação, um conjunto de linhas orientadoras e o planeamento que deve ser assumido na experiência, as tarefas realizadas e os respetivos resultados.

5.1 Avaliação Empírica

O ambiente desenvolvido para a exploração de sistemas de software tem como base a visualização e a sua interpretação como o principal meio para conseguir atingir o objetivo de facilitar a compreensão de software.

De acordo com a metáfora visual seguida e explorada na secção 4.3.1, os sistemas de software são projetados como cidades, onde os edifícios representam classes desse sistema e cujos distritos representam os seus pacotes. Tudo isto é construído explorando métricas de software que podem ser extraídas.

A avaliação pode revelar-se complexa para sistemas de maior dimensão numa primeira experiência por parte do utilizador.

O processo de avaliação deve partir de uma compreensão alto nível, reconhecendo e explorando o ambiente virtual.

Salienta-se como maior preocupação para a avaliação a inexperiência por parte dos participantes no uso de dispositivos de realidade virtual, tais como Oculus Rift e HTC Vive, que pode levar a um tempo de adaptação maior, seja pela necessidade de conhecer as funcionalidades ou pelo entusiasmo de explorar um novo equipamento. No entanto, sendo a realidade virtual uma tecnologia de recente disponibilidade e ainda em exploração pelas grandes empresas, revela-se importante que a maioria dos participantes não tenha essa experiência de utilização, pois representam, em determinada medida, o público alvo do ambiente virtual.

O ceticismo quanto à aplicação da realidade virtual deve ser anulado no momento em que é experienciado a facilidade da movimentação do participante no ambiente virtual, ou seja, na cidade criada a partir do sistema de software.

5.2 Objetivos

Na exploração da literatura, em particular no trabalho realizado por Wettel *et al.* [WLR11], foi possível definir linhas orientadoras para conseguir atingir os objetivos desta dissertação na experiência realizada. Nesse contexto, a avaliação experimental focou-se nas áreas de Engenharia de Software, Visualização de Software e Realidade Virtual, devendo abordar os seguintes pontos.

- **Todos os participantes serem da área de informática**. Para efeitos comparativos é importante que o participante tenha presente o comum mecanismo de compreensão de sistemas de software, ou seja, analisar código. Assim, todos os participantes devem ter experiência em desenvolvimento de software.

- **Envolver participantes com várias experiências**. Neste contexto, define-se a experiência do participante como o seu grau académico concluído. Este aspeto revela-se interessante por facultar várias visões do contributo científico dado pela ferramenta e não foi considerado fator prejudicial aos resultados porque a principal influência corresponde à experiência de programação, que não se relaciona com o grau académico.

- **Diferentes tipos de estudo**. Como a maioria dos estudos empíricos acontecem com estudantes em ambiente controlados é necessário que esse estudo contribua para o processo de aprendizagem dos estudantes e que tenha objetivos educacionais [Woh07].

- **Motivação do participante**. Recorrer a novas tecnologias como o caso de dispositivos de realidade virtual e a temas novos ou que estão nos interesses dos participantes, acrescenta uma motivação extra à realização da experiência [SOT09].

- **Fornecer um tutorial exploratório da ferramenta**. Uma comparação fidedigna e justa da ferramenta necessitaria de treino anterior à experiência, de forma a dotar os participantes dos

conhecimentos básicos ao uso do hardware e do software. Esse tutorial deve ser realizado, se possível, num momento afastado ao da experiência. [WLR11, MCS05].

- **Permitir aos participantes conhecer e perceber a ferramenta**. Deve existir uma fase de aprendizagem, de forma a que o participante extraia o máximo de conhecimentos da ferramenta [SOT09].
- **Limitar a duração da experiência**. Os participantes devem ter uma duração máxima para a realização do teste e serem informados acerca desse tempo limite [SOT09].
- **Caso de estudo**. Identificar casos de projetos de estudantes para usar na investigação, sem interferir com os objetivos educacionais [Woh07].
- **Relação com a ferramenta**. De forma a maximizar a confiança nos resultados obtidos, a experiência dos participantes na ferramenta deve ser praticamente nula [SOT09].

5.3 Planeamento da Experiência

O objetivo da experiência controlada consiste em levar o participante a experienciar situações de *liveness* com recurso à realidade virtual de forma a observar se existe uma redução do esforço na compreensão de um sistema de software.

Dispositivo

Para interação com a realidade virtual foi utilizado o Oculus Rift.

Local

A experiência foi realizada em ambiente fechado numa sala do Departamento de Informática da Faculdade de Engenharia da Universidade do Porto. A sala dispunha de um monitor, que transmitia o que era visualizado pelo participante, e de um espaço livre de 3x3 metros para a movimentação do utilizador. Se o participante se aproximasse dos limites do espaço livre, uma grelha delimitadora apareceria no ambiente virtual, alertando-o para se afastar.

Questionário

A experiência consistiu no preenchimento de um questionário que foi interrompido por três vezes, de forma propositada, para a realização de tarefas práticas, explicadas na secção 5.4. O questionário iniciou-se pedindo alguns dados para caracterização do participando, experiência na área e outras questões relacionadas com o tema desta dissertação. Seguiu-se o período de questões relacionadas com as tarefas. Por fim, foram apresentadas questões relacionadas com a interação e de usabilidade.

Participantes

Todos os intervenientes nos testes participaram de livre vontade e com o interesse de conhecer e explorar a ferramenta. A amostra incluiu apenas participantes que estão diretamente integrados na área de informática com forte componente de programação, mais concretamente estudantes, investigadores e professores. Ou seja, todos os participantes são do mundo académico, apesar de alguns possuírem percursos profissionais paralelos na indústria.

Foi assinado por todos os intervenientes um consentimento de participação após serem informados do contexto da experiência, ausência de riscos para a saúde e confidencialidade dos dados. O formato do documento está presente no Anexo A.

Duração

A literatura previne a necessidade de existir um tempo adequado para exposição à ferramenta por parte dos participantes, que pode ir de algumas horas até semanas [SOT09, SOT08, Pla04]. A experiência deve influenciar o tempo de exposição, sendo que participantes com uma experiência reduzida devem ter um maior prévio contacto com a ferramenta [MCS05].

Limitar a duração temporal da experiência é um fator valorizado para os participantes porque lhes permitem uma maior gestão do seu tempo. Revela-se também importante para os testes porque cria a mesma base de tempo para a realização das tarefas a todos os participantes. Sensalire *et al.* investigou durações médias de experiências que remontam a várias horas [SOT09].

Esta investigação decidiu atribuir uma duração máxima de 25 minutos à experiência.

Exposição à ferramenta

Todos os participantes mostraram interesse na realização da experiência por considerarem o objetivo da ferramenta pertinente e útil. No entanto, não tinham qualquer conhecimento do funcionamento da mesma, nem tiveram qualquer exposição prévia.

Como tal, os participantes receberam, antes de iniciar a experiência, um tutorial de utilização da ferramenta e principais funcionalidades.

A primeira tarefa imposta na experiência tinha o objetivo de expor o participante a todas as funcionalidades do ambiente e obter *feedback* quanto à facilidade em interagir ou visualizar elementos do sistema de software. O tempo dado a essa exposição foi o necessário para o participante conhecer e compreender as funcionalidades, não obrigando a uma experiência repetitiva e minuciosa.

Integridade dos dados

Todos os participantes foram ainda informados da autoria do projeto, no momento anterior à realização dos testes. De forma a tentar maximizar o criticismo e reduzir a generosidade que os participantes pudessem ter nas respostas, o que teria influência nos dados [SOT09], foi incentivada a crítica dado que todas as respostas seriam anónimas.

5.4 Tarefas de Teste

Os participantes foram convidados a realizar três tarefas, acompanhadas por conjuntos de questões, com o objetivo de avaliar diferentes situações no ambiente virtual.

Tarefa T1: **Reconhecimento e experienciação das funcionalidades e controlos no ambiente virtual**.
O participante deve aprender a utilizar as funcionalidades disponíveis e os controlos do dispositivo de RV. Após esta fase introdutória, o objetivo é este experienciar a visualização de todos os componentes e as interações possíveis. A Figura 5.1 ilustra vários participantes realizando a T1 na sua primeira exposição à ferramenta.

Figura 5.1: Participantes na primeira exposição ao ambiente virtual.

Tarefa T2: **Identificação de um ciclo infinito que impede a correta execução do sistema.**
O participante tem o objetivo de identificar um ciclo infinito através da visualização. Para isso fará uso dos conhecimentos obtidos na primeira tarefa e de toda a informação que é possível obter pelo ambiente virtual. A Figura 5.2 demonstra a resposta do ambiente virtual ao ciclo infinito provocado. É notório que as invocações dentro do ciclo acontecem de forma consecutiva e sem qualquer interferência do participante e sem possibilidade de as terminar. O participante é informado que do lado do IDE não existe qualquer erro e o sistema está bloqueado.

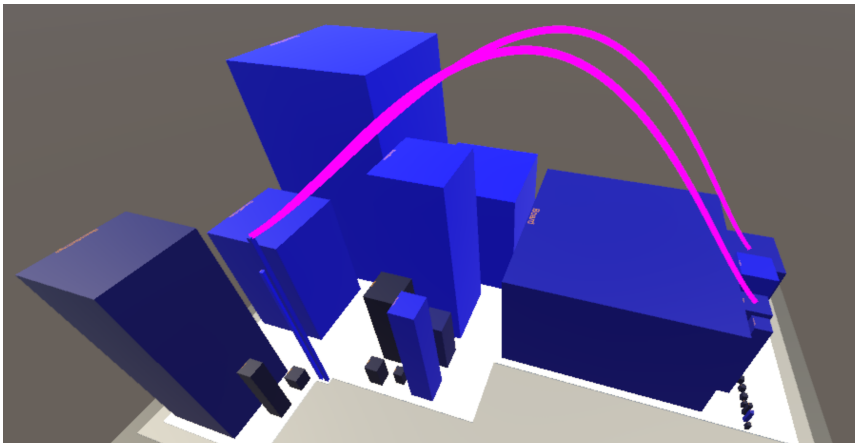


Figura 5.2: Visualização das invocações em ciclo infinito na tarefa 2.

Tarefa T3: **Identificação da classe que invocou um método com pelo menos um argumento a *null*.**
O participante tem o objetivo de identificar a classe que enviou um argumento a *null* numa invocação. A representação visual com o auxílio de som deve permitir o sucesso da tarefa. A Figura 5.3 representa a visualização da classe que realizou uma invocação com um argumento nulo. A sua cor fica intermitente entre vermelho e a sua própria cor. Em adição à mudança de cor, é despoletado um alerta sonoro nessa classe para que o participante seja alertado do acontecimento. O alerta sonoro é especialmente útil quando a classe que realizou a invocação não está no campo de visão do participante.

Atributos de Interesse

Aos testes realizados nas três tarefas e na resposta ao questionário foram integrados dois atributos de interesse com o objetivo de avaliar a ferramenta.

- **Duração**. Cada uma das três tarefas foi cronometrada de forma a comparar o tempo necessário à realização das tarefas. Esta métrica revela-se importante na comparação entre participantes e na construção da média.

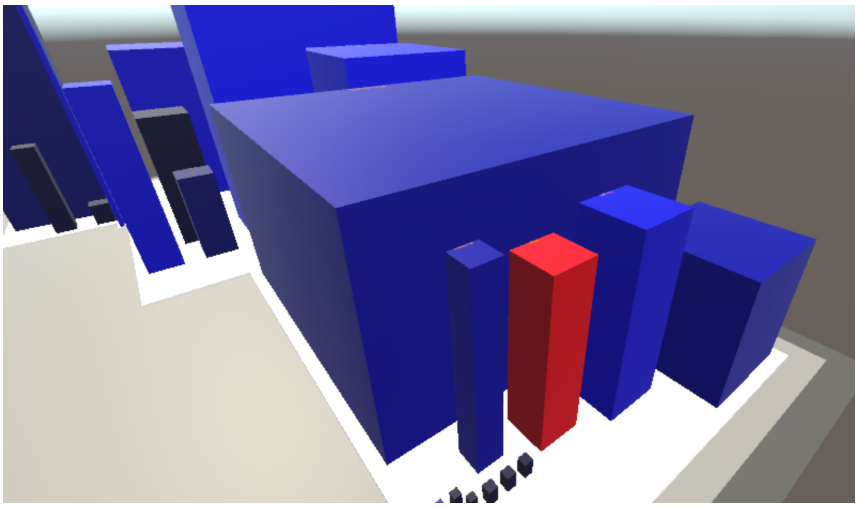


Figura 5.3: Visualização da classe que realizou uma invocação com um argumento nulo na tarefa 3.

- **Dificuldade**. Representa a perceção obtida pelo agente de controlo, o autor da dissertação, durante a execução de cada teste por parte dos participantes. Este dado é quantitativo e representará uma classificação de 0 a 5, sendo que 5 corresponde a uma tarefa realizada integralmente e com um bom desempenho.

Em adição às respostas dadas pelos participantes nos questionários que responderam durante os testes, estes atributos são do interesse da investigação na medida em que visam levantar aspetos indiretamente controlados pelos participantes. Os participantes não tinham noções de tempo no ambiente virtual e não souberam qual foi a classificação dada pelo agente de controlo quanto à dificuldade que mostraram ter na resolução das tarefas. A duração é um atributo quantitativo, sendo fácil a sua medição, e a dificuldade é um atributo qualitativo, cuja atribuição depende da opinião do agente de controlo.

5.5 Resultados

A experiência controlada de estudo de utilização foi realizada por 25 participantes com competências de engenharia de software e integrados no mundo académico.

Os resultados são baseados nas respostas dadas ao questionário que acompanhou a realização da experiência e à opinião do agente de controlo da experiência.

A experiência foi acompanhada pela resposta a determinadas questões referentes ao momento em questão.

Todas as questões relacionadas com as tarefas ou sobre a avaliação da ferramenta foram baseadas na escala de Likert dividindo as respostas possíveis em “Discordo totalmente”, “Discordo”, “Não tenho opinião”, “Concordo” e “Concordo totalmente"ou "Muito difícil", "Difícil", "Não tenho opinião", "Fácil"e "Muito fácil". Para efeitos de análise, foram atribuídos valores de 1 a 5,

Tabela 5.1: Número de participantes por género.

Género	Nº de Participantes
Masculino	19
Feminino	6

respetivamente aos dois casos anteriores, sendo "1" a avaliação mais negativa e "5" a avaliação mais positiva.

Caracterização e Experiência

A Tabela 5.1 relaciona a participação consoante o género. As idades estão compreendidas entre os 18 e os 36 anos ($\bar{x} = 23.44 \pm 3.15$). A relação entre os graus académicos é referida na Figura 5.4. O fator género e o fator idade são considerados pouco significativos para a experiência. No entanto, o fator considerado preponderante foi a experiência em programação. Todos os participantes confirmaram ter experiência, entre os quais 88% programam praticamente de forma diária.

Mais de metade dos participantes confirmou que, em média, costuma sentir alguma dificuldade na compreensão de sistemas de software (76%) e quase todos costumam recorrer a ferramentas de auxílio para a compreensão dos sistemas (92%). No entanto, 88% dos participantes nunca usaram ferramentas visuais para auxiliar na compreensão de código.

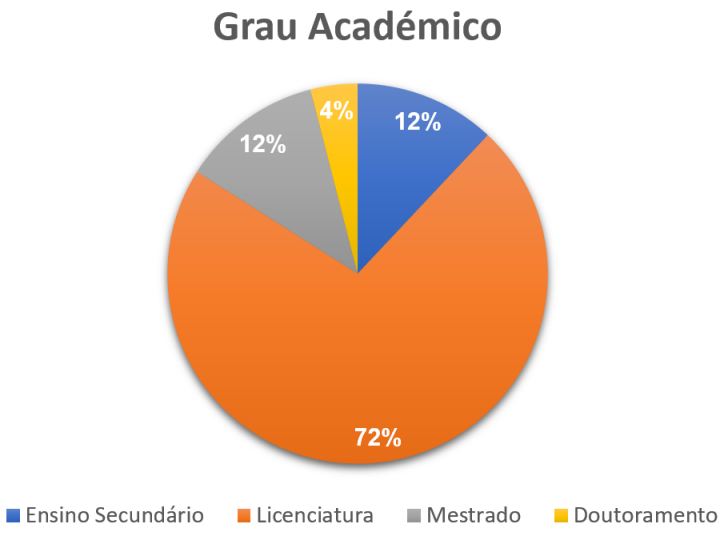


Figura 5.4: Grau académico concluído pelo participante.

Experiência com Oculus Rift

A familiarização dos participantes com o uso do dispositivo usado para efetuar as tarefas, o *VR Headset* Oculus Rift, em especial com os seus controladores está destacado na Tabela 5.2.

Tabela 5.2: Número de participantes com experiência com os Oculus Rift.

Experiência com Oculus Rift	Nº de Participantes
Nenhuma	18
Alguma	4
Bastante	3

Este dado é considerado de extrema importância, porque a experiência de utilização com Oculus poderá levar a uma maior dificuldade de adaptação ao sistema, seja no uso dos controladores, seja na experiência da imersividade da realidade virtual. Em relação a este último ponto, um pequeno número de participantes confidenciou que se tratava da sua primeira experiência em RV.

No entanto, a utilização de participantes sem experiência em ambientes virtuais não foi deixada ao acaso, porque é expectável que utilizadores num contexto real tenham também pouca experiência com *headsets* de RV, pelo motivo da realidade virtual ser ainda um aspeto em inovação e pouco explorado.

Tarefa 1 - T1

A primeira de três tarefas realizadas foi explicada na Secção 5.4. A T1 expõe, pela primeira vez, o participante ao ambiente virtual e às suas funcionalidades.

A Figura 5.5 mostra as respostas dadas quanto à realização da T1. Pela sua análise, quase a totalidade dos participantes concordaram com a facilidade de execução na mesma.

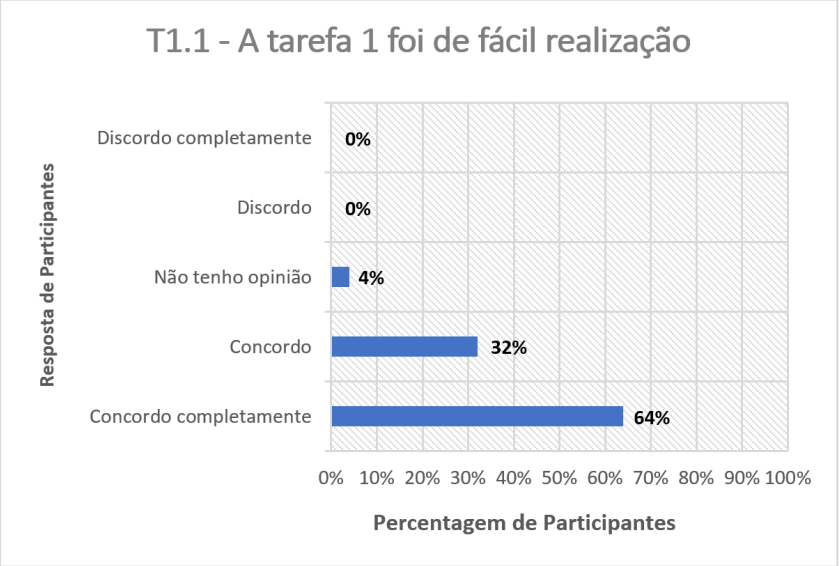
As atividades de reconhecimento e de exploração das principais funcionalidades foram também classificadas quanto à sua facilidade de utilização. As Figuras 5.6 e 5.7 reproduzem resultados da T1.2a e da T1.2b.

Em relação à T1.2a denota-se uma menor facilidade em identificar pacotes. Tal situação deve-se à pequena dimensão do sistema em testes que continha apenas 3 níveis de pacotes, ou seja, 3 níveis de distritos, e, adicionalmente, todos os edifícios estavam ao mesmo nível de distrito.

A tarefa T1.2b obrigava a uma maior interação com os controladores do *headset*, o que criou uma maior dificuldade nos participantes que tinham uma menor experiência de utilização do Oculus. A movimentação no sistema também saiu pouco considerada, porque os participantes não utilizavam a movimentação física possível. Ou seja, os participantes tinham um espaço físico onde se podiam mover, em adição à funcionalidade de teleporte.

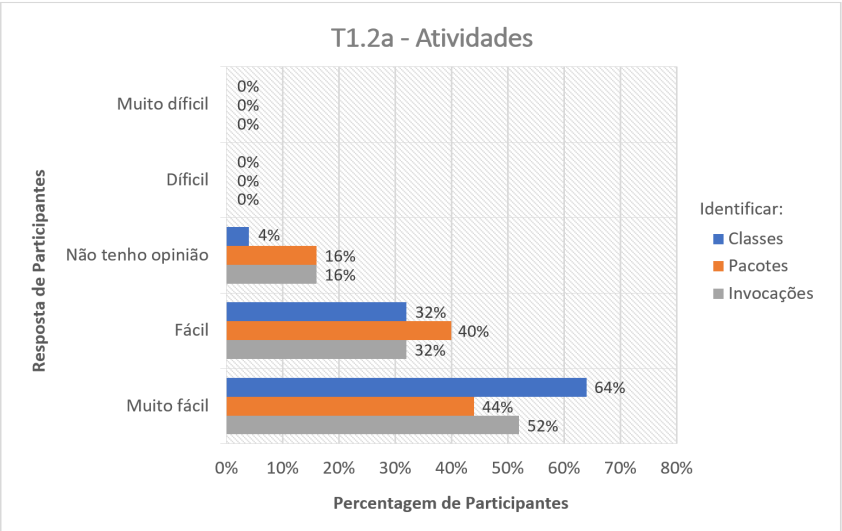
A avaliação do agente controlador da experiência quanto à dificuldade demonstrada pelos participantes na resolução da tarefa 1 teve uma classificação média de $\bar{x} = 4.56 \pm 0.65$. Este valor representa a perceção obtida pelo agente e é classificada com uma pontuação de 1 a 5. Dado este resultado, foi considerado que a tarefa foi realizada de forma positiva e integralmente por todos os participantes.

Em suma, os questionários realçam uma facilidade na utilização da ferramenta para utilizadores expostos a ela pela primeira vez.



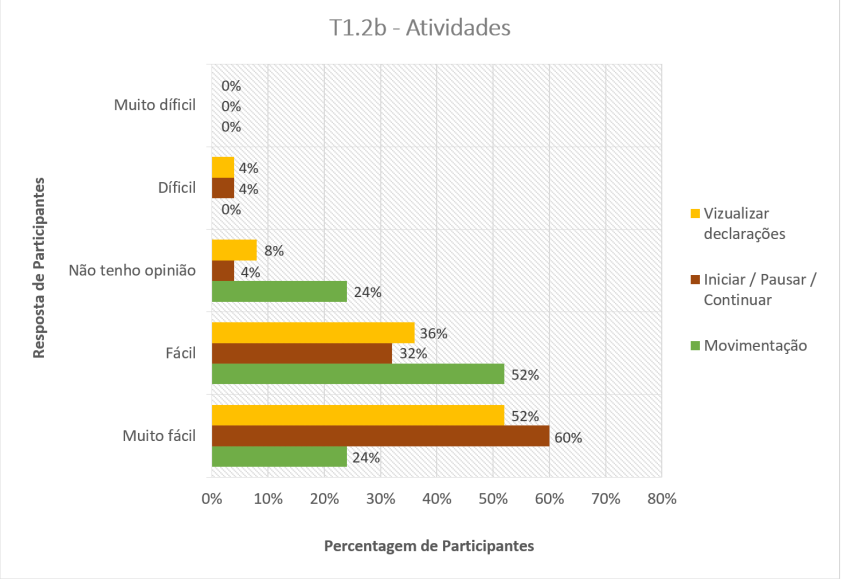
T1.1	1º Quartil	Mediana	3º Quartil
	4	5	5

Figura 5.5: Facilidade na realização da Tarefa 1.



T1.2a	1º Quartil	Mediana	3º Quartil
Identificar classes	4	5	5
Identificar pacotes	4	4	5
Identificar invocações	4	5	5

Figura 5.6: Facilidade na realização das atividades na Tarefa 1 (a).



T1.2b	1º Quartil	Mediana	3º Quartil
Visualizar declarações	4	5	5
Iniciar/Pausar/Continuar	4	5	5
Movimentação no sistema	4	4	4

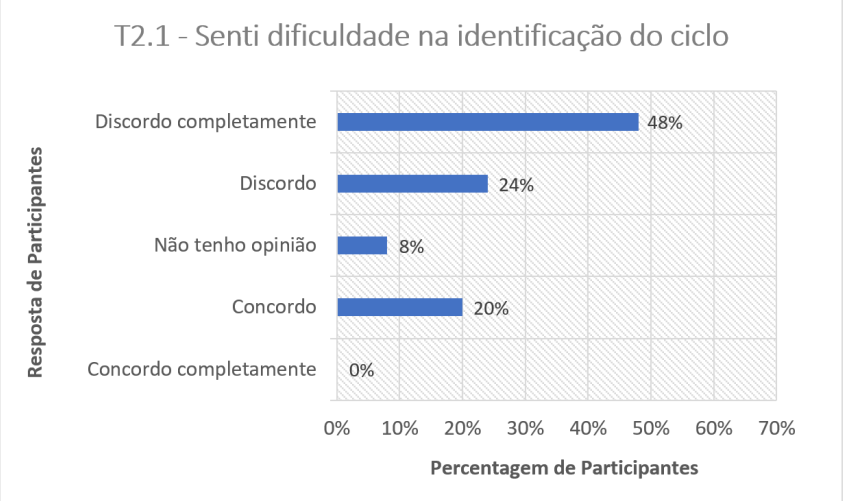
Figura 5.7: Facilidade na realização das atividades na Tarefa 1 (b).

Tarefa 2 - T2

A tarefa 2 levava o sistema de software em análise a cair num ciclo infinito. Ou seja, o ambiente ficava com as mesmas invocações sempre em execução e o IDE ficava bloqueado, mas sem apresentar qualquer erro ou aviso.

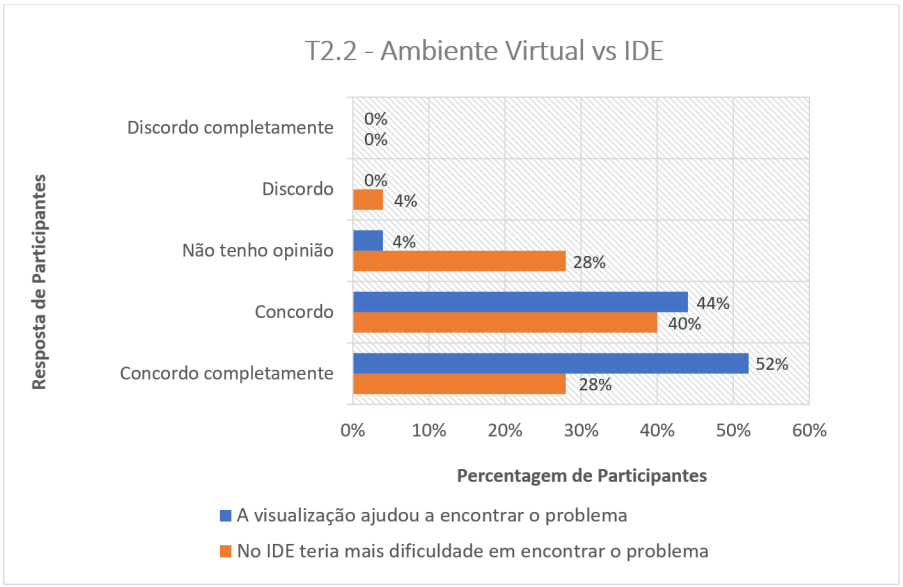
Após a realização da tarefa, os participantes, na maioria, responderam que não sentiram dificuldade na identificação do ciclo, Figura 5.8. Por outro lado, os principais impedimentos à perfeita identificação do ciclo infinito foi a facilidade do mesmo. Alguns participantes confidenciaram que a resposta ciclo infinito foi a primeira impressão que tiveram, mas supuseram que poderia ser algo mais complexo, decidiram explorar e examinar as invocações antes de responder, levando a um aumento da duração da experiência.

A oposição realizada pelo ambiente virtual ao IDE corrente do participante, para a realização desta tarefa, foi demonstrada na Figura 5.9. Trata-se de uma oposição objetiva considerada pelo participante. Pela análise, os participantes consideram que a visualização ajudou a encontrar o problema e que no seu IDE teriam mais dificuldade a encontrar o mesmo problema. Isto acontece, porque no IDE não há qualquer tipo de *feedback* ao utilizar que explicite o que está a acontecer e onde está a acontecer, levando o utilizador a escrutinar o código fonte ou a usar ferramentas adicionais. O ambiente virtual irá representar as invocações que estejam em ciclo e facilmente o utilizador se apercebe do que está a acontecer, o local e qual a chamada de método.



T2.1	1º Quartil	Mediana	3º Quartil
	1	2	3

Figura 5.8: Dificuldade sentida na realização da Tarefa 2.



T2.2	1º Quartil	Mediana	3º Quartil
IDE	3	4	5
Visualização	4	5	5

Figura 5.9: Comparação entre visualização e IDE na Tarefa 2.

A avaliação do agente controlador quanto à dificuldade demonstrada na resolução da tarefa 2 teve uma classificação média de $\bar{x} = 4.24 \pm 0.78$. O resultado médio é inferior ao obtido pelos

participantes na tarefa anterior, visto que esta tarefa apresenta mais dificuldades na sua realização, exigindo um maior raciocínio.

Em suma, o participante considerou vantajosa a utilização do ambiente virtual na realização deste tipo de atividades.

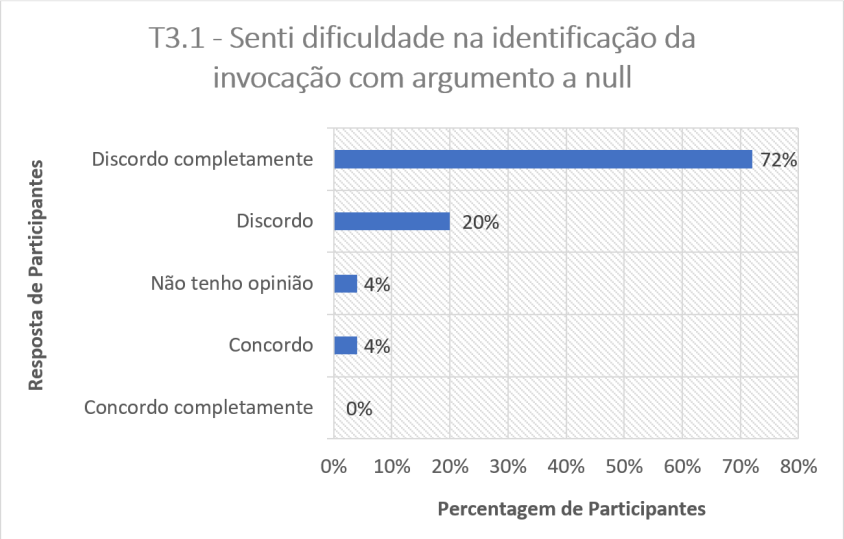
Tarefa 3 - T3

A tarefa 3 procurava determinar a capacidade dos participantes em encontrarem um objeto, neste caso, um edifício com a cor intermitente a vermelho, ou seja, um edifício que numa das invocações a um método que a sua classe realizou, um dos argumentos estava a *null*.

A solução seria os participantes utilizarem a funcionalidade de *scroll*, que permitia aumentar a altura do visualizador em relação à cidade. Se os participantes se posicionassem em cima do edifício mais alto não conseguiriam ver o objeto, pelo que esta estratégia não permitiria encontrar o edifício a vermelho intermitente.

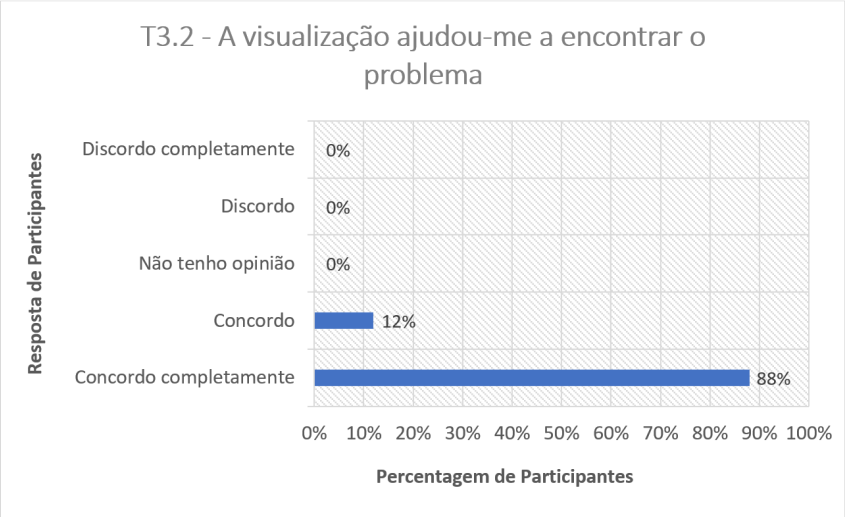
Pela análise das Figuras 5.10 e 5.11 é possível concluir que a maioria dos participantes não sentiu dificuldade na realização da tarefa e considera que a visualização ajudou a encontrar o problema.

A maioria dos participantes acabaram por encontrar a solução mais prática e usaram a funcionalidade de *scroll*. Alguns outros decidiram percorrer todas as ruas da cidade pelo solo e tiveram mais dificuldade em encontrar o objeto e a duração da experiência foi maior.



T3.1	1º Quartil	Mediana	3º Quartil
	1	1	2

Figura 5.10: Dificuldade sentida na identificação de um *null* na Tarefa 3.



T3.2	1º Quartil	Mediana	3º Quartil
	5	5	5

Figura 5.11: Contribuição da visualização para a Tarefa 3.

A avaliação do agente controlador quanto à dificuldade demonstrada na resolução da tarefa 3 teve uma classificação média de $\bar{x} = 4.84 \pm 0.37$. A tarefa tinha um baixo grau de dificuldade. A classificação média não foi máxima porque alguns participantes demoraram a perceber que uma vista superior da cidade era o mecanismo mais fácil de responder à tarefa.

Em suma, a implementação desta funcionalidade foi considera útil na tentativa de evitar que aconteçam exceções *null*. Em adição, o som emitido pelo edifício que invoca um método com um argumento a *null* é um fator extra na localização do edifício porque o som propaga no espaço podendo ouvir-se mais longe ou mais perto. A funcionalidade de som não foi utilizada nesta experiência.

Avaliação pelos participantes ao Ambiente Virtual

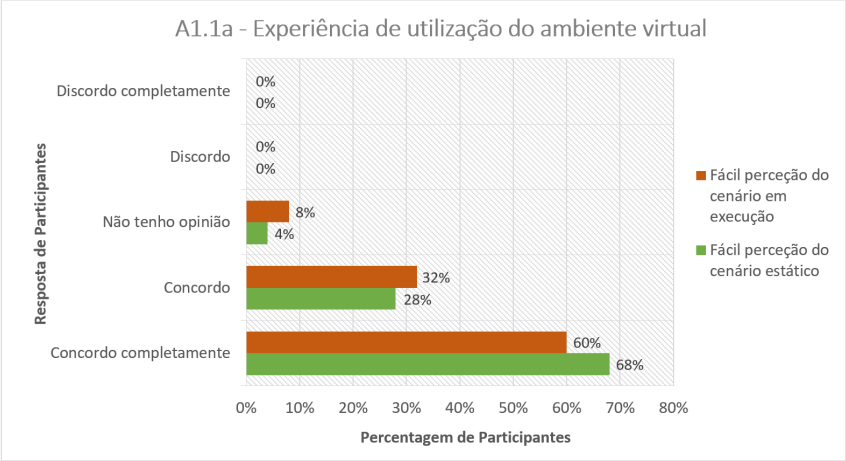
Após a conclusão das tarefas, os participantes avaliaram a experiência de utilização do ambiente virtual, Figuras 5.12 e 5.13.

O conjunto A1.1a questionou quanto à perceção da informação estática e da informação em execução. Os resultados são semelhantes entre si e confirmam a facilidade de perceção para ambos os cenários. Os participantes conseguiram obter as informações desejadas a partir da experiência.

O conjunto A1.1b expõe quatro declarações a que os participantes responderam. A utilidade dos botões foi confirmada para a grande maioria dos participantes à exceção de um que considerou que os botões não deviam constar no ambiente virtual, mas sim serem botões físicos presentes nos controladores do Oculus. A utilização das cores destacou-se positivamente por ser mais uma forma de exprimir informação sem ser necessária qualquer interação com os objetos. Quanto à facilidade de utilização e à interface, apesar da maioria das respostas positivas, os participantes

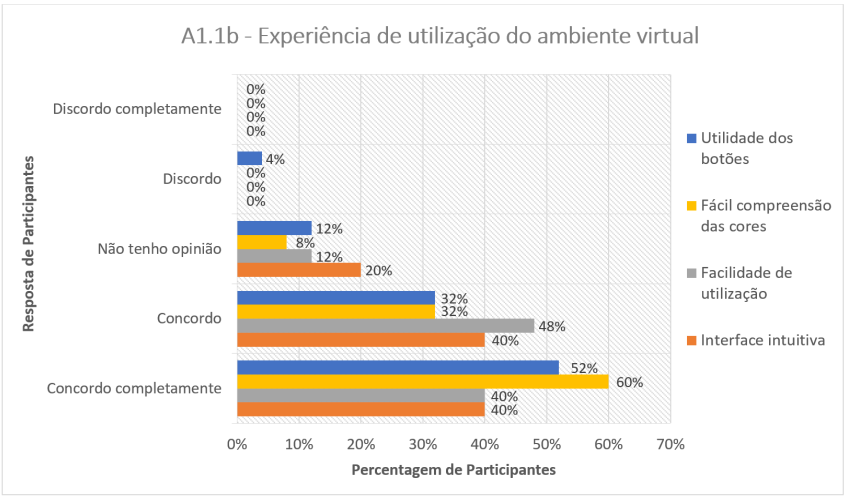
consideraram que um tempo de exposição maior favorecia a utilização da ferramenta e a interação com a interface.

Relativamente ao interesse dos participantes em voltar a utilizar a ferramenta, 96% concordou, ou seja, apenas um participante não demonstrou opinião quanto a esta declaração, Figura 5.14.



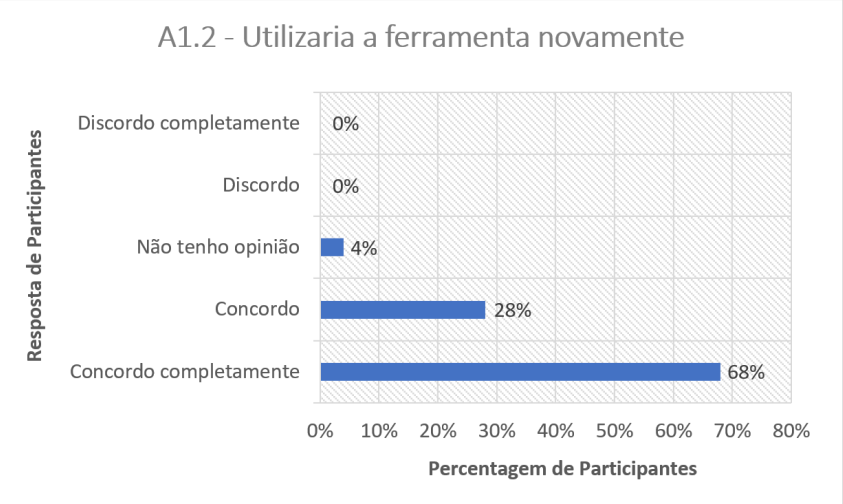
A1.1a	1º Quartil	Mediana	3º Quartil
Cenário Execução	4	5	5
Cenário Estático	4	5	5

Figura 5.12: Experiência de utilização do ambiente virtual (a).



A1.1b	1º Quartil	Mediana	3º Quartil
Utilidade dos botões	4	5	5
Compreensão das cores	4	5	5
Facilidade de utilização	4	4	5
Interface intuitiva	4	4	5

Figura 5.13: Experiência de utilização do ambiente virtual (b).



A1.2	1º Quartil	Mediana	3º Quartil
	4	5	5

Figura 5.14: Interesse em utilizar a ferramenta novamente.

Este interesse revela-se positivo por os participantes na experiência com RV, em alguns casos na sua primeira experiência, não terem sentido desconforto na utilização e por considerarem uma ideia com potencial para ser usada em contextos reais.

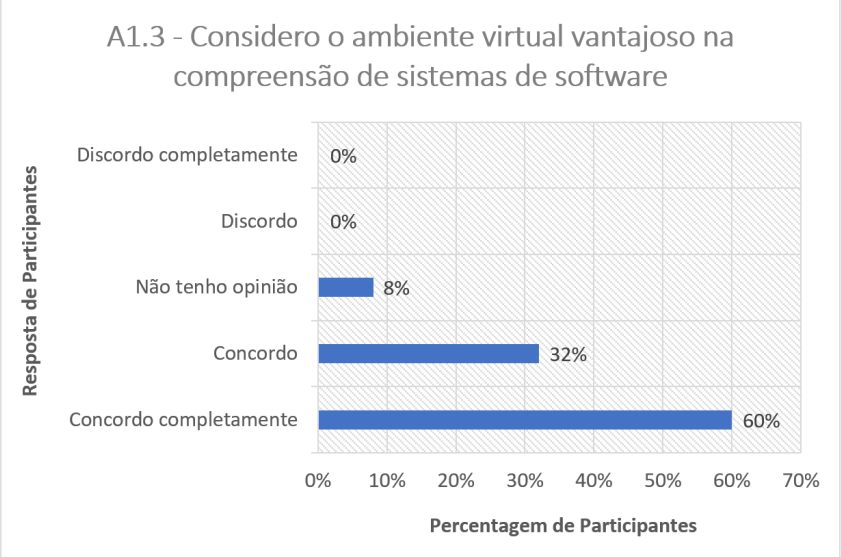
Finalizando este conjunto de questões, os participantes indicaram com base na sua opinião se ambiente virtual que experienciaram era vantajoso na compreensão de sistemas de software, Figura 5.15. Os resultados revelam-se encorajadores ao objetivo do ambiente virtual, pela grande presença de respostas positivas em confronto com nenhuma resposta negativa.

Em suma, considerando a avaliação geral ao ambiente virtual, os participantes encontraram vantagens na utilização da ferramenta.

System Usability Scale

Por fim, os participantes responderam à escala de usabilidade do sistema, que consiste em 10 perguntas com o objetivo de apontarem aspetos que merecem atenção relativamente à usabilidade.

A Figura 5.16 mostra a percentagem de respostas dadas para cada uma das 10 questões. As cores utilizadas na tabela têm o objetivo de reduzir o esforço na compreensão da mesma, sendo que valores próximos de 0% recebem a cor vermelha e, em oposição, valores próximos de 100% recebem a cor verde.



A1.3	1º Quartil	Mediana	3º Quartil
	4	5	5

Figura 5.15: Avaliação geral ao ambiente virtual.

SUS	Gostaria de usar este sistema com frequência	Acho o sistema demasiado complexo	Achei o sistema fácil de usar	Acho que é necessário a ajuda de uma pessoa com conhecimentos técnicos para usar o sistema	Acho que as várias funções do sistema estão bem integradas	Acho que o sistema apresenta muita inconsistência	Acho que as pessoas aprenderiam rapidamente a usar este sistema	Achei o sistema muito complicado de usar	Senti-me confiante a usar o sistema	Precisava de aprender muitos conhecimentos antes de poder usar este sistema
Concordo completamente	40%	0%	52%	0%	56%	0%	44%	0%	44%	0%
Concordo	48%	4%	44%	28%	32%	0%	52%	0%	48%	8%
Não tenho opinião	12%	4%	4%	16%	12%	4%	0%	8%	8%	16%
Discordo	0%	36%	0%	28%	0%	20%	4%	16%	0%	24%
Discordo completamente	0%	56%	0%	28%	0%	76%	0%	76%	0%	52%

Figura 5.16: Avaliação geral ao ambiente virtual.

Após analise dos resultados e seguindo o cálculo do *score* do *System Usability Scale* foi atribuída uma classificação de 83.8, que se traduz numa nota A, ou seja, a nota máxima possível. Os participantes mostram assim interesse pela usabilidade da ferramenta testada.

Desafios

A partir da análise dos dados apresentados e do *feedback* recebido dos participantes, o maior desafio encontrado por estes foi a reduzida experiência na utilização de *headsets* de RV. Esta situação levou a aumentos no tempo de aprendizagem e maior dificuldade nas interações devido

ao prematuro conhecimento dos controladores.

A Figura 5.17 realça que a experiência de utilização do Oculus Rift favoreceu a T1, reduzindo o tempo necessário à sua realização, ou seja, o tempo de exposição necessário para explorar o ambiente virtual foi mais reduzido. O coeficiente de determinação obteve um valor de 0.6339, aproximadamente.

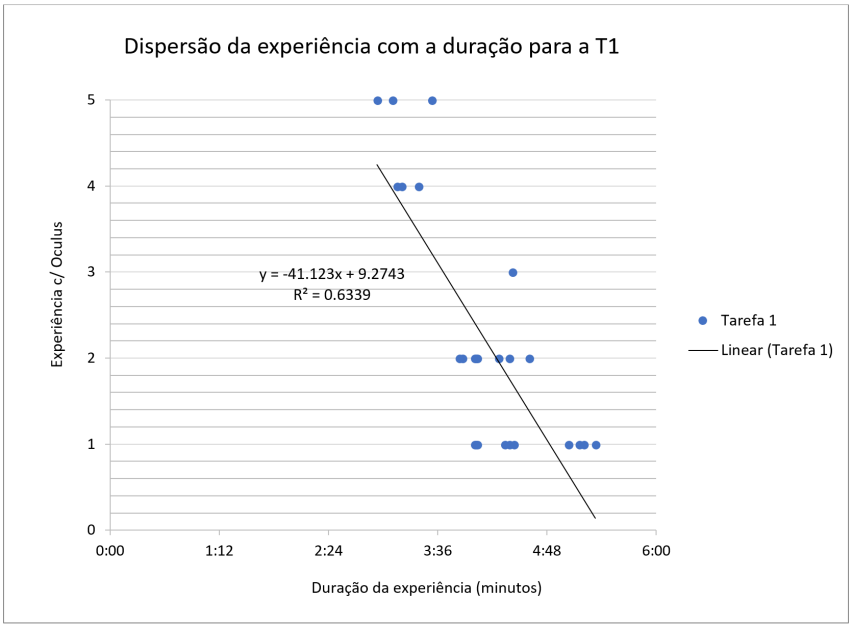


Figura 5.17: Dispersão da experiência com Oculus e a duração para a T1.

O agente controlador da experiência notou evolução na capacidade de interação com o sistema ao longo do tempo. Certamente, numa segunda participação em testes, os participantes sentirão mais confiança na utilização do dispositivo de RV e assumirão melhores resultados. Concluindo, este desafio pode ser ultrapassando se o participante obtiver maior experiência com o Oculus.

Uma análise semelhante foi realizada para as tarefas T2 e T3 opondo a duração da experiência à experiência de utilização dos Oculus, mas não surgiram resultados significativos. Ou seja, ter maior ou menor experiência com o dispositivo de RV não traz benefícios ao nível do tempo necessário para a resolução do problema. Em T2, talvez seja um fator mais significativo comparar a prática de programação do participante com a duração. No entanto, como todos são programadores, em particular 88% dos participantes afirmaram que programam praticamente de forma diária, é considerado que não existe grande discrepância de conhecimentos que criasse uma diferenciação para a resolução desta tarefa. Os dados comprovam isso mesmo, não havendo qualquer resultado significativo.

5.6 Sumário

Este capítulo detalhou a avaliação realizada ao ambiente virtual utilizando o Maze como caso de estudo numa experiência controlada. Os resultados detalham as tarefas realizadas por 25 participantes e as suas conclusões. Todas as tarefas foram concluídas com sucesso e nenhum participante desistiu.

O conjunto de resultados obtidos, em praticamente todos os pontos analisados, revelam-se positivos pela grande presença de respostas favoráveis à utilização do ambiente virtual. A partir destes dados é possível mostrar que um ambiente virtual reduz o esforço necessário nas tarefas propostas e que é um ponto de partida para o projeto em desenvolvimento de *live software development*.

Assim, a contribuição proposta pelo ambiente virtual pode ser considerada positiva.

Capítulo 6

Conclusões e Trabalho Futuro

Este capítulo conclui a dissertação apresentando um sumário de todo o documento e um conjunto de recomendações para trabalho futuro.

6.1 Conclusões

Um ambiente virtual que consegue instanciar um sistema de software de forma *live* e inseri-lo num contexto de realidade virtual, conduz a uma profunda imersão no essencial de um sistema em execução, na sua análise estática e dinâmica. A aplicação de metáforas aperfeiçoa e familiariza esse conteúdo, criando um ambiente rico e tangível.

O programador pode entrar num sistema e observá-lo como uma cidade, onde os pacotes são distritos, as classes são edifícios e as invocações a métodos são conexões. A informação recolhida para a compreensão do sistema não se limita apenas à visualização, mas também à interação.

A investigação em *live software development* procura reduzir o esforço necessário para compreender e evoluir sistemas de software. Nesse sentido, como ponto de partida, foi realizada uma revisão bibliográfica às principais áreas de interesse desta dissertação que permitiram recolher vários contributos científicos e que foram consecutivamente explorados, como analisado no Capítulo 2.

Após definir o objetivo e encontrar bases sólidas de vários investigadores na revisão bibliográfica, contextualizou-se o problema e apresentou-se, com maior fundamentação, o problema e respetivas hipóteses. Adicionalmente, a arquitetura foi realçada com a justificação de algumas decisões estruturais. A aplicação de casos de estudo de várias dimensões e complexidades justifica a pertinência deste projeto, como apresenta o Capítulo 3.

Com o sucesso da implementação do âmbito proposto, foi apresentado o processo de criação do ambiente virtual. Ambiente esse que comunica, visualiza e permite a compreensão de sistemas de software por quem o utiliza. A interface de utilização é adicional, mas necessária para as primeiras exposições ao ambiente, como expõe o Capítulo 4.

Os resultados mostram que o ambiente virtual teve sucesso naquilo que se comprometeu e que, de facto, apresenta vantagens em termos de compreensão de código. A adaptação ao ambiente é considerada a necessária comparada com qualquer outra ferramenta usada pela primeira vez. Desta forma, com sucessivas utilizações e aumento da experiência na utilização do *headset* de realidade virtual, o programador conseguirá fazer uso da totalidade das funcionalidades implementadas, como avalia o Capítulo 5.

Neste sentido, a aplicação de *liveness* foi conseguida ao seu nível informativo, significativo, responsivo e *live*. O utilizador recebe o *feedback* do sistema de software imediatamente. A contribuição desta investigação responde, na integra, aos objetivos levantados.

Como responsável pelo ambiente virtual surge o motor que move os dados da análise do sistema até à instanciação do mundo virtual, permitindo uma redução do esforço associado à compreensão e à manutenção.

Assim, é possível concluir que o motor de ambiente virtual é vantajoso e que constitui um ponto de partida para o desenvolvimento e continuação da investigação em *live software development*.

6.2 Trabalho Futuro

Como qualquer projeto de software implementado, o motor de ambiente virtual pode e deve passar pelo processo de manutenção, seja ele evolutivo ou corretivo.

Formatos dos Edifícios

Os edifícios instanciados no ambiente virtual assumem todos um formato paralelepípedo, correspondendo ao único *prefab*, criado para o efeito, que o representa. Um *prefab* é considerado um *template* que pode guardar componentes e propriedades que são imediatamente atribuídas aos objetos que o usarem.

No entanto, podem ser implementados vários *prefabs* com vários formatos, como cilindros, pirâmides ou até combinações de vários, e dar oportunidade ao utilizar de definir ao seu agrado o aspeto do edifício através do menu. Outra alternativa seria o próprio motor usar o formato do edifício como uma métrica do software e ele próprio dividir.

A Figura 6.1 exemplifica a diversidade de formatos que poderão existir apenas com a alteração do *prefab*. Caso seja mantida semelhanças com uma cidade, a metáfora até agora não será perdida, pelo contrário, será apenas mais uma atualização.

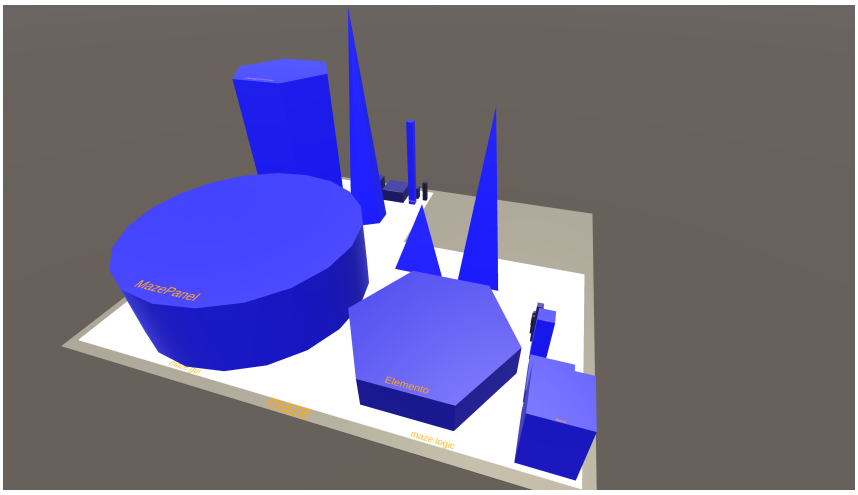


Figura 6.1: Diversidade de formatos 3D para os edifícios.

Cores dos Objetos

As cores são uma métrica interessante e ainda pouco explorada. Uma palete de cores poderá criar um conjunto de soluções que enriquecerão a visualização sem a necessidade de qualquer interação. A Figura 6.2 aprofunda um exemplo colorido para este contexto.

No entanto, ao utilizar cores em abundância podem ser levantadas dificuldades ao nível do daltonismo. Felizmente, várias soluções foram propostas para resolver este tipo de desafios, como a aplicação de sistemas de identificação de cores, por exemplo ColorAdd[1]. Cada edifício poderá albergar um símbolo, representativo da cor, no seu material como textura.

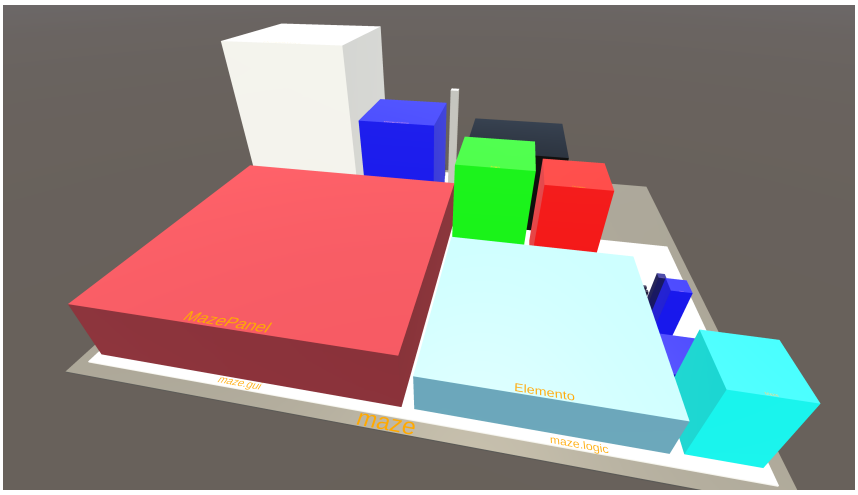


Figura 6.2: Diversidade de cores para os objetos.

[1]ColoAdd, http://www.coloradd.net/, Último Acesso em: 20/06/2018

Melhorias ao conceito de cidade

A metáfora visual da cidade permite criar um ambiente mais familiar e atrativo aos utilizadores, no entanto a implementação usada apresenta-se de uma forma simples.

Revela-se interessante apostar em novos conteúdos, sejam eles métricas para o sistema de software em análise ou apenas decorativos, sem que se tornem distrativos.

Sistema sonoro para situações de oclusão

Numa cidade complexa e com vários elementos ou edifícios com uma grande variedade de tamanhos é muito provável que aconteçam oclusões. Como tal, pode estar a acontecer alguma coisa relevante para o utilizar, mas este não consegue ver.

O ambiente virtual já permite que um edifício imita um som e que este se propague no espaço com a respetiva perda de sinal. Como tal, esta funcionalidade pode servir de princípio para uma aplicação a questões de oclusão ou a situações em que seja importante notificar o utilizador que algum pormenor relevante está a acontecer.

Anotações 3D que surgem no *headset* também poderão ser uma aposta para esta questão.

Explosion view

Conseguir tirar partido de um espaço sem qualquer utilidade e acrescentar-lhe mais elementos é uma tarefa para a *explosion view*.

Se ao selecionar um edifício, este abrisse e mostrasse o seu conteúdo seria uma forma de rentabilizar o seu espaço interior. O conteúdo a mostrar poderia ser exatamente aquilo que compõe a classe, ou seja, os métodos e os atributos, com uma representação 3D.

Seguindo o contexto de uma cidade, a utilização de andares num edifício para representar métodos de uma classe talvez seja um fator interessante.

É sugerido que a instanciação destas estruturas ocorram apenas quando é despoletado um evento, como um *click*, evitando uma instanciação de todos os objetos no início da execução, minimizando assim o esforço do processamento dos objetos.

A vista explosiva permitiria uma análise mais pormenorizada aos componentes, tornando-os em algum objeto virtual e não ser apenas texto.

Comunicação bidirecional com o repositório

A comunicação com o repositório que disponibiliza ao motor toda a informação estática e dinâmica do sistema de software é realizada de forma unidirecional, desde o servidor até ao motor do ambiente virtual.

No entanto, a comunicação no sentido oposto deve ser implementada num esforço partilhado com o repositório que, por sua vez, ainda não consegue receber dados do motor.

O simples exemplo de ser o próprio utilizador no ambiente virtual a modificar algum pormenor e salvar essa alteração no repositório seria uma mais valia. Ou no momento em que o ambiente virtual faz pausa à sua visualização, essa pausa ser repercutida até ao IDE e pausar o próprio IDE.

Em suma, é possível afirmar que foi aberto um caminho para o *live software development* através do desenvolvimento do ambiente virtual e do seu respetivo motor. No entanto, o âmbito deste projeto deve ser expandido e adicionadas novas funcionalidades, de forma a que nunca cesse a motivação de procurar aquilo que à partida parece impossível, neste caso, reduzir o esforço necessário à compreensão de sistemas de software.

Referências

[AD07a] S. Alam e P. Dugerdil. Evospaces: 3d visualization of software architecture. In *19th International Conference on Software Engineering and Knowledge Engineering*, volume 7, pages 500–505. IEEE, 2007.

[AD07b] S. Alam e P. Dugerdil. Evospaces visualization tool: Exploring software architecture in 3d. In *14th Working Conference on Reverse Engineering (WCRE 2007)*, pages 269–270, Oct 2007. doi:10.1109/WCRE.2007.26.

[Agu03] Ademar Aguiar. *Framework Documentation: A Minimalist Approach*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, Sep 2003.

[Bae98] Ronald Baecker. Sorting out sorting: A case study of software visualization for teaching computer science. *Software visualization: Programming as a Multimedia Experience*, 1:369–381, 1998.

[BAW98] M. M. Burnett, J. W. Atwood e Z. T. Welch. Implementing level 4 liveness in declarative visual programming languages. In *Proceedings. 1998 IEEE Symposium on Visual Languages (Cat. No.98TB100254)*, pages 126–133, Sep 1998. doi:10.1109/VL.1998.706155.

[BFdH+13] Sebastian Burckhardt, Manuel Fahndrich, Peli de Halleux, Sean McDirmid, Michal Moskal, Nikolai Tillmann e Jun Kato. It's alive! continuous feedback in ui programming. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 95–104, New York, NY, USA, 2013. ACM. URL: http://doi.acm.org/10.1145/2491956.2462170.

[BK01] Sarita Bassil e Rudolf K. Keller. Software visualization tools: Survey and analysis. *Proceedings of the 9th International Workshop on Program Comprehension*, page 7, May 2001.

[Bra04] Joel R. Brandt. Run-time modification of the class hierachy in a live java development environment. Technical Report WUCSE-2004-71, Washington University in St. Louis, May 2004.

[BWDL08] Andrea Biaggi, Richard Wettel, Prof Dr e Michele Lanza. A 3d visualization plug-in for eclipse, 2008.

[CIL86] Shi-Kuo Chang, Tadao Ichikawa e Panos A. Ligomenides. *Visual Languages*. Springer US, First edition, 1986. doi:10.1007/978-1-4613-1805-7.

[CLR96] Virginio Cantoni, Stefano Levialdi e Vito Roberto. *Artificial Vision: Image Description, Recognition and Communication*. Academic Press, First edition, 1996. URL: https://doi.org/10.1016/B978-0-12-444816-2.X5000-X.

[CNdSR16] Cristiano Carvalheiro, Rui Nóbrega, Hugo da Silva e Rui Rodrigues. User redirection and direct haptics in virtual environments. In *Proceedings of the 2016 ACM on Multimedia Conference*, MM '16, pages 1146–1155, New York, NY, USA, 2016. ACM. URL: http://doi.acm.org/10.1145/2964284.2964293.

[Cza13] Evan Czaplicki. Interactive programming, Sep 2013. URL: http://elm-lang.org/blog/interactive-programming.

[Die07] Stephan Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. doi:10.1007/978-3-540-46505-8.

[DN14] P. Dugerdil e M. Niculescu. Visualizing software structure understandability. In *2014 23rd Australian Software Engineering Conference*, pages 110–119, Apr 2014. doi:10.1109/ASWEC.2014.17.

[Dom18] Gil Domingues. A software repository for live software development. Master's thesis, University of Porto, Porto, 2018.

[ECW16] Jonathan Edwards, Jodie Chen e Alessandro Warth. Live end-user programming. LIVE 2016, 2016.

[EPP15] Anthony Elliott, Brian Peiris e Chris Parnin. Virtual reality in software engineering: Affordances, applications, and challenges. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, pages 547–550, Piscataway, NJ, USA, 2015. IEEE Press. URL: http://dl.acm.org/citation.cfm?id=2819009.2819098.

[ESW17] Martin Erwig, Karl Smeltzer e Xiangyu Wang. What is a visual language? *Journal of Visual Languages & Computing*, 38:9–17, 2017. URL: http://linkinghub.elsevier.com/retrieve/pii/S1045926X15300264, doi:10.1016/j.jvlc.2016.10.005.

[FKH15] F. Fittkau, A. Krause e W. Hasselbring. Exploring software cities in virtual reality. In *2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT)*, pages 130–134, Set 2015. doi:10.1109/VISSOFT.2015.7332423.

[FKH17] Florian Fittkau, Alexander Krause e Wilhelm Hasselbring. Software landscape and application visualization for system comprehension with explorviz. *Information and Software Technology*, 87:259 – 277, 2017. URL: http://www.sciencedirect.com/science/article/pii/S0950584916301185.

[FZ17] Florian Fittkau e Christian Zirkelbach. Explorviz. https://www.explorviz.net, 2017. Acedido em: 30-01-2018.

[Ger94] Nahum Gershon. From perception to visualization. *Scientific Visualization*, pages 129–139, 1994. URL: https://ci.nii.ac.jp/naid/10020543894/en/.

[Gol03] Kenneth J. Goldman. A demonstration of jpie: An environment for live software construction in java. *In the Conference Companion, 18th Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 403–414, 2003.

[Gol04a] Kenneth J. Goldman. An interactive environment for beginning java programmers. *Science of Computer Programming*, 53(1):3 – 24, 2004. Practice and experience with Java in education. URL: http://www.sciencedirect.com/science/article/pii/S0167642304000590.

[Gol04b] Kenneth J. Goldman. Live software development with dynamic classes. Technical report, Department of Computer Science and Engineering Washington University in St. Louis, Aug 2004.

[Gro17] Object Management Group. Unified modeling language 2.5.1. www.omg.org/spec/UML/2.5.1/, 2017. Acedido em: 14-01-2018.

[GRR99] Martin Gogolla, Oliver Radfelder e Mark Richters. Towards three-dimensional representation and animation of uml diagrams. In *Proceedings of the 2Nd International Conference on The Unified Modeling Language: Beyond the Standard*, UML'99, pages 489–502, Berlin, Heidelberg, 1999. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=1767297.1767349.

[Hai59] Lois M. Haibt. A program to draw multilevel flow charts. In *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference*, IRE-AIEE-ACM '59 (Western), pages 131–137, New York, NY, USA, 1959. ACM. URL: http://doi.acm.org/10.1145/1457838.1457861.

[Han03] Christopher Michael Hancock. *Real-time Programming and the Big Ideas of Computational Literacy*. PhD thesis, Cambridge, MA, USA, 2003. AAI0805688.

[Hil92] Daniel D. Hils. Visual languages and computing survey: Data flow visual programming languages. *Journal of Visual Languages & Computing*, pages 69–101, 1992.

[Hol06] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, Sep 2006. doi:10.1109/TVCG.2006.147.

[KEV16] Gabriël Konat, Sebastian Erdweg e Eelco Visser. Towards live language development. 2016.

[KF17] Mehmet Kaya e James W Fawcett. Identification of extract method refactoring opportunities through analysis of variable declarations and uses. *International Journal of Software Engineering and Knowledge Engineering*, 27(01):49–69, 2017.

[KM00] C. Knight e M. Munro. Virtual but visible software. In *2000 IEEE Conference on Information Visualization. An International Conference on Computer Visualization and Graphics*, pages 198–205, 2000. doi:10.1109/IV.2000.859756.

[Knu63] Donald E. Knuth. Computer-drawn flowcharts. *Commun. ACM*, 6(9):555–563, Sep 1963. URL: http://doi.acm.org/10.1145/367593.367620.

[Kos02] Rainer Koschke. *Software Visualization for Reverse Engineering*, pages 138–150. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. URL: https://doi.org/10.1007/3-540-45875-1_11.

[KRB18] Juraj Kubelka, Romain Robbes e Alexandre Bergel. The road to live programming: Insights from the practice. In *Proceedings of the 40th International Conference on*

Software Engineering, ICSE '18, pages 1090–1101, New York, NY, USA, 2018. ACM. URL: http://doi.acm.org/10.1145/3180155.3180200.

[LC07] Christian F. J. Lange e Michel R. V. Chaudron. Interactive views to improve the comprehension of uml models - an experimental validation. In *15th IEEE International Conference on Program Comprehension (ICPC '07)*, pages 221–230, Jun 2007. doi:10.1109/ICPC.2007.23.

[LGD09] M. Lanza, H. Gall e P. Dugerdil. Evospaces: Multi-dimensional navigation spaces for software evolution. In *2009 13th European Conference on Software Maintenance and Reengineering*, pages 293–296, Mar 2009. doi:10.1109/CSMR.2009.14.

[LL13] R. Lemma e M. Lanza. Co-evolution as the key for live programming. In *2013 1st International Workshop on Live Programming (LIVE)*, pages 9–10, May 2013. doi:10.1109/LIVE.2013.6617340.

[McD07] Sean McDirmid. Living it up with a live programming language. In *Proceedings of the 22Nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications*, OOPSLA '07, pages 623–638, New York, NY, USA, 2007. ACM. URL: http://doi.acm.org/10.1145/1297027.1297073.

[McD13] Sean McDirmid. Usable live programming. In *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, Onward! 2013, pages 53–62, New York, NY, USA, 2013. ACM. URL: http://doi.acm.org/10.1145/2509578.2509585.

[McD16] Sean McDirmid. The promise of live programming. LIVE 2016, 2016.

[McD17] Sean McDirmid. Live programming as gradual abstraction. LIVE 2017, 2017.

[MCS05] A. Marcus, D. Comorski e A. Sergeyev. Supporting the evolution of a software visualization tool through usability studies. In *13th International Workshop on Program Comprehension (IWPC'05)*, pages 307–316, May 2005. doi:10.1109/WPC.2005.34.

[MGAN17] L. Merino, M. Ghafari, C. Anslow e O. Nierstrasz. Cityvr: Gameful software visualization. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 633–637, Sep 2017. doi:10.1109/ICSME.2017.70.

[MLM01] Jonathan I. Maletic, Jason Leigh e Andrian Marcus. Visualizing software in an immersive virtual reality environment. In *in Proceedings of ICSE'01 Workshop on Software Visualization*, pages 12–13. Society Press, 2001.

[MTRK14] Walid Maalej, Rebecca Tiarks, Tobias Roehm e Rainer Koschke. On the comprehension of program comprehension. *ACM Trans. Softw. Eng. Methodol.*, 23(4):31:1–31:37, Sep 2014. URL: http://doi.acm.org/10.1145/2622669.

[MTUK95] Paul Milgram, Haruo Takemura, Akira Utsumi e Fumio Kishino. Augmented reality: A class of displays on the reality-virtuality continuum. In *Telemanipulator and telepresence technologies*, volume 2351, pages 282–293. International Society for Optics and Photonics, 1995.

[Mye86] B. A. Myers. Visual programming, programming by example, and program visualization: A taxonomy. In *Proceedings of the SIGCHI Conference on Human Factors*

in Computing Systems, CHI '86, pages 59–66, New York, NY, USA, 1986. ACM. URL: http://doi.acm.org/10.1145/22627.22349.

[NS10] Kamruddin Md Nur e Hasan Sarwar. Software visualization tools for software comprehension. *SKIMA 2010*, pages 185 – 191, 2010.

[PBG03] T. Panas, R. Berrigan e J. Grundy. A 3d metaphor for software production visualization. In *Proceedings on Seventh International Conference on Information Visualization, 2003. IV 2003.*, pages 314–319, Jul 2003. doi:10.1109/IV.2003.1217996.

[PBS93] Blaine A. Price, Ronald M. Baecker e Ian S. Small. A principled taxonomy of software visualization. *Journal of Visual Languages and Computing*, 4(3):211 – 266, 1993. URL: http://www.sciencedirect.com/science/article/pii/S1045926X83710153.

[PDB17] Cornelius Preidel, Simon Daum e André Borrmann. Data retrieval from building information models based on visual programming. *Visualization in Engineering*, 5(1):18, Oct 2017. URL: https://doi.org/10.1186/s40327-017-0055-0.

[PdQ06] Marian Petre e Ed de Quincey. A gentle overview of software visualisation. *Psychology of Programming Interest Group (PPIG) Newsletter*, Setembro 2006. URL: http://gala.gre.ac.uk/5564/.

[Pei17] Brian Peiris. Github: Riftsketch, 2017. URL: https://github.com/brianpeiris/RiftSketch.

[Pet95] Marian Petre. Why looking isn't always seeing: Readership skills and graphical programming. *Commun. ACM*, 38(6):33–44, June 1995. URL: http://doi.acm.org/10.1145/203241.203251.

[Pla04] Catherine Plaisant. The challenge of information visualization evaluation. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, pages 109–116, New York, NY, USA, 2004. ACM. URL: http://doi.acm.org/10.1145/989863.989880.

[PPDSF12] F. Petrillo, M. Pimenta e C. Dal Sasso Freitas. O estado-da-arte das ferramentas de visualizaçãoo de software. *15th Ibero-American Conference on Software Engineering, CIbSE 2012*, 2012. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84886681175&partnerID=40&md5=29d0b66e70e98975421c3dad8b06b4f3.

[Rei95] Steven P. Reiss. An engine for the 3d visualization of program information. *Journal of Visual Languages & Computing*, 6(3):299 – 323, 1995. URL: http://www.sciencedirect.com/science/article/pii/S1045926X85710178.

[RG00] Oliver Radfelder e Martin Gogolla. On better understanding uml diagrams through interactive three-dimensional visualization and animation. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '00, pages 292–295, New York, NY, USA, 2000. ACM. URL: http://doi.acm.org/10.1145/345513.345358.

[SBF14] IEEE Computer Society, Pierre Bourque e Richard E. Fairley. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. IEEE Computer Society Press, Los Alamitos, CA, USA, 3rd edition, 2014.

[SF16] Christopher Schuster e Cormac Flanagan. Live programming by example: Using direct manipulation for live program synthesis. LIVE 2016, 2016.

[SOT08] Mariam Sensalire, Patrick Ogao e Alexandru Telea. Classifying desirable features of software visualization tools for corrective maintenance. In *Proceedings of the 4th ACM Symposium on Software Visualization*, SoftVis '08, pages 87–90, New York, NY, USA, 2008. ACM. URL: http://doi.acm.org/10.1145/1409720.1409734.

[SOT09] M. Sensalire, P. Ogao e A. Telea. Evaluation of software visualization tools: Lessons learned. In *2009 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 19–26, Sep 2009. doi:10.1109/VISSOF.2009.5336431.

[SS17] N. Singh e S. Singh. Virtual reality: A brief survey. In *2017 International Conference on Information Communication and Embedded Systems (ICICES)*, pages 1–6, Feb 2017. doi:10.1109/ICICES.2017.8070720.

[Tan90] Steven L. Tanimoto. Viva: A visual language for image processing. *Journal of Visual Languages and Computing*, page 12, Feb 1990.

[Tan03] S. L. Tanimoto. Programming in a data factory. In *IEEE Symposium on Human Centric Computing Languages and Environments, 2003. Proceedings. 2003*, pages 100–107, Oct 2003. doi:10.1109/HCC.2003.1260209.

[Tan13] Steven L. Tanimoto. A perspective on the evolution of live programming. *Proceedings of the 1st International Workshop on Live Programming*, page 4, May 2013.

[TC09] A. R. Teyseyre e M. R. Campo. An overview of 3d software visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(1):87–105, Jan 2009. doi:10.1109/TVCG.2008.86.

[VBAM09a] Alex Villazón, Walter Binder, Danilo Ansaloni e Philippe Moret. Advanced runtime adaptation for java. *SIGPLAN Not.*, 45(2):85–94, Oct 2009. URL: http://doi.acm.org/10.1145/1837852.1621621.

[VBAM09b] Alex Villazón, Walter Binder, Danilo Ansaloni e Philippe Moret. Advanced runtime adaptation for java. In *Proceedings of the Eighth International Conference on Generative Programming and Component Engineering*, GPCE '09, pages 85–94, New York, NY, USA, 2009. ACM. URL: http://doi.acm.org/10.1145/1621607.1621621.

[VNP17a] J. Vincur, P. Navrat e I. Polasek. Demonstração de utilização da ferramenta. https://goo.gl/inrcZs, 2017. Acedido em: 02-01-2018.

[VNP17b] J. Vincur, P. Navrat e I. Polasek. Vr city: Software analysis in virtual reality environment. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 509–516, Jul 2017. doi:10.1109/QRS-C.2017.88.

[Wet10] Richard Wettel. *Software Systems as Cities*. PhD thesis, Faculty of Informatics of the Università della Svizzera Italiana, Sep 2010.

[WL07] R. Wettel e M. Lanza. Visualizing software systems as cities. In *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 92–99, Jun 2007. doi:10.1109/VISSOF.2007.4290706.

[WLR11] R. Wettel, M. Lanza e R. Robbes. Software systems as cities: a controlled experiment. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 551–560, May 2011. doi:10.1145/1985793.1985868.

[Woh07] Claes Wohlin. Empirical software engineering: Teaching methods and conducting studies. In *Proceedings of the 2006 International Conference on Empirical Software Engineering Issues: Critical Assessment and Future Directions*, pages 135–142, Berlin, Heidelberg, 2007. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=1767399.1767459.

[WWF+13] J. Waller, C. Wulf, F. Fittkau, P. Döhring e W. Hasselbring. Synchrovis: 3d visualization of monitoring traces in the city metaphor for analyzing concurrency. In *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*, pages 1–4, Sep 2013. doi:10.1109/VISSOFT.2013.6650520.

[YM98] P. Young e M. Munro. Visualising software in virtual reality. In *Program Comprehension, 1998. IWPC '98. Proceedings., 6th International Workshop on*, pages 19–26, Jun 1998. doi:10.1109/WPC.1998.693276.

REFERÊNCIAS

Anexo A

Declaração de Consentimento de Participação na Experiência

DECLARAÇÃO DE CONSENTIMENTO

(Baseada na declaração de Helsínquia)

No âmbito da realização da tese de Mestre no Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto, intitulada **Towards a Live Software Development Environment**, realizada pelo estudante **Diogo da Silva Amaral**, orientada pelo Prof. Ademar Aguiar e sob a coorientação do Prof. Rui Nóbrega, eu abaixo assinado, _____, declaro que compreendi a explicação que me foi fornecida acerca do estudo que irei participar, nomeadamente o carácter voluntário dessa participação, tendo-me sido dada a oportunidade de fazer as perguntas que julguei necessárias.

Tomei conhecimento de que a informação ou explicação que me foi prestada versou os objetivos, os métodos, o eventual desconforto e a ausência de riscos para a minha saúde, e que será assegurada a máxima confidencialidade dos dados.

Explicaram-me, ainda, que poderei abandonar o estudo em qualquer momento, sem que daí advenham quaisquer desvantagens.

Por isso, consinto participar no estudo e na recolha de imagens necessárias, respondendo a todas as questões propostas.

Porto, __ de _____ de _____

_____

(Participante ou seu representante)

Anexo B

Questionário Aplicado à Experiência Controlada

Towards a Live Software Development Environment

*Obrigatório

1. **ID** *

Caracterização do participante

2. **Género** *

Marcar apenas uma oval.

- ◯ Feminino
- ◯ Masculino
- ◯ Outra: _____

3. **Idade** *

4. **Habilitações Literárias (grau concluído)** *

Marcar apenas uma oval.

- ◯ < 12 º ano
- ◯ 12 º ano
- ◯ Licenciatura
- ◯ Mestrado
- ◯ Doutoramento
- ◯ Outra: _____

Conhecimentos e prática tecnológica

5. **Prática em Engenharia de Software** *

Marcar apenas uma oval.

	1	2	3	4	5	
Nunca programo	◯	◯	◯	◯	◯	Programo diariamente

6. **Em média, sinto dificuldade na compreensão de sistemas de software (código/funcionamento de um programa)** *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

7. **Uso com frequência ferramentas de auxílio na compreensão de sistemas de software (ex. ferramentas de debug)**
   Marcar apenas uma oval.

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

8. **Já utilizei alguma ferramenta de visualização para auxiliar na compreensão de um sistema de software. (Se sim, diga qual)** *
   Marcar apenas uma oval.

   ◯ Não
   ◯ Outra: _____

Tarefa 0
Realização da tarefa

Avaliação à Tarefa 0

9. **A tarefa foi de fácil realização.** *
   Marcar apenas uma oval.

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

10. **Atividades** *
    Marcar apenas uma oval por linha.

	1 - Muito difícil	2	3	4	5 - Muito fácil
Identificar classes	◯	◯	◯	◯	◯
Identificar pacotes	◯	◯	◯	◯	◯
Visualizar declarações	◯	◯	◯	◯	◯
Identificar invocações de métodos	◯	◯	◯	◯	◯
Iniciar/Pausar/Continuar execução	◯	◯	◯	◯	◯
Movimentação no sistema	◯	◯	◯	◯	◯

Tarefa 1
Realização da tarefa

Avaliação à Tarefa 1

11. **Senti dificuldade na identificação do ciclo infinito.** *
    Marcar apenas uma oval.

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

12. **No meu IDE corrente teria mais dificuldade em encontrar o sistema bloqueado no ciclo.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

13. **A visualização ajudou a encontrar o problema.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

Tarefa 2
Realização da tarefa

Avaliação à Tarefa 2

14. **Senti dificuldade na identificação da invocação com null exception.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

15. **A visualização ajudou-me a encontrar o problema.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

Avaliação da ferramenta testada

16. **Experiência de utilização do ambiente virtual.** *
Marcar apenas uma oval por linha.

	1 - Discordo	2	3	4	5 - Concordo
Fácil perceção do cenário estático	◯	◯	◯	◯	◯
Fácil perceção do cenário da execução	◯	◯	◯	◯	◯
Fácil compreensão das cores	◯	◯	◯	◯	◯
Utilidade dos botões	◯	◯	◯	◯	◯
Interface intuitiva	◯	◯	◯	◯	◯
Facilidade de utilização	◯	◯	◯	◯	◯

17. **Utilizaria a ferramenta novamente.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

18. **Considero o ambiente virtual vantajoso na compreensão de sistemas de software.** *

Marcar apenas uma oval.

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

19. **Avaliação geral (System Usability Scale)** *

Marcar apenas uma oval por linha.

	1 - Discordo	2	3	4	5 - Concordo
Gostaria de usar este sistema com frequência	◯	◯	◯	◯	◯
Acho o sistema demasiado complexo	◯	◯	◯	◯	◯
Achei o sistema fácil de usar	◯	◯	◯	◯	◯
Acho que é necessário a ajuda de uma pessoa com conhecimentos técnicos para usar o sistema	◯	◯	◯	◯	◯
Acho que as várias funções do sistema estão bem integradas	◯	◯	◯	◯	◯
Acho que o sistema apresenta muita inconsistência	◯	◯	◯	◯	◯
Acho que as pessoas aprenderiam rapidamente a usar este sistema	◯	◯	◯	◯	◯
Achei o sistema muito complicado de usar	◯	◯	◯	◯	◯
Senti-me confiante a usar o sistema	◯	◯	◯	◯	◯
Precisava de aprender muitos conhecimentos antes de poder usar este sistema	◯	◯	◯	◯	◯

20. **Opinião, sugestões ou críticas.**