Revista Brasileira de Informática na Educação, Volume 24, Número 3, 2016

Metodologia de Desenvolvimento de Jogos Sérios: especificação de ferramentas de apoio *open source*

Title: Methodology for Development of Serious Games:: specification of open source tools

Rafaela Vilela da Rocha
Universidade de São Paulo (ICMC - USP)
rafaela.vilela@gmail.com

Aparecida Maria Zem-Lopes
Universidade de São Paulo (ICMC - USP)
Faculdade de Tecnologia de Jahu (FATEC Jahu)
cida.zem@gmail.com

Laís Zagatti Pedro
Universidade de São Paulo (ICMC - USP)
laiszagatti@gmail.com

Ig Ibert Bittencourt
Universidade Federal de Alagoas (IC - UFAL)
ig.ibert@ic.ufal.br

Seiji Isotani
Universidade de São Paulo (ICMC - USP)
sisotani@icmc.usp.br

Resumo *O uso de jogos sérios vem crescendo para fins de aprendizado, treinamento e avaliação do desempenho dos seus usuários. Entretanto, para obter sucesso como produto final, seu desenvolvimento tem que ser sistemático e multidisciplinar. Sendo assim, a escolha das ferramentas de suporte impacta muito além do custo financeiro. Este artigo apresenta uma visão geral da metodologia iterativa e integradora para desenvolvimento de jogos sérios com foco na descrição de ferramentas de código livre. Foram revisadas e selecionadas as ferramentas adequadas, de modo a fornecer um conjunto que poderá ser usado de acordo com as necessidades do projeto do jogo sério. Elas são agrupadas em ferramentas de uso geral, sistemas de gerenciamento de banco de dados, motores de jogos, ferramentas para uso no projeto, e uso na implementação. Além disso, os usos da metodologia e dessas ferramentas são exemplificados com dois cenários distintos.*

Palavras-Chave: *Jogos sérios, Metodologia de desenvolvimento, Ferramentas de código livre.*

Abstract *Serious games are being used ever more for purposes of learning, training and human performance evaluation. However, to be successful as a final product, its development has to be systematic and multi-disciplinary. Thus, the choice of support tools impact beyond the financial cost. This paper describes an overview of the iterative and integrative methodology for developing serious games, expanding the specification with open source tools. Appropriate tools are reviewed and selected in order to provide a set of applications that could be used according to serious games design needs. They are grouped into support tools to general use, database management systems, game engines, design and implementation. In addition, the uses of this methodology and these tools are exemplified with two different scenarios.*

Keywords: *Serious games, Development methodology, Open source tools.*

DOI: 10.5753/RBIE.2016.24.03.109

1 Introdução

Jogos Sérios (JSs) são jogos utilizados com propósito de ensino-aprendizagem ou treinamento de pessoas, e não apenas para diversão [1]. Apesar do seu sucesso de uso em áreas tais como, saúde, defesa, gerenciamento de emergências, negócios, entre outras, seu desenvolvimento é um processo complexo, com alto custo (de recursos humanos, financeiros, tempo e materiais), que requer profissionais qualificados (atores do desenvolvimento) [1-3]. Esses atores devem ter conhecimentos desde sobre aprendizagem, avaliação, simulação e de jogos em si, até do conteúdo no domínio de aplicação e das ferramentas que serão utilizadas no processo de produção do JS [2, 3].

Para oferecer apoio ao desenvolvimento de JSs, diferentes metodologias foram propostas [3-8], além das metodologias específicas existentes nas áreas de simulação [2, 10-13], jogo [14-16], e aprendizagem e treinamento [17-20]. Entretanto, de forma geral, elas não descrevem todo o ciclo de vida, compreendendo os múltiplos requisitos dos jogos sérios: definição dos objetivos de aprendizagem/ treinamento e conteúdo, avaliação do desempenho do aprendiz, fidelidade lógica e física da simulação, não linearidade, com foco no desenvolvimento de conteúdos para engajar o aprendiz, tais como a inclusão de desafios, recompensas, níveis e *feedback* contínuo [8, 21]. As metodologias de *design* de jogos, modelagem da simulação ou *design* instrucional enfatizam apenas os elementos de jogo, ou da simulação, ou da instrução. Já as metodologias para fins de produção de jogos sérios, jogos e simulações educacionais unificam alguns elementos, mas sem abranger todos os requisitos necessários, criando uma falta de balanceamento no produto final criado (por exemplos, um jogo educacional com conteúdo adequado, mas que não gera motivação e engajamento para ser jogado; ou um jogo bastante lúdico, mas que não possibilita a aprendizagem pretendida). Além disto, nem sempre são especificados os artefatos a serem produzidos em cada processo, bem como as ferramentas que podem apoiar o trabalho dos atores envolvidos (há apenas algumas ferramentas de autoria).

Este artigo apresenta ferramentas de código livre que podem ser usadas para produzir os diferentes artefatos durante o desenvolvimento de jogos sérios usando a metodologia iterativa e integradora especificada em [22], ou seja, estende a especificação da metodologia abordando ferramentas de apoio ao desenvolvimento de jogos sérios. Na seção 2, o estado da arte é apresentado. O método de pesquisa é descrito na seção 3. Na seção 4, é apresentada a metodologia de Rocha [22], para fundamentar sua extensão. Na seção 5, são descritas as ferramentas que são agrupadas por tipo de artefatos gerados. Na seção 6, são apresentados os resultados de dois produtos desenvolvidos usando a metodologia. Na Seção 7, os resultados são discutidos, seguidos das considerações finais (Seção 8).

2 Fundamentos e Trabalhos Relacionados

Um jogos sério pode ser um instrumento efetivo de aprendizagem e treinamento ou motivação ao aprendizado [89, 90, 91], além de fornecer apoio para a avaliação realizada pelos instrutores [33]. Entretanto, para que o aprendizado seja efetivo, ele tem que ser ativo, experiencial, contextualizado, baseado em problemas e fornecer *feedback* imediato (princípios de aprendizagem efetiva) [20; 92, 93]. Dessa forma, para projetar JSs efetivos, os conhecimentos de diferentes atores precisam ser integrados, tais como, o conhecimento do domínio de especialistas (professores, instrutores ou profissionais na área), as competências de desenvolvedores (analistas, programadores, etc.) e o conhecimento de *designers* de jogos.

Além disso, as pesquisas em áreas afins, tradicionais e mais desenvolvidas, tais como, Engenharia de *Software*, *Design* Instrucional, *Interface* Homem-Computador, Modelagem e Simulação, Educação e Psicologia, podem beneficiar e produzir melhores soluções para o desenvolvimento e jogos sérios mais efetivos. Para isto, é necessária a integração dessas áreas, tanto quanto em relação ao produto final, pois cada área produz diferentes produtos finais; quanto em relação a metodologia de desenvolvimento (atores, processos, produtos, projetos), pois há diferentes metodologias de desenvolvimento em cada área.

As metodologias existentes na literatura podem ser classificadas em três grupos (conforme Figura 1): (1) Metodologias específicas para jogo, ou simulação, ou aprendizagem/treinamento; (2) Metodologias para desenvolvimento de simulação e jogo educacional; e (3) Metodologias para desenvolvimento de jogos sérios. Elas são apresentadas a seguir.

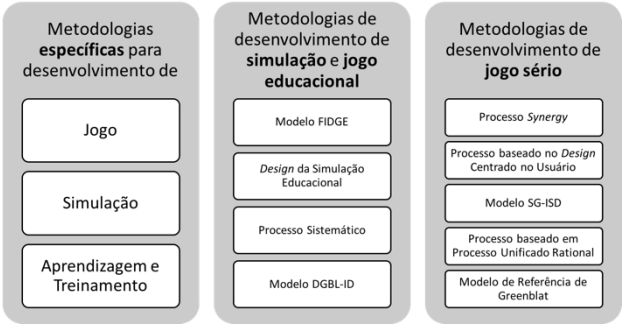

Figura 1: Visão geral dos grupos de metodologias para desenvolvimento de diferentes produtos finais

2.1. Metodologias Específicas

As metodologias específicas são para desenvolvimen-

to de jogos de entretenimento [14-16], simulações [2, 10-13], materiais/objetos educacionais [17-19] e materiais de treinamento [20]. Apesar de serem metodologias utilizadas em cada área, e muitas vezes estendidas para o desenvolvimento de jogos sérios, elas não auxiliam em requisitos particulares dos JSs, pois há diferenças significativas entre eles e jogos de entretenimento, simulações e materiais educacionais. No Quadro 1 são apresentadas as diferenças entre as características de cada produto comparadas às características de jogos sérios.

Jogos de propósitos gerais podem ter uma narrativa abstrata ou irreal, pois eles focam na diversão, exceto os jogos de simulação que geralmente simulam de modo simplificado equipamentos, fenômenos ou ambientes da realidade [25].

Nas simulações, as fidelidades física e comportamental são os graus em que o modelo de simulação: se parece fisicamente (*interface*, áudios e controles); ou age (informações, estímulos e respostas) e reage (comportamento e interatividade) às tarefas sendo executadas, comparadas ao equipamento, ambiente ou fenômeno do mundo real [23]. Entretanto, estudos indicaram que a percepção de verossimilhança é mais importante do que a alta fidelidade [23]. Além de que, alguns componentes de simulação que reduzem o realismo podem aumentar o aprendizado, tais como, parar e reiniciar, ou um mecanismo de *feedback* avançado [24].

Os materiais educacionais têm conteúdos estruturados, lógicos e sequenciais (ou seja, linear), com foco no processo para auxiliar o instrutor [26]. Ao contrário, jogos sérios são não lineares, possibilitando que as interações do jogadores afetem o resultado final do jogo.

Produtos	Objetivos	Representação da realidade	Linearidade do conteúdo	Foco de auxílio
Jogos	Entretenimento	Abstrata ou irreal	Não linear	Jogador
Simulações`	Treinamento	Fidelidade física, comportamental, ou verossimilhança	Não linear	Operador
Materiais educacionais e/ou de treinamentos	Aprendizagem e/ou treinamento e/ou avaliação	Abstrata ou verossimilhança	Linear	Instrutor
Jogos sérios	Aprendizagem, treinamento e avaliação	Fidelidade física, comportamental, ou verossimilhança	Não linear	Aprendiz

Quadro 1: Comparação entre as características de jogos sérios com jogos de entretenimento, simulações, materiais educacionais e de treinamentos

2.2. Metodologias para desenvolvimento de simulação e jogo educacional

Jogos educacionais não simulam comportamentos de sistemas e processos, seu foco está em fornecer e avaliar conhecimentos [8]. Já as simulações educacionais simulam em algum grau a realidade (tais como, simuladores de Física), mas sem incluir elementos de jogos que podem engajar e desafiar os aprendizes.

As metodologias de desenvolvimento de simulação e jogo educacional são: GAMED [3], *Design* da Simulação Educacional [4], Modelo FIDGE [5], Processo Sistemático [8] e Modelo DGBL-ID [27]. Elas não abordam os requisitos de fidelidade lógica e física da simulação, não oferecem apoio para avaliação e *feedback* do desempenho do aprendiz, e não integram jogo-instrução de forma efetiva.

2.3. Metodologias para desenvolvimento de jogos sérios

As metodologias de desenvolvimento de jogos sérios são: Processo *Synergy* [6], Processo baseado no *Design* Centrado no Usuário [7], Modelo SG-ISD [28], Processo baseado em Processo Unificado *Rational* [29] e Modelo de Referência de Greenblat [30]. Elas têm foco no *design* instrucional e do jogo, porém sem integração detalhada, não apoiam a avaliação do JS, e apenas descrevem o que deve ser avaliado sobre o desempenho do aprendiz, sem oferecer suporte. Além disto, algumas abordam apenas algumas fases do desenvolvimento [7, 29].

De forma geral, essas metodologias nem sempre detalham os artefatos e os papéis dos atores que estão envolvidos em cada processo. Da mesma forma, as ferramentas de suporte se limitam a aplicativos de autoria, sem oferecer opções para os usuários.

Neste contexto, é necessária uma nova metodologia de desenvolvimento de jogos sérios que integre, de fato, as áreas de jogo, simulação, aprendizagem, avaliação e domínio de aplicação. Somado a isso, não há metodologia na literatura que especifica o uso de ferramentas *open source* em cada fase de desenvolvimento de jogos sérios. Essas ferramentas podem viabilizar a produção dos artefatos, inclusive gerá-los automaticamente de forma padronizada, o que pode contribuir para aumentar a qualidade do produto final. Além disso, essas ferramentas podem mudar de acordo com as características do jogo sério (por exemplos, plataforma, forma de acesso, número de jogadores, forma de persistência dos dados coleta-

dos durante o jogo) e da equipe que o desenvolve (por exemplos, conhecimentos e habilidades específicas).

3 Método de Pesquisa

Para superar os desafios apresentados, uma nova metodologia iterativa e integradora foi criada por Rocha [22], para apoiar o desenvolvimento de jogos sérios em específico, com a finalidade de treinar e avaliar o desempenho humano. Além da metodologia, descrita na Seção 4, também foram criados artefatos, modelos, componentes e uma arquitetura de suporte que guiam e apoiam cada processo do ciclo de vida, descritos em [22].

A metodologia foi utilizada para criar 20 protótipos de simulações interativas e um jogo sério completo. Durante seu uso, foram identificados, selecionados e usados diferentes tipos de ferramentas, desde as ferramentas comuns ao desenvolvimento de *software* em geral, tais como, editores de texto e sistema de controle de versão, até as mais específicas para a criação do jogo, tais como ferramentas de modelagem 3D e motores de jogos. Dentre as opções que podem ser utilizadas, as principais ferramentas de código livre são apresentadas na Seção 5 (foco deste artigo, conforme Figura 2).

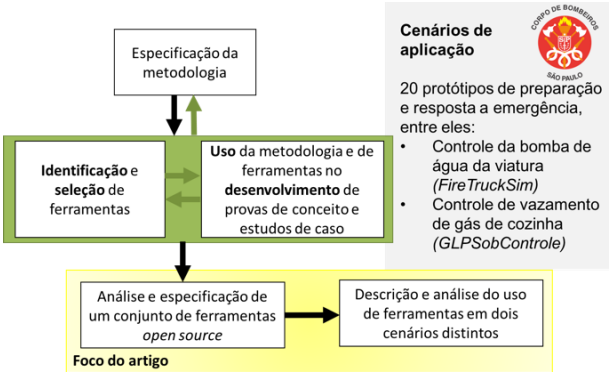

Figura 2: Visão geral do método de pesquisa e foco deste artigo

Dois exemplos de cenários desenvolvidos usando a metodologia e diferentes ferramentas são apresentados e discutidos nas Seções 6 e 7.

4 Metodologia de Desenvolvimento de Jogos Sérios

A metodologia é especificada em Rocha [22], bem como os modelos integradores (programa de treinamento, modelos formais, diagramas) e os critérios para construção de JSs (cenários, fases, descrição dos objetivos, *interface*, avaliações e *feedback*). Em Rocha [22] também são descritos os desafios, teorias e abordagens usadas para produzir um jogo sério (que possui requisitos particulares), atingindo os objetivos de aprendizagem/competência pretendida. Além disso, são especificados os papéis dos atores envolvidos na produção, cada processo e seus artefatos de entrada e saída; bem como, as avaliações e validações de aprendizagem com o JS e do próprio JS. Dessa forma, a descrição apresentada neste artigo aborda apenas os conceitos tecnológicos para fundamentação dos diferentes tipos de ferramentas *open source* que serão apresentados na Seção 5.

A Figura 3 apresenta os processos da metodologia, proposta por Rocha [22], que são descritos a seguir. Ela é dividida em três etapas: Pré-Produção (uma fase de planejamento), Produção (quatro fases que compreendem os processos de Análise, Projeto, Implementação e Integração e Testes), e Pós-Produção (duas fases realizadas depois que o jogo sério foi produzido: Execução e Avaliação). O processo "(1) Planejamento" (na **Pré-produção**) visa a elaboração do plano inicial, a partir das necessidades de treinamento, contendo os procedimentos e as competências que serão treinadas (treinamento) e a situação do mundo real que será simulada (jogo & simulação). Esse plano deve ser feito pelo responsável técnico, que pode ser um profissional ou uma equipe que contém um ou mais pedagogo, professor, treinador e/ou especialista no domínio.

Na etapa de **Produção**, no processo "(2) Análise", os requisitos do jogo sério são especificados, incluindo dados do treinamento e avaliação (definidos pelo responsável técnico); e dados do jogo, simulação, e arquitetura de suporte (definidos pelo analista de sistema). Em "(3) Projeto", os requisitos anteriormente descritos são transformados em projetos de jogo (responsabilidade do *designer*), simulação, arquitetura e banco de dados (responsabilidades do analista de sistema), e programa de treinamento e avaliação (responsável técnico). Em "(4) Implementação", os projetos do processo anterior são implementados pela equipe desenvolvedora, apoiados pelo responsável técnico. Essa equipe é formada por um ou mais programador e *designer* de arte e áudio, ou os respectivos responsáveis pelo desenvolvimento do código-fonte (programação), da arte e do áudio (que podem incluir desde modeladores, animadores, produtores de som, até atores de voz e testadores). Em "(5) Integração e Teste", os artefatos são integrados e testados até a conclusão do jogo sério. Os atores da equipe desenvolvedora realizam suas funções para integrar e testar os recursos desenvolvidos no processo anterior.

Durante a **Pós-produção**, no processo "(6) Execução", os aprendizes realizam o treinamento (simulação e treinamento, sendo que o jogo sério realiza a medição e fornece *feedback*) e, ao final, os aprendizes fazem uma autoavaliação (usando um questionário descrito em Rocha [22]). O instrutor é responsável por acompanhar e auxiliar os aprendizes. Já no processo "(7) Avaliação", o instrutor é responsável por gerar os relatórios de desempenho, realizar sua avaliação e fornecer o *feedback* aos

aprendizes (avaliação do desempenho humano). Além disso, os dados coletados ficam disponíveis para avaliação do programa de treinamento e do jogo sério (avaliação do treinamento).

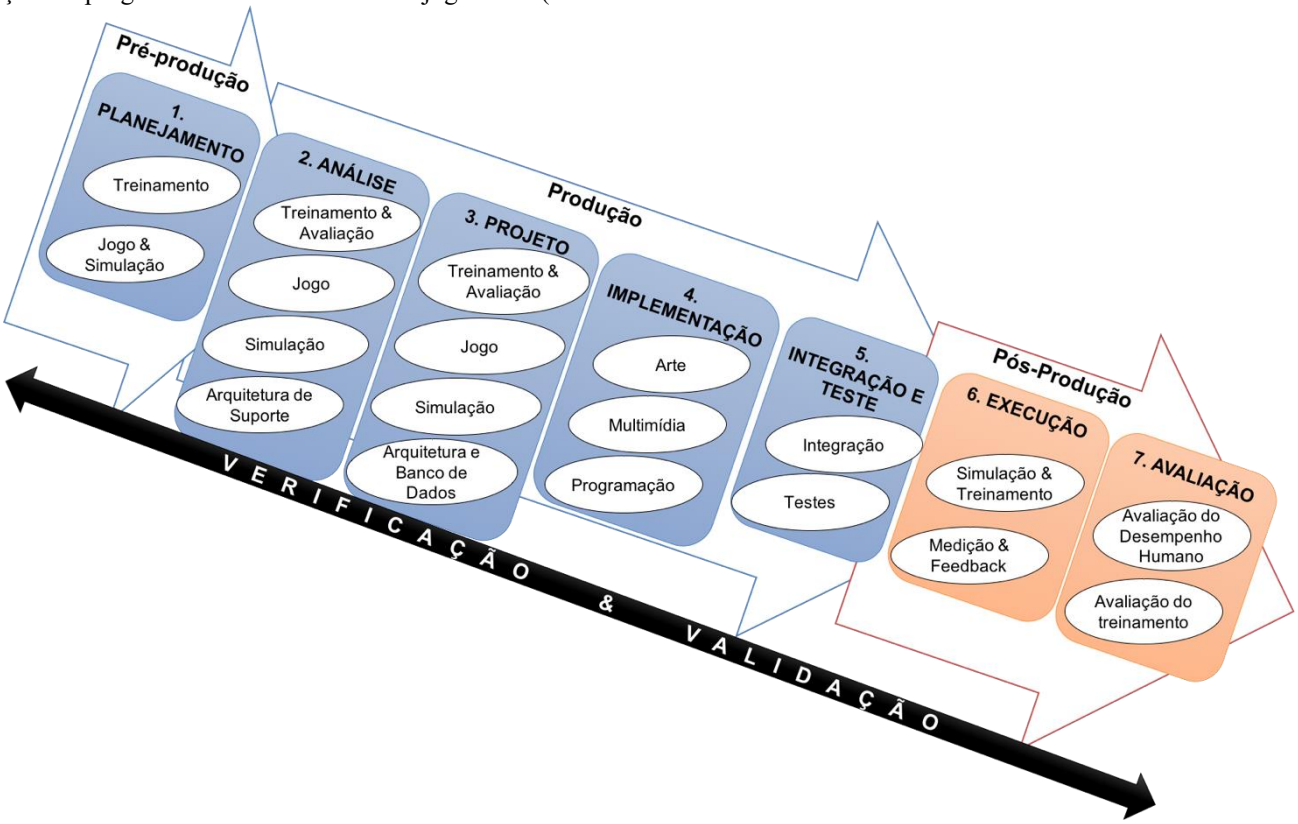

Figura 3: Visão geral dos processos da metodologia de desenvolvimento de jogos sérios (adaptada de [22]) (em azul: desenvolvimento do jogo; em laranja: execução e avaliação do jogo pronto; em preto: verificação e validação que é realizada ao final de cada processo)

Ao final de cada processo, a atividade de **"Verificação e Validação"** deve ser conduzida pela equipe desenvolvedora, em conjunto com o responsável técnico. Essa atividade serve de pontos de controle para a qualidade dos produtos gerados, e caso seja necessário, para a melhoria ou correção dos artefatos produzidos, o desenvolvimento pode voltar em algum processo anterior, de uma forma iterativa e incremental.

A metodologia criada foi avaliada por especialistas em Engenharia de *Software* e desenvolvimento de JSs [22]. Já o jogo sério completo, nomeado *GLPSobcontrole* foi utilizado e avaliado pelo Corpo de Bombeiros do Estado de São Paulo. Esse jogo e uma simulação interativa para celular, nomeada *FireTruckSim* (para controle do painel do caminhão de bombeiros) são descritos na Seção 6. Na próxima seção é apresentada a extensão da metodologia de desenvolvimento de JS, a partir da descrição das ferramentas que podem ser usadas para criar os diferentes artefatos.

5 Artefatos e Ferramentas de Código Livre

O objetivo deste trabalho é estender a especificação da metodologia, que foi criada em Rocha [22], de modo a contemplar as **ferramentas *open source*** que podem apoiar os atores durante seu uso. A escolha das ferramentas foi feita a partir das necessidades dos artefatos que serão criados em cada processo, incluindo os critérios de código livre e multiplataforma (*Windows, Linux e MacOS*).

A escolha das ferramentas abrange três plataformas computacionais distintas, em relação ao jogo sério: (1) a que os aprendizes e instrutores utilizarão para executá-lo (cliente); (2) a que ele ficará disponível e armazenará os resultados do treinamento (servidor); e (3) a que os atores o desenvolverão (produção), conforme a Figura 4.

As plataformas do **cliente** e do **servidor**, utilizadas na execução e avaliação do jogo sério (Figura 3, processos 6 e 7), são especificadas na atividade de "Análise" da "arquitetura de suporte" (Figura 3, processo 2) e projetadas no em "Projeto" de "arquitetura e banco de dados" (Figu-

ra 3, processo 3). Essa especificação inclui os requisitos mínimos e necessários para o cliente (*hardware* e *software*) e o servidor (hospedagem, banco de dados e arquitetura de suporte). O *Motor de jogos, descrito na Figura 4, executará o jogo sério (JS) nos clientes, porém ele será usado durante as fases de Implementação, Integração e Testes, pela equipe desenvolvedora. Da mesma forma, o **SGBD (Sistema de Gerenciamento de Banco de Dados) será executado no servidor para armazenar os dados coletados durante a execução do jogo sério, porém será projetado, criado e testado durante os processos de produção.

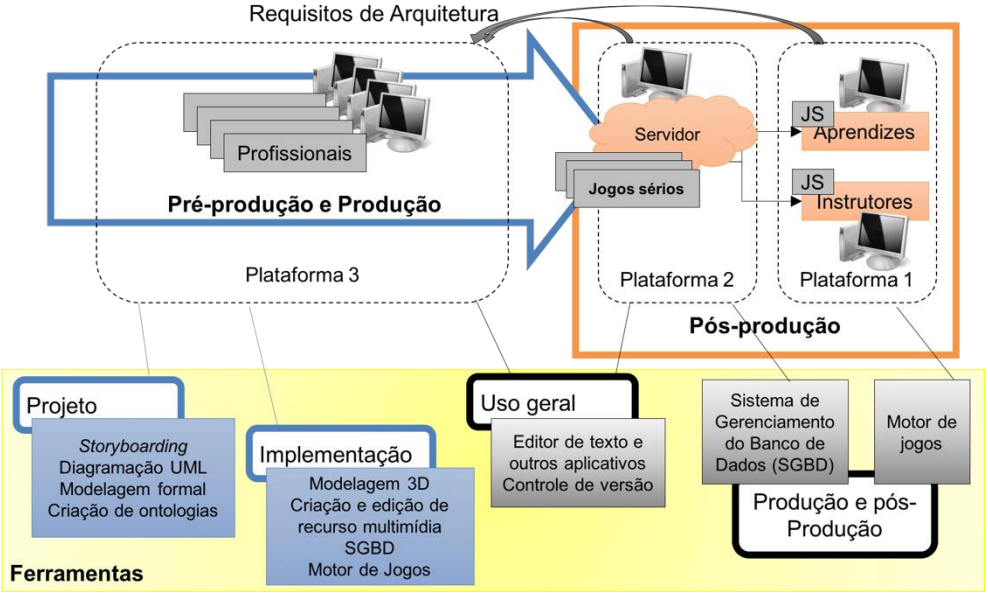

Figura 4: Visão geral das distintas plataformas computacionais e dos grupos de ferramentas relacionadas (adaptado de [32])

A plataforma de **produção** é a plataforma na qual o jogo sério será planejado, analisado, projetado, implementado, integrado e testado pelos atores (Figura 3, processos de 1 a 5). Para isto, devem ser consideradas as necessidades dos treinamento e dos usuários finais (requisitos da arquitetura do JS).

No Quadro 2 é apresenta uma visão geral dos processos da metodologia, suas atividades principais, os papéis dos atores responsáveis por essas atividades e os artefatos criados por eles. Também são relacionadas as ferramentas que foram selecionadas e que são descritas a seguir, agrupadas de acordo com o seu uso: (1) **uso geral**: editor de texto e controle de versão; (2) **sistemas de gerenciamento de banco de dados**; (3) **motores de jogos**; (4) **uso no projeto**: para *storyboarding*, diagramação UML, modelagem formal e criação de ontologias; (2) **uso na implementação**: para modelagem 3D, criação e edição de recurso multimídia (imagem, áudio e vídeo), conforme ilustrado na Figura 4. Em cada subgrupo, as ferramentas são descritas em ordem alfabética.

Processos	Foco das atividades	Papéis dos atores	Artefato(s) de saída	*Uso geral:*	*Uso específico para:*
Planejamento	Treinamento	Especialistas do domínio	Planejamento inicial	**Editor de texto e outros aplicativos do pacote** (Apache OpenOffice, LibreOffice e Google Docs©) **Controle de versão** **- Centralizado** (CVS, SVN) **- Descentralizado** (Git, Mercurial)	
	Jogo & Simulação	Analista			
Análise	Treinamento & Avaliação	Treinador, especialista em treinamentos e/ou pedagogo	Modelo de Especificação de Treinamento e Avaliação		
	Jogo	Analista	Modelo de Especificação de Jogo		
	Simulação	Analista (requisitos) e Especialista no domínio	Modelo de Especificação de Simulação		

	Arquitetura de Suporte	Analista	Modelo de Especificação da Arquitetura		
Projeto	Treinamento & Avaliação	Equipe de desenvolvimento	Modelo Integrador Avaliação – Programa de Treinamento		
	Jogo		*Storyboards*, Protótipos, Diagramas		***Storyboarding*** (GIMP, Inkscape) **Diagramação UML** (Argo UML, Dia, UMLet)
	Simulação		Modelos formais (DFA e eventos discretos - DEVS) Ontologias		**Modelagem formal:** **-DFA** (Automata editor, JFLAP, Visual Automata Simulator) **-DEVS** (CD++ Toolkit, DEVSJAVA 2.7, James II) **Criação de ontologias** (OBO-Edit2, pOWL, Protégé
	Arquitetura e Banco de Dados		Modelos de Arquitetura e Banco de Dados		
Implementação	Arte	Equipe de desenvolvimento	Cenários 2D/3D, animações	**Editor de texto e outros aplicativos do pacote** (Apache OpenOffice, LibreOffice e Google Docs©) **Controle de versão** **- Centralizado** (CVS, SVN) **- Descentralizado** (Git, Mercurial)	**Modelagem 3D** **-Paramétrica** (BRL-CAD, FreeCAD) **-Gráfica** (Art of Illusion, Blender, Wings 3D)
	Multimídia		Áudios, vídeos		**Criação e edição de recurso multimídia** **-Imagem** (GIMP, Inkscape) **-Áudio** (Audacity, LMMS, Sound eXchange, Traverso DAW) **-Vídeo** (Avidemux, CamStudio, Jahshaka, Shotcut)
	Programação		Classes, objetos, banco de dados		**Motor de jogos** (BGE, CS, Delta3D, jME, jPCT-AE, OGRE, OSG, Panda3D **SGBD** **-Relacional** (Firebird, MySQL, PostgreSQL) **-Não relacional** (Cassandra, HBase, MongoDB)
Integração e Testes	Integração	Equipe de desenvolvimento	Jogo final, banco de dados e formulários web		As mesmas ferramentas usadas na implementação
	Testes				
Execução	Instrutor e aprendizes	Simulação & Treinamento	Dados no BD e em formulários web		SGBD, Motor de jogos e Jogo sério (produto final desenvolvido)
		Medição & *Feedback*			
Avaliação	Instrutor	Avaliação do desempenho humano	Relatórios individuais		
	Especialistas no domínio	Avaliação do treinamento	Relatórios de avaliação do programa de treinamento		
Verificação e Validação	Equipe de desenvolvimento	Verificação	Artefatos verificados e validados e relatórios		
	Especialistas no domínio, instrutores e aprendizes	Validação			

Quadro 2: Visão geral das ferramenta que podem ser usadas para gerar os artefatos de cada processo da metodologia

5.1. Ferramentas de Uso em Geral

Durante o desenvolvimento de qualquer *software* é importante documentar todo o projeto sendo desenvolvido, bem como ter o controle de versão de um conjunto de

arquivos, como suporte à colaboração, segurança e a rastreabilidade do que está sendo produzido. Para isso, podem ser usados editores de texto (bem como, outras ferramentas de escritório, que geralmente vêm no pacote, tais como, planilhas e apresentação de slides) e sistemas de controle de versão.

Entre os **editores de texto** de código livre, há o **Apache OpenOffice** [80] desenvolvido em C++ e Java; e o **LibreOffice** [81] desenvolvido em C++, Java e Python; ambos têm versões disponíveis para Linux, MacOS e Windows. O **Google Docs** © [82], para edição de documentos, e outros aplicativos do pacote (por exemplos, Google Presentation ©: apresentações, Google Spreadsheets ©: planilhas, Google Forms ©: formulários) podem ser usados em suas versões gratuitas *online*.

Os **sistemas de controle de versão** são classificados em dois tipos, conforme o gerenciamento do repositório: (1) Centralizado: servidor central único, por exemplo, **CVS** [83] e **SVN** [84]; e (2) Descentralizado: repositório por usuário, tais como, **Git** [85] e **Mercurial** [86]. O CVS (*Concurrent Version System*), sistema mais antigo, possui um funcionamento simples que é ainda muito utilizado. Ele pode ser acessado por *interface* Web (ViewCVS) ou desktop (WinCVS ou MacCVS). Já o SVN (SubVersion) também possui grande aceitação dos desenvolvedores, principalmente em projetos de código livre. Diferentes aplicativos clientes podem ser utilizados para acessar o servidor, dependendo do sistema operacional, tais como, TortoiseSVN (Windows) e Xcode (MacOS). O Git é um sistema de versão distribuído que é rápido e eficiente. Entre os clientes com *interface* gráfica estão o GitHub (Windows e MacOS), e o GitEye e GitCola (ambos Windows, MacOS e Linux). Já o Mercurial é o sistema distribuído mais indicado para projetos de grande porte, devido sua rapidez, escalabilidade e facilidade de uso em relação ao Git. EasyMercurial é um exemplo de *interface open-source* multiplataforma.

5.2. Sistemas de Gerenciamento de Banco de Dados

Um requisito importante dos jogos sérios é a avaliação do desempenho do aprendiz após o treinamento, por meio de relatórios. Para que isto aconteça, é necessário armazenar todos os dados do treinamento em banco de dados. Dessa forma, é importante definir o Sistema de Gerenciamento de Banco de Dados (SGBD), relacional ou não relacional, e projetá-lo.

Os principais SGBDs objeto relacional e multiplataforma são: **Firebird** [66]: ideal para pequenos volumes de dados, possui componentes de acesso em C++, Java, .Net, Python, PHP, Perl, entre outros; **MySQL** [67]: mais comum, ideal para grandes volumes de dados, possui *interface* para diversas linguagens, tais como, C/C++, Java, Python e PHP; e **PostgreSQL** [68]: de rápido desempenho, ideal para grandes volumes de dados, possui recursos de dados estatísticos e *interface* de programação para C/C++, Java, .Net, Python, Perl, Tcl, entre outras.

Para aplicações que demandam escalabilidade, disponibilidade e melhor desempenho do SGBD, soluções NoSQL (não somente SQL) devem ser utilizadas. Nesse caso, algumas opções são: **Cassandra** [69]: possui armazenamento chave/valor, escrito em Java, ideal para armazenar dados muito grandes mas com uma *interface* agradável; **HBase** [70]: orientado a colunas, escrito em Java, possui fácil integração com Hadoop, que é uma plataforma distribuída de processamento de grandes massas de dados não estruturados e semiestruturados; e **MongoDB** [71]: orientado a documentos, escrito em C++, ideal para aplicação que requer consultas dinâmicas.

5.3. Motores de jogos

Motores de jogos, em inglês *game engine*s, são aplicativos e conjuntos de bibliotecas que abstraem funcionalidades importantes na criação de jogos e simulações [56]. Essas funcionalidades podem ir desde a comunicação com dispositivos de entrada (teclado, *mouse*, *joystick*) e saída (*hardware* gráfico/vídeo e áudio), até cálculos físicos e inteligência artificial, conforme estrutura básica apresentada na Figura 5. Devido a quantidade e diversidade de funcionalidades, a criação de um motor de jogo é uma tarefa complexa de programação e custosa em termos de tempo e dinheiro. Por isso, empresas se especializaram em construir motores para jogos, ou componentes de motores. Dessa forma, o desafio é escolher o motor que melhor atende os requisitos do projeto de jogo sério que será desenvolvido. Abaixo são listados motores de código livre e multiplataforma (Windows, Linux e MacOS). Todos estes motores possuem suporte à renderização, texturização, animação, simulação de física e malhas. Apenas o jME e o jPCT-AE foram desenvolvidos em Java, todos os outros usaram C/C++. Uma comparação entre os principais motores de jogos *open source* é apresentada em [57].

Blender Game Engine (BGE) [58] é um sistema de modelagem 3D, animação, composição, renderização e desenvolvimento de simulações e jogos 3D em tempo real. **Crystal Space (CS)** [59] é um *framework* para desenvolvimento de aplicações gráficas 3D, composto por um motor de renderização e um sistema de entidade, que suporta declarações específicas de funções em alto nível. **Delta3D** [60] é um motor de simulação e jogo, que contém uma camada de APIs que integra diversas bibliotecas existentes para diferentes fins. Ele possui também um gerenciador de jogo que administra a comunicação entre as entidades estáticas e dinâmicas, e os dispositivos de entradas. **jMonkey Engine 3.0 (jME)** [61] é um motor desenvolvido em Java, baseado na API gráfica OpenGL.

jPCT-AE [62] é um motor de renderização 3D orientado ao desenvolvimento de jogos para Java e Android. **OGRE** [63] é um motor gráfico orientado a objetos. **OpenSceneGraph (OSG)** [64] é um *toolkit* gráfico de alto nível para o desenvolvimento de aplicações gráficas de alta performance, que possui um *framework* orientado a objetos, em cima da OpenGL, para o rápido desenvolvimento de aplicações gráficas. **Panda3D** [65] é motor de jogo e um *framework* para renderização 3D, que utiliza um grafo de cena para gerenciar os objetos no ambiente virtual.

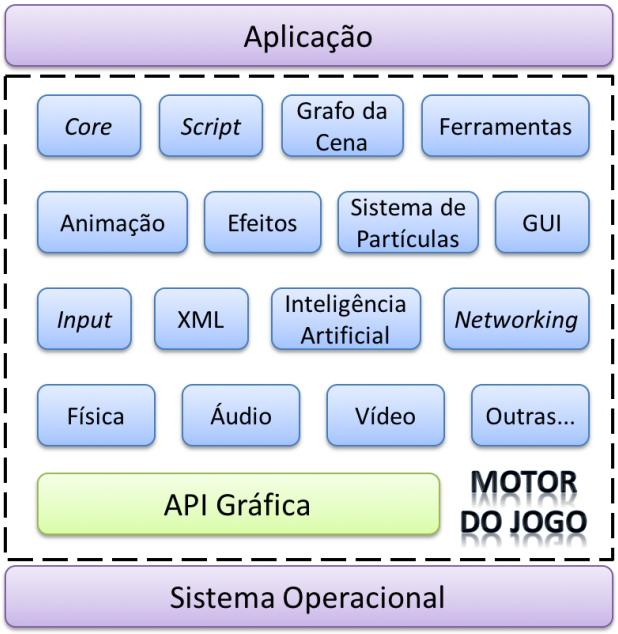

Figura 5: Visão geral da estrutura básica de um motor de jogo

5.4. Ferramentas de Apoio ao Projeto do Jogo Sério

Um dos diferenciais da metodologia criada, descrita em Rocha [22], é a especificação e o uso de modelos integradores como base para o projeto dos JSs, de modo a possibilitar que equipes multidisciplinares projetem a simulação, o jogo e cada uma de suas fases, com as avaliações e *feedback* necessários. Isso inclui *storyboards*, que integram a descrição do cenário com as interações; diagrama estruturado, que integra objeto do jogo e simulação; modelos formais, que integram simulações e avaliações; e ontologias que descrevem o conhecimento do domínio e cenários simulados. A seguir são descritas ferramentas que podem ser usadas para criar esses artefatos.

Storyboards são usados para entender e projetar o cenário de treinamento, seus objetos e personagens, e descrever a *interface* de interação e o fluxo de ações que pode acontecer durante o treinamento [33]. Eles apoiam o trabalho colaborativo por serem de fácil entendimento, tanto para os desenvolvedores quanto para o responsável técnico e as outras pessoas envolvidas. As ferramentas de código livre que podem ser usadas são os editores de imagens: **GIMP** [34]: para imagens *bitmap* (mapa de bits), desenvolvido em C e GTK+; e **Inkscape** [35] para imagens vetoriais, desenvolvido em C e C++. Ambos têm versões para Windows, Linux e Mac OS. A diferença é o tipo de imagem a ser criada, sendo que cada ferramenta tem funcionalidades específicas para isto.

Diagramas baseados na Linguagem de Modelagem Unificada (*Unified Modeling Language*™ - UML) devem ser utilizados para representar modelos estruturados e/ou comportamentais. Além disto, um diagrama baseado no diagrama estruturado de classe UML foi especificado em Rocha [22] para integrar diferentes características de um objeto simulado: arte, áudio, programação e *interface*. Para modelar estes diagramas, podem ser usadas as ferramentas: **Dia** [36] desenvolvida em C, **ArgoUML** [37] e **UMLet** [38], ambas desenvolvidas em Java.

Modelos formais devem ser utilizados para modelar o comportamento do procedimento de treinamento e os pontos de avaliação do desempenho humano. Em Rocha [22], foram especificados os processos para a modelagem do treinamento usando Especificação de Sistemas de Eventos Discretos (DEVS) (quando o fator tempo implica em mudanças de estado) e/ou Autômato Determinístico Finito (DFA) (quando o tempo não influencia a simulação). Para a visualização e modelagem DEVS, podem ser usados: **CD++ Toolkit** [39] desenvolvida em C++; **DEVSJAVA** [40] e **JAMES II** [41], ambas desenvolvidas em Java. Para a modelagem DFA, podem ser usados: **Automata Editor** [42]: editor para desenhar DFA, desenvolvido em C/C++; **JFLAP** [43]: ferramenta gráfica de edição de autômatos formais determinísticos e não determinísticos (DFA e NFA), desenvolvida em Java; **Visual Automata Simulator** [44]: ferramenta para simular, visualizar e editar DFA ou NFA, também desenvolvida em Java.

Ontologia pode ser definida como um corpo de conhecimento para descrever um domínio [45]. Ela pode ser utilizada para diferentes propósitos, tais como, representar o conhecimento de um domínio ou definir as características do ambiente de simulação [22, 46, 47]. As ontologias podem ser criadas a partir de editores visuais, tais como as ferramentas independentes de plataforma (Windows, Mac, Linux): **OBO-Edit2** [48]: editor de ontologias baseadas em grafo direcionado acíclico; **pOWL** [49]: plataforma de desenvolvimento de aplicação Web Semântica, com ontologias baseadas em frames; e **Protégé** [50]: *framework* e editor de ontologias baseadas em *frames*, com diferentes tipos de arquivos. Os editores Protégé e OBO-Edit2 foram desenvolvidos em Java e podem ser executados no computador local ou via web. Já o pOWL foi desenvolvido em PHP e banco de dados SQL,

sendo executado apenas pelo servidor web, tendo que instalá-lo para utilizar. O Protégé é o editor mais utilizado pela comunidade, e, além de ter um *framework*, possui inúmeros *plug-ins* com diferentes funcionalidades. O pOWL também contém *plug-ins* e uma API orientada a objeto. Já o OBO-Edit2 tem uma API.

5.5. Ferramentas de Apoio à Implementação do Jogo Sério

O jogo contém recursos (também chamados de *assets*) de texto, arte (imagens 2D e modelos 3D), áudio, vídeo, código, etc. Após o projeto, este conteúdo deve ser criado ou reusado de repositórios (gratuitos ou pagos). Para criar, modelar e editar esses *assets*, diversas ferramentas podem ser usadas. A seguir são apresentadas as ferramentas de modelagem 3D, motores de jogos e banco de dados). Elas devem ser instaladas e usadas em um computador local.

Para criação de modelo 3D, há *software* de modelagem paramétrica (sólidos 3D), tais como, o **BRL-CAD** [51] desenvolvido em C, C++ e Tcl, e **FreeCAD** [52] desenvolvido em C++ e Python; e *software* de modelagem gráfica que possui diferentes funções, que vão desde texturização até animação, tais como, **Blender** [53] que também possui um motor de jogos, desenvolvido em C, C++ e Python; **Art of Illusion** [54] modelagem e animação, desenvolvido em Java; e **Wings3D** [55] que inclui texturização, desenvolvido em Erlang.

Recursos multimídia incluem imagens, áudios e vídeos digitais. As ferramentas de edição de imagens são: **GIMP** [34] e **Inkscape** [35], as mesmas para a criação de *Storyboards,* descritas na subseção 5.4.

Para edição de **áudio** digital, há quatro principais ferramentas de gravação e edição que são multiplataforma: **Audacity** [72]: gravação e tratamento de áudios (eliminação de ruídos e voz), desenvolvido em C e C++; **LMMS** [73]: estação de áudio com instrumentos musicais e edição de arquivos MIDI, desenvolvido em C++; **Sound eXchange** [74]: edição de áudio, desenvolvido em C; **Traverso DAW** [75]: gravação de áudio a partir de linha de entrada, desenvolvido em C++. De forma geral, esses editores possuem diferentes recursos e efeitos de edição, e suporte de formato de entrada e saída.

Para edição de **vídeo**, as ferramentas multiplataforma são: **Avidemux** [76]: edição e processamento de vídeos, desenvolvida em C++; **CamStudio**: para gravação de vídeo a partir da tela [77]; **Jahshaka** [78]: inclui edição e animação de conteúdo 3D, desenvolvida em Phyton; e **Shotcut** [79]: inclui a gravação de vídeo a partir da tela e da webcam, desenvolvida em C++.

6 Uso das Ferramentas e Resultados

Como resultado deste trabalho, foi feita uma revisão e seleção das principais ferramentas *open source* que podem ser utilizadas em conjunto com a metodologia de desenvolvimento de jogos sérios, para criar os artefatos em cada processo. Cada ferramenta pode ser melhor utilizada em um tipo de cenário de aplicação, de acordo com as necessidades. Dois exemplos de cenários desenvolvidos usando a metodologia (*FireTruckSim* e *GLPSobControle*) e diferentes ferramentas são descritos a seguir (resumidos no Quadro 3).

Processos	Foco da atividades	*FireTruckSim*	*GLPSobControle*
Planejamento	Treinamento	Google Docs ©	Google Docs ©
	Jogo & Simulação		
Análise	Treinamento & Avaliação	Google Docs ©	Google Docs ©
	Jogo		
	Simulação		
	Arquitetura de Suporte		
Projeto	Treinamento & Avaliação	Não há	Google Apresentações©
	Jogo	Papel e lápis e Dia (UML)	Google Apresentações© (*interface*) e ArgoUML (UML)
	Simulação	JFLAP(DFA)	Automata Editor (DFA)
	Arquitetura e Banco de Dados	Papel e lápis	Google Apresentações©
Implementação	Arte	Blender, reúso (modelo 3D)	Microsoft Power Point (imagens 2D), Unity3D (cenários 2D/3D, animações e sistemas de partículas), reúso (modelos 3D)
	Multimídia	Não há	CamStudio (vídeo), reúso (áudio)
	Programação	jPCT-AE e Java	Unity3D, C#, PHP, MySQL
Integração e Testes	Integração	jPCT-AE e Java	Unity3D, C#, PHP e MySQL
	Testes		

Quadro 3: Resumo das diferentes ferramentas usadas no desenvolvimento da simulação *FireTruckSim* e do jogo sério *GLPSobControle*

6.1 Simulação interativa *FireTruckSim*

FireTruckSim é uma simulação simples 2D/3D, para celulares com o sistema operacional *Android,* para controle do painel da viatura do Corpo de Bombeiros (operação da bomba de água), conforme Figura 6 [87]. Os requisitos iniciais dessa simulação eram: (1) o uso de diferentes dispositivos de interação (celular e *desktop*); (2) uso de modelos 3D, além do 2D; (2) simulação distribuída, ou seja, simulação executada no servidor e visualizada nos dispositivos de interação.

Figura 6: *FireTruckSim*: simulação criada para celulares Android [87]

O plano inicial (processo de Planejamento) e a especificação (Análise) foram criados colaborativamente com o Google Docs ©, usando os modelos de artefatos descritos na metodologia. No processo de Projeto, a arquitetura e *storyboards* foram criados usando papel e lápis. A arquitetura foi especificada usando o modelo em camadas: apresentação (*interface* 2D/3D para diferentes dispositivos), comportamento (simulação usando DFA), e comunicação (baseada no protocolo e arquitetura de alto-nível HLA). Foram usadas as ferramentas Dia (para modelagem dos diagramas de UML) e JFLAP (para o DFA - apresentado na Figura 7).

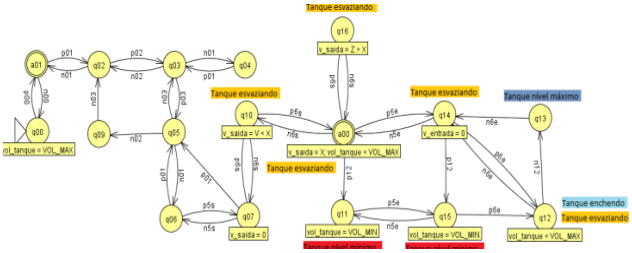

Figura 7: *FireTruckSim*: DFA modelado com JFLAP [87]

No processo de Implementação, Blender foi usado na modelagem dos elementos gráficos (para reúso de um modelo 3D gratuito disponível na Internet e uso de imagens 2D criadas por um *designer* externo a equipe do projeto) e o motor de jogos jPCT-AE foi usado na implementação da *interface* para celular Android [86]. Além disso, houve a implementação do serviço de simulação do DFA (no servidor) e do serviço de comunicação HLA usando java e a API poRTIco que implementa o protocolo HLA (em ambos, cliente e servidor). A integração dos recursos criados e usados foi feita usando a linguagem Java.

Os principais desafios do projeto *FireTruckSim* foram: (1) a necessidade de motor gráfico/ de renderização 3D adequado para celular, para não consumir muitos recursos do sistema (processamento, memória, bateria, etc.); (2) a falta de documentação, suporte e exemplos de uso dos motores de jogos/gráficos para celulares; e (3) a necessidade de acesso a uma versão estável do código-fonte do motor de jogo/gráfico para implementar a comunicação com o servidor. Vários motores de jogos para a plataforma Google Android foram comparados e alguns testados (por exemplo, Ardor3D, Forget3D, Delta 3D, O3D, X3DOM, jMonkey Engine), sendo que o jPCT-AE foi o selecionado a partir dos requisitos do projeto. Por exemplos, Delta3D implementava HLA mas não tinha uma versão para Android e Ardor3D não tinha uma versão estável e suporte ao desenvolvedor.

A simulação *FireTruckSim* foi testada e avaliada pela equipe de desenvolvimento e especialista no domínio, sendo que os requisitos iniciais de desenvolvimento foram atingidos.

6.2 Jogo sério *GLPSobControle*

O segundo exemplo é um jogo sério, com três fases 2D e três fases 3D, multiplataforma, com acesso on-line via navegador *Web*, para controle de vazamento de gás de cozinha, conforme Figura 8 [22]. Os requisitos iniciais desse jogo sério eram: (1) a criação de diferentes fases, com diferentes recursos de interação (jogo 2D, ambiente virtual 3D, vídeo); (2) acesso via Web (multiplataforma) com coleta e armazenamento dos dados no servidor; (3) uso da arquitetura em camadas (descrita em 6.1) e possibilidade de implementação de simulações distribuídas.

Figura 8: *GLPSobControle*: jogo sério multiplataforma [22]

Nos processos de Planejamento e Análise, o plano inicial e a especificação foram criados usando o Google Docs©, a partir dos artefatos descritos na metodologia. No processo de Projeto, a arquitetura, o programa de treinamento e avaliação e os *storyboards* foram criados usando o Google Apresentações©. Também foram usadas as ferramentas *open source* ArgoUML e Automata Editor para modelagem dos diagramas de UML (projeto do

jogo) e DFA (modelagem da simulação).

Na implementação, o banco de dados foi desenvolvido em linguagem MySQL com acesso implementado em PHP. A arquitetura cliente e servidor web é apresentada na Figura 9 [22]. Foi utilizado a versão gratuita do motor de jogo Unity3D© [88], com programação na linguagem C#. Recursos de arte (texturas e modelos 3D) e áudio foram reusados de repositórios gratuitos disponíveis na Internet e integrados ao ambiente virtual usando a Unity3D. As imagens 2D foram criadas usando o *Microsoft Power Point*© e salvas no formato .png. O arquivo de vídeo foi gravado usando a ferramenta CamStudio, pois o vídeo apresenta a execução da fase 3 do jogo, sendo que a sequência de ações é feita incorretamente. A integração dos recursos criados foi feita usando o motor de jogos Unity 3D (arte, áudio e programação) e as linguagens C# (cliente) e PHP (servidor).

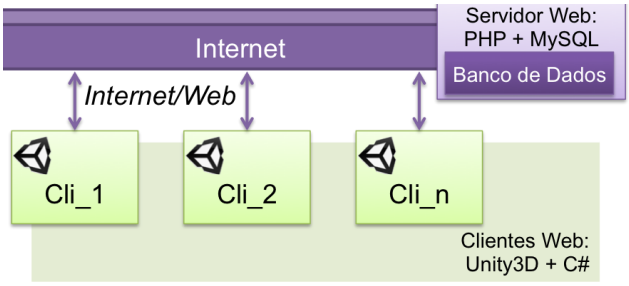

Figura 9: *GLPSobControle*: arquitetura com acesso via Web [22]

Os principais desafios do projeto *GLPSobControle* foram: (1) o reúso de modelos 3D com diferentes resoluções; (2) a necessidade de sistemas de partículas para simulação de explosão e vazamento de gás; e (3) a falta de documentação, suporte e exemplos de uso dos motores de jogos. Vários motores de jogos foram comparados e alguns testados (por exemplo, Blender Game Engine, Crystal Space, Delta 3D, Irrlicht, jMonkey Engine, Ogre3D, OpenSceneGraph, Panda3D), sendo que a Unity3D, apesar de não ser *open source*, foi selecionada a partir dos requisitos do projeto. Dentre os melhores motores de jogos analisados estavam Delta3D e JMonkey Engine 2.0, pois eram as únicas das ferramentas *open source* analisadas que tinham efeitos especiais para explosão, sendo que Delta3D não tinha uma versão estável e suporte ao desenvolvedor; e jMonkeyEngine é um motor baseado em grafos (i.e., representa todos os elementos dos jogos, geometrias, áudio, física, etc., em grafos), o que dificultou a integração de diferentes modelos 3D. Ao passo que, a Unity3D é um ambiente de desenvolvimento (IDE - *Integrated Development Environment*) e motor para desenvolvimento de jogos que possui uma *interface* gráfica, na qual é possível importar, criar e configurar os recursos (tais como, modelos 3D; imagens e texturas; áudio e vídeo; sistema de partículas para simulação de explosão, água, poeira, entre outras), além da programação usando *scripts*. Além disso, há documentação, tutoriais e uma comunidade ativa de desenvolvedores.

O jogo sério *GLPSobControle* foi testado e avaliado pela equipe de desenvolvimento e por dois especialistas no domínio; e foi feito um estudo de caso com oito bombeiros que utilizaram o jogo sério. Os resultados indicam que este JS possibilita o treinamento das competências intencionadas. Além disto, ele permitiu identificar uma oportunidade de melhoria no protocolo operacional treinado [22].

7 Discussões

A efetividade de um jogo sério é resultado da sua efetiva concepção e produção somado ao seu correto uso. Sendo assim, um metodologia com qualidade aumentará a probabilidade do jogo ser efetivo. Essa qualidade é influenciada pelos **processos** (usados para desenvolver os artefatos); pelos próprios **artefatos** criados; por seu **projeto** (planejamento, gerenciamento de risco, controle e monitoramento); e pelos **atores** (ou seja, suas competências para criar os artefatos) [2]. Um artefato define "o que" se quer produzir e um processo "como" será produzido. Entretanto, as competências dos atores influenciam nesse processo e no resultado do artefato criado, principalmente pela multidisciplinariedade requerida no desenvolvimento de jogos sérios.

Um jogo sério contém requisitos em comum a recursos afins (jogo, simulação, material educacional), para os quais existem metodologias específicas de desenvolvimento. Entretanto, adotar uma dessas metodologias não é suficiente para cobrir todos os requisitos de jogos sérios. Além disso, se cada profissional adotar uma metodologia própria, poderá haver falta de integração e/ou desbalanceamento da jogabilidade e efetividade da aprendizagem com o JS [31]. Entretanto, por serem metodologias de áreas bem mais desenvolvidas, muitas teorias e práticas evoluídas e usadas em cada uma dessas áreas podem beneficiar o desenvolvimento de jogos sérios, se elas forem integradas ao seu desenvolvimento.

Nesse contexto, uma metodologia que integra e padroniza "o que" e "como" deve ser produzido em cada etapa pode auxiliar na produção de um jogo sério efetivo. Essa padronização também pode possibilitar o reúso de artefatos, e até a sua geração automática. Entretanto, as metodologias existentes ou focam requisitos específicos dos jogos sérios, ou, quando mais abrangentes, não abordam todos os requisitos para uma aprendizagem efetiva. Essas limitações são, muitas vezes, causadas pelo próprio contexto de que jogos sérios são inseridos em outras áreas de pesquisa e desenvolvimento, tais como, soluções em simulação distribuída, engenharia de *software*, jogo como objeto educacional, realidade virtual e aumentada, treinamento e desenvolvimento humano, sistemas de

medição de desempenho humano; ou em subáreas de pesquisa, tais como, desenvolvimento de jogos para dispositivos específicos (por exemplo, *consoles* ou celulares) ou jogos para domínios particulares (por exemplo, saúde, educação).

Além da padronização dos processos e dos artefatos que devem ser produzidos, entender as características das plataformas e ferramentas é o primeiro passo tanto para poder utilizá-las, como para estendê-las, de modo a criar um ambiente e *framework* que possibilite mais recursos de apoio à equipe de desenvolvimento, tanto nos aspectos de reúso e interoperabilidade de artefatos, quanto em sua geração automática.

Nesse contexto, uma nova metodologia foi proposta e avaliada por Rocha [22], descrevendo processos, papéis dos atores, e artefatos de entrada e saída. Entretanto, abordou parcialmente a descrição de ferramentas. Dessa forma, este artigo propõe uma extensão à metodologia, especificando ferramentas de código livre que podem ser usadas.

De forma geral, ter um conjunto de opções de ferramentas possibilita a escolha do que mais se adequa a cada contexto. Ainda mais, essas ferramentas de código livre e seus artefatos gerados podem ser estendidos para padronizar modelos de saídas conforme artefatos de entradas requeridos nos processos seguintes, diminuindo esforços no desenvolvimento (tempo, dinheiro, recursos humanos). A seguir são discutidas algumas possibilidades de como essas ferramentas e artefatos podem ser expandidos.

As ferramentas de modelagem DFA e DEVS podem ser estendidas para gerar os modelos padronizados que servem de entradas para os componentes que simulam o treinamento e avaliação. Da mesma forma, as ferramentas de diagrama UML podem ser estendidas para contemplar as alterações realizadas no diagrama estruturado de classe, o qual especifica as características do objeto simulado.

As ontologias, além de especificar o cenários do programa de treinamento, poderiam ser utilizadas para especificar os dados que serão coletados e armazenados no banco de dados, bem como serem usadas para gerar as tabelas automaticamente.

Além disso, a identificação dos formatos padrões, que as ferramentas geram para os diferentes *assets* (tais como, modelos 3D, imagens e texturas, áudios e vídeos) possibilitará a criação de um repositório compartilhado para o desenvolvimento de novos jogos sérios, podendo até, dada a padronização ou com uso de conversores, serem reusados em diferentes plataformas.

8 Considerações Finais

Há uma lacuna entre as áreas que oferecem apoio à criação de jogos sérios (*design* de jogos, modelagem de simulações, *design* de instrução e treinamento), de modo que as metodologias propostas para o desenvolvimento de jogos sérios e educacionais são limitadas ao não abrangerem todos os requisitos necessários de cada área.

Além disto, ao mesmo tempo que a metodologia tem que guiar e apoiar o desenvolvimento do jogo sério, de forma sistemática, padronizada e multidisciplinar, ela também tem que ser flexível para atender às diferentes necessidades dos atores que estão usando-a. Desta forma, os produtos gerados e ferramentas usadas mudam dependendo das características do jogo sério e da equipe (quantidade e competências dos atores). Assim, a escolha das ferramentas adequadas impacta no sucesso do desenvolvimento do jogo sério, tanto em termos de custo financeiro, quanto de prazo de entrega e qualidade do produto final. Como por exemplo, um modelo 3D pode não ser exportado para um formato aceito por um motor de jogo, ou quando exportado ficar abaixo da qualidade aceitável. Isto implicaria em retrabalho, o que aumentará o custo e prazo de entrega do produto final.

Neste artigo, foi apresentado um conjunto de ferramentas de código aberto que pode ser utilizado para criar os artefatos em cada processo da metodologia. Além disto, foram apresentados dois estudos de caso, entre os 20 protótipos desenvolvidos com o uso da metodologia, para exemplificar os usos dessas ferramentas.

Como trabalhos futuros, estas ferramentas deverão ser analisadas e estendidas, de modo a facilitar o trabalho dos atores, por meio da geração de artefatos de diferentes processos do ciclo de vida da metodologia, seu reúso (criação de um repositório de artefatos) e interoperabilidade de objetos e ambientes simulados.

Outros aspectos da metodologia também deverão ser aprofundados, tais como: (1) identificação de cenários com diferentes necessidades de projeto do jogo sério, para realização de novas provas de conceitos usando a metodologia e as ferramentas descritas; (2) gerenciamento da qualidade dos projetos e equipe de desenvolvimento, abordando a descrição e avaliação das competências necessárias para cada ator envolvido; e (3) definição de métricas e indicadores para avaliar o uso da metodologia.

Agradecimentos

Agradecemos ao CNPq, FAPESP e CAPES (projeto Procad e processos 2010/17005-7, 237216/2012-4 e 150613/2015-6), pelo apoio financeiro; ao Corpo de Bombeiros do Estado de São Paulo, pelo auxílio; e a todos que participaram do projeto na Universidade Fede-

ral de São Carlos, pelas colaborações.

Referências

[1] C. Aldrich. Learning by doing: a comprehensive guide to simulations, computer games, and pedagogy in e-learning and other educational experiences. Pfeiffer, San Francisco, 2005.

[2] O. Balci. A Life Cycle for Modeling and Simulation. *Simulation*. 88(7):870-883, 2012.

[3] S. Aslan, O. Balci. GAMED: Digital Educational Game Development Methodology. *Simulation*. 91(4):307-319, 2015.

[4] W.K. Adams, et al. A Study of Educational Simulations Part I – Engagement and Learning. Journal of Interactive Learning Research, 18(3):397-419, 2008.

[5] G.K. Akilli; K. Cagiltay. An Instructional Design/Development Model for the Creation of Game-Like Learning Environments: the FIDGE model. In: Pivec, M. (Ed.), Affective and Emotional Aspects of Human-Computer Interaction: Game-Based and Innovative Learning. IOS Press, páginas 93-112, 2006.

[6] K. Becker; J. Parker. Serious Instructional Design: ID for digital simulations and games. In P. RESTA (ED.), *Proceedings of Society for Information Technology & Teacher Education International Conference* 2012. Chesapeake: AACE, páginas 2480-2485, 2012.

[7] S. Freitas; S. Jarvis. A Framework for Developing Serious Games to meet Learner Need. In *Interservice/Industry Training, Simulation, and Education Conference*, páginas 1-11, 2006.

[8] R.T. Hays. The Effectiveness of Instructional Games: a literature review and discussion. Technical Report, Naval Air Warfare Center Training Systems Division, 2005

[9] H. Engström; E. Ambring; C-J Dahlin; E. Sjöstrand. Making a Game of the Old Testament Balancing Authenticity, Education and Entertainment. *IADIS International Journal on WWW/Internet*, 9(1):1-7, 2011.

[10] J. Banks, J.S. Carson II, B.N. Nelson, D.M. Nicol. Discrete-Event System Simulation. 3rd. ed. Prentice-Hall, New Jersey, 2001.

[11] IEEE. 1516.3-2003- IEEE Recommended Practice for High Level Architecture (HLA): federation development and execution process (FEDEP). IEEE, New York, 2003.

[12] IEEE. 1730-2010- Distributed Simulation Engineering and Execution Process (DSEEP). IEEE, Washington, 2010.

[13] F. Ford. The Euclid RTP 11.13 SE Development & Exploitation Process (SEDEP). European Simulation Interoperability Workshops, 04E-SIW-037, Edinburgh, páginas 1-10, 2004.

[14] H.M. Chandler. Manual de Produção de Jogos Digitais. 2. ed. Bookman, Porto Alegre, 2012.

[15] J. Novak. Desenvolvimento de Games. Cengage, São Paulo, 2010.

[16] P. Schuytema. Design de Games: uma abordagem prática. Cengage, São Paulo, 2008.

[17] R.K. Branson, G.T. Rayner, J.L. Cox, J.P. Furman, F.J.King, W.H.Hannum. In *Interservice Procedures for Instructional Systems Development.* (5 vols.). Ft. Monroe, U.S. Army Training and Doctrine Command, 1975.

[18] W. Dick; L. Carey; J.O Carey. The Systematic Design of Instruction. Allyn & Bacon, 2004.

[19] J. Kemp. Instructional Design: a plan for unit and course development. Fearon-Pitman, Belmont, 1977.

[20] ABNT. NBR ISO 10015:2001 - Gestão da Qualidade: diretrizes para treinamento. ABNT, Rio de Janeiro, 2001.

[21] J. Trybus. Game-Based Learning: What it is, Why it Works, and Where it's Going. NMI White Paper. New Media Institute, New York, 2010.

[22] R.V. Rocha. Metodologia iterativa e modelos integradores para desenvolvimento de jogos sérios de treinamento e avaliação de desempenho humano. Tese de Doutorado, Universidade de São Carlos, 2014.

[23] A.H. Feinstein; H.M. Cannon. Constructs of Simulation Evaluation. *Simulation & Gaming*, 33(4): 425-440, 2002.

[24] R.T. Hays; M.J. Singer. *Simulation Fidelity in Training System Design: bridging the gap between reality and training*. Springer, New York, 1989.

[25] D. Michael; S. Chen. *Serious Games: games that educate, train, and inform.* Thomson Course Technology, Boston, 2006.

[26] J. Gordon; R. Zemke. The Attack on ISD. *Training Magazin*e, 37(4):42, 2000.

[27] N.A.M, Zin; A. Jaafar; W.S. Yue. Digital Game-Based Learning (DGBL) Model And Development Methodology For Teaching History. *WSEAS Transactions On Computers*, 2(8):322-333, 2009.

[28] S. Kirkley; S. Tomblin; J. Kirkley. Instructional Design Authoring Support for the Development of Serious Games and Mixed Reality Training. In *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC).* Bloomington, páginas 1-11, 2005.

[29] H.F. Rodrigues; L.S. Machado; A.M. Valença. Definição e Aplicação de um Modelo de Processo para o Desenvolvimento de Serious Games na Área de Saúde. In *Proc. Congresso da Socieda-*

de Brasileira de Computação - Workshop de Informática Médica, páginas 1532-1541, 2010.

[30] D.J. Van Der Zee; B. Holkenborgb; S. Robinson. Conceptual modeling for simulation-based serious gaming. In *Decision Support Systems*, v. 54(1):33-45, 2012.

[31] B.B. Marklund. *Games in Formal Educational Settings Obstacles for the Development and Use of Learning Games*. Licentiate Dissertation (Informatics), University of Skövde, Sweden, 2013.

[32] R.V. Rocha; Zem-Lopes, A.M.; L.Z. Pedro; I.I. Bittencourt; S. Isotani. Metodologia de Desenvolvimento de Jogos Sérios: especificação de ferramentas de apoio open source. In *XXVI Simpósio Brasileiro de Informática na Educação*, Maceió, páginas 489-498, 2015.

[33] A. Rankin, J.N. Field, R. Kovordanyi, M. Morin, J. Jenvald, H. Eriksson. Training Systems Design: bridging the gap between users and developers using storyboards. In *Proceedings of the 29th Annual European Conference on Cognitive Ergonomics*, New York, páginas 205-212, 2011.

[34] The GIMP Team. GIMP: GNU Image Manipulation Program. http://www.gimp.org/, 2015.

[35] The Inkscape Team. Inkscape. https://inkscape.org/, 2015.

[36] The Gnome Project. DIA. https://wiki.gnome.org/Apps/Dia, 2015.

[37] CollabNet, Inc. ArgoUML. http://argouml.tigris.org, 2015.

[38] UmLet. UMLet: Free UML Tool for Fast UML Diagrams. http://www.umlet.com, 2015.

[39] G. Wainer. C++ toolkit for Discrete-Event modeling and simulation. http://cell-devs.sce.carleton.ca, 2015.

[40] The Arizona Center for Integrative Modeling and Simulation (ACIMS). DEVSJAVA. http://acims.asu.edu/software/devsjava, 2015.

[41] Modeling & Simulation Research Group. JAMES II: JAva-based Multipurpose Environment for Simulation II. www.jamesii.org, 2015.

[42] Automata Editor. http://automataeditor.sourceforge.net, 2015.

[43] S.H. Rodger. JFLAP. http://www.jflap.org/jflaptmp, 2015.

[44] J. Bovet. VAS: Visual Automata Simulator. https://www.cs.usfca.edu/~jbovet/vas.html, 2015.

[45] B. Swartout; R. Patil, K. Knight; T. Russ. Toward Distributed Use of Large-Scale Ontologies. In *Proceedings of AAAI97 Spring Symposium Series Workshop on Ontological Engineering*: AAAI Press, páginas 138-148, 1997.

[46] W. Bille; O. Troyer; F. Kleinermann; B. Pellens; R. Romero. Using Ontologies to Build Virtual Worlds for the Web. In *IADIS International WWW/Internet 2004 Conference*, páginas 683-690, 2004.

[47] D.C. Paiva. Modelagem e Simulação de Multidões Humanas em Situações da Vida Cotidiana usando Ontologias. Dissertação de Mestrado, Universidade do Vale do Rio Sinos, 2006.

[48] The OBO-Edit Working Group. OBO-Edit2. http://oboedit.org/, 2015.

[49] Sören Auer. pOWL: Semantic Web development platform. http://sourceforge.net/projects/powl/, 2015.

[50] Stanford Center for Biomedical Informatics Research. Protégé. http://protege.stanford.edu, 2015.

[51] U.S. Army Research Laboratory. BRL-CAD: Open Source Solid Modeling. www.brlcad.org, 2015.

[52] The Freecad Team. FreeCad: an open-source parametric 3D CAD modeler. http://www.freecadweb.org, 2015.

[53] Blender Foundation. Blender Game Engine. http://www.blender.org, 2015.

[54] Peter Castman. Art of Illusion. http://www.artofillusion.org, 2015.

[55] D. Gudmundsson, R. Jones. WINGS 3D. http://www.wings3d.com, 2015.

[56] B. Feijó, E.W. Clua, F.S. Silva. Introdução à Ciência da Computação com Jogos: aprendendo a programar com entretenimento. Elsevier, Rio de Janeiro, 2010.

[57] R.V.Rocha; R.B. Arajo. Metodologia de Design de Jogos Sérios para Treinamento: Ciclo de vida de criação, desenvolvimento e produção. In *XII Simpósio Brasileiro de Jogos e Entretenimento Digital*, São Paulo, páginas 63-72, 2013.

[58] Blender Foundation. Blender Game Engine: introduction. https://www.blender.org/manual/game_engine/introduction.html, 2015.

[59] Crystal Space Team. Crystal Space. http://www.crystalspace3d.org, 2015.

[60] Delta3D. Delta3D. http://www.delta3d.org, 2015.

[61] jMonkeyEngine. jMonkeyEngine: a cross-platform game engine for adventurous Java developers. http://jmonkeyengine.org, 2015.

[62] H. Foerster. jPCT-AE: the free 3D solution for Java and Android. http://www.jpct.net/jpct-ae, 2015.

[63] Torus Knot Software Ltd. OGRE: Object-Oriented Graphics Rendering Engine. http://www.ogre3d.org, 2015.

[64] OSG Community. OpenSceneGraph. http://trac.openscenegraph.org/projects/osg/, 2015.

[65] Carnegie Mellon University. Panda3D. https://www.panda3d.org, 2015.

[66] Firebird Foundation Incorporated. Firebird: true universal open source database. http://www.firebirdsql.org, 2015.

[67] Oracle Corporation. MySQL: the world's most popular open source database. https://www.mysql.com, 2015.

[68] The PostgreSQL Global Development Group. PostgreSQL: the world's most advanced open source database. http://www.postgresql.org, 2015.

[69] The Apache Software Foundation. Cassandra. http://cassandra.apache.org, 2015.

[70] The Apache Software Foundation. HBase. http://hbase.apache.org, 2015.

[71] Mongo. MongoDB. http://www.mongodb.org, 2015.

[72] The Audacity Team. Audacity: open source and cross-platform software for recording and editing sounds. http://web.audacityteam.org, 2015.

[73] The LMMS Team. Linux MultiMedia Studio (LMMS): open source digital audio workstation. https://lmms.io, 2015.

[74] SoX. Sound eXchange. http://sox.sourceforge.net, 2015.

[75] The Traverso Team. Traverso DAW. http://traverso-daw.org, 2015.

[76] Avidemux. http://avidemux.sourceforge.net, 2015

[77] CamStudio Open Source: free streaming vídeo software. http://camstudio.org/, 2015.

[78] The Jahshaka Project Ltd. Jahshaka: realtime editing and effects. www.jahshaka.com, 2015.

[79] Meltytech, LLC. Shotcut. http://shotcut.org, 2015.

[80] The Apache Software Foundation. Apache OpenOffice: the free and open productivity suite. https://www.openoffice.org/pt-br/. 2015.

[81] The Document Foundation. LibreOffice: uma suite office livre. https://pt-br.libreoffice.org/, 2015.

[82] Google. Google Docs https://www.google.com/docs/about/, 2015.

[83] Free Software Foundation, Inc. CVS: Concurrent Version System. http://savannah.nongnu.org/projects/cvs, 2015.

[84] The Apache Software Foundation. SVN: Apache Subversion. http://subversion.apache.org, 2015.

[85] Software Freedom Conservancy. Git. http://git-scm.com, 2015.

[86] Matt Mackall. Mercurial. https://mercurial.selenic.com/wiki/Mercurial, 2015.

[87] R.H.P. Lima; R.B. Araujo. Interface de Visualização 3D em Dispositivos Móveis para Simulações de Treinamento, Relatório Científico, Universidade Federal de São Carlos, 2011.

[88] Unity Technologies. Unity3D. http://unity3d.com, 2015.

[89] F. Bellotti; B. Kapralos; P. Moreno-Ger; R. Berta. Assessment in and of Serious Games: An Overview. In *Advances in Human-Computer Interaction*, páginas 1-11, 2013.

[90] T.M. Connolly; E.A. Boyle; E. MacArthur; T. Hainey. J. Boyle. A systematic literature review of empirical evidence on computer games and serious games. *Computers & Education*, 59(2): 661–686, 2012.

[91] L. Donovan. *The Use of Serious Games in the Corporate Sector: a state of the art report*. Learnovate Centre, 2012. Disponível em: http://www.learnovatecentre.org/wp- content/uploads/2013/06/Use_of_Serious_Games_in _the_Corporate_Sector_PRINT_FINAL. pdf. Acesso em: nov. 2013.

[92] Boyle, E.; Connolly, T.M.; Hainey, T. The role of psychology in understanding the impact of computer games. *Entertainment Computing*, 2(2):69-74, 2011.

[93] K.L. Ratwani. *Game-Based Training Effectiveness Evaluation in an Operational Setting.* Study Report 2010-02. 2010. Disponível em: http://www.dtic.mil/cgi- bin/GetTRDoc?AD =ADA530660. Acesso em: dez. 2013.