UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANÁLISE DO PROGRAMA "SCRATCH" PARA CRIAÇÃO DE OBJETOS DE APRENDIZAGEM COM INTEGRAÇÃO DE MÍDIAS

TRABALHO DE GRADUAÇÃO

Bruno Giacomelli Ilgenfritz

Santa Maria, Rio Grande do Sul, Brasil
2010

ANÁLISE DO PROGRAMA “SCRATCH” PARA CRIAÇÃO DE OBJETOS DE APRENDIZAGEM COM INTEGRAÇÃO DE MÍDIAS

por

Bruno Giacomelli Ilgenfritz

Monografia apresentada ao Curso de Ciência da Computação,
da Universidade Federal de Santa Maria (UFSM, RS),
como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Profa. Dra. Roseclea Duarte Medina

Trabalho de Graduação N. 280

Santa Maria, RS, Brasil

2010

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova a Monografia

ANÁLISE DO PROGRAMA "SCRATCH" PARA CRIAÇÃO DE OBJETOS DE APRENDIZAGEM COM INTEGRAÇÃO DE MÍDIAS

elaborada por
Bruno Giacomelli Ilgenfritz

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Roseclea Duarte Medina, Prof. Dra.
(Presidente/Orientador)

Oni Reasilvia de Oliveira Sichonany, Profa. (UFSM)

Giliane Bernardi, Profa. Dra. (UFSM)

Santa Maria, julho de 2010.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

ANÁLISE DO PROGRAMA "SCRATCH" PARA CRIAÇÃO DE OBJETOS DE APRENDIZAGEM COM INTEGRAÇÃO DE MÍDIAS

Autor: Bruno Giacomelli Ilgenfritz
Orientador: Profa. Roseclea Duarte Medina
Local e data da defesa: Santa Maria, 07 de julho de 2010.

Scratch é um projeto *open-source* do grupo *Lifelong Kindergarten*, do Instituto de Tecnologia de Massachusetts (MIT), onde os autores desenvolveram uma linguagem de programação voltada para crianças e jovens. Contando com um ambiente integrado de desenvolvimento (IDE) próprio, essa plataforma permite a construção visual da lógica de programação, tornando o processo de desenvolvimento de aplicações interativas mais acessível a diversas faixas etárias ou níveis de fluência computacional. A linguagem e sua IDE são desenvolvidas sobre a linguagem de programação *Squeak*, uma modificação *open-souce* da linguagem *Smalltalk-80* que conta com um suporte diversificado a formatos multimídia. Neste trabalho, avalia-se a capacidade da IDE *Scratch* em permitir o desenvolvimento de Objetos de Aprendizagem interativos, procurando ao mesmo tempo expandir a plataforma para suportar mídias de vídeo.

Palavras-chave: *Scratch, Squeak, Smalltalk-80*; Mídias, MIT, Objetos de Aprendizagem; Plataforma, Programação; Educação.

ABSTRACT

Trabalho de Graduação
Course of Computer Science
Universidade Federal de Santa Maria

ANALYSIS OF "SCRATCH" PROGRAM FOR CREATION OF LEARNING OBJECTS WITH MEDIA INTEGRATION

Author: Bruno Giacomelli Ilgenfritz
Advisor: Profa. Roseclea Duarte Medina

Scratch is an open-source Project from *Lifelong Kindergarten* group, at Massachusetts Institute of Technology (MIT), where the authors developed a programming language intended to children and youth. Featuring an integrated development environment (IDE), this platform allows the construction of visual programming logic, making the process of developing interactive applications more accessible to different age groups or levels of computational fluency. The language and its IDE are carried on the programming language *Squeak*, a open-source modification of Smalltalk-80 language that has a diversified support to multimedia formats. In this paper, the capacity of the *Scratch* IDE is evaluated to allow the development of interactive learning objects, while seeking to expand the platform to support video media.

Keywords: *Scratch, Squeak, Smalltalk-80;* Media, MIT, Learning Objects; Platform, Programmation; Education.

SUMÁRIO

1 INTRODUÇÃO

Com a revolução das tecnologias da informação e comunicação (TICs), vive-se hoje em uma sociedade em constante mudança. Suas características trazem para uma realidade cada vez mais conectada em rede, com uma economia global e voltada para a informação e para uma nova cultura: a da virtualidade. As TICs vão gradativamente influenciando o dia-a-dia e modificando a forma como as pessoas se comunicam, trabalham, se relacionam, aprendem e ensinam (MORAN, 2009).

A educação não permanece alheia a esse contexto. O desenvolvimento científico-tecnológico tem um impacto direto nas instituições de ensino, já que é por meio delas que diversas gerações de jovens tomam conhecimento de novas formas de pensamento, administração, atuação e comunicação, habituando-se às novas ferramentas e aos novos paradigmas (CASTELLS, 2003 apud MORAN, 2009). A Internet, por exemplo, potencializou desde o seu surgimento a incorporação de tecnologias da informação e comunicação em diversas esferas da atividade universitária, sejam elas culturais, administrativas, acadêmicas ou científicas. (MORAN, op. cit.). Ainda segundo o mesmo autor, no Brasil, a influência da tecnologia se faz presente no dia-a-dia das escolas, mesmo que não estejam incorporadas ao ensino e à aprendizagem. Os alunos constantemente trazem para a escola questões que dizem respeito ao mundo interconectado por meio das tecnologias e mídias, fazendo com que os professores se sintam desafiados.

Existem diversos meios para que os professores trabalhem com seus alunos conteúdos pedagógicos ao mesmo tempo em que utilizam os recursos tecnológicos ou de mídia disponíveis. Alguns exemplos desses recursos no âmbito da tecnologia de computação são vídeos, redes (e.g. fóruns, email), hipertexto ou conteúdos iterativos, como jogos. Um desafio a ser enfrentado, entretanto, é permitir que o próprio educador produza conteúdo educacional personalizado e interativo de uma forma que a criação não seja demasiada difícil ou limitada pelas habilidades computacionais dele.

Um determinado gênero de *software*, chamado de “Sistema de Autoria” (ou “Ferramenta de Autoria”), procura resolver essa necessidade permitindo que pessoas não-habituadas ao uso da tecnologia ou de mídias possam criar seus próprios recursos sem a necessidade de aprender a programar. A seguinte definição elucida o objetivo desse tipo de ferramenta de maneira geral:

> Um sistema de autoria é um programa ou conjunto de programas os quais permitem um instrutor ou desenvolvedor instrucional (i.e. o autor) criar programas de estudo sem a necessidade de programar. O autor especifica o conteúdo a ser ensinado e, usualmente, a lógica de instruções ou estratégia a ser usada; o sistema de autoria automaticamente gera o código sem erros que corresponde a essas especificações. (KEARSLEY, 1982, p. 1)

Mesmo existindo muitas ferramentas de autoria, por vezes os educadores não conseguem ou não são capazes de criar bons materiais. Um exemplo é inabilidade de criação. Para ilustrar isso, Savi (2008) menciona a questão dos jogos educativos, em que empresas especialistas criam produtos educacionais atraentes e divertidos, porém falhos no aspecto de aprendizado. Em contrapartida, os jogos desenvolvidos por educadores com viés mais acadêmico não conseguem atrair a atenção dos alunos, uma vez que seus criadores têm pouco conhecimento da arte, ciência e cultura de projetos de jogos.

Além disso, Ellington (1998. apud BALASUBRANIAN, 2006) comenta que muitas vezes existe receio entre os educadores em usar a tecnologia em sala de aula, o que dificulta uma adoção mais abrangente de tais ferramentas para criação de recursos pedagógicos. Exemplos dessas preocupações são os alunos não participarem ou cooperarem com as atividades propostas ou então que a execução e tutoria da atividade exponham a inabilidade técnica dos professores, tanto em criar materiais como em usar as ferramentas, perante alunos que possuam mais conhecimento ou mais prática com a tecnologia que está sendo utilizada. Duffin (2004 apud CUBAN, 2001) lista algumas questões práticas para os professores ao decidir em usar ou não usar em atividades com computadores em suas aulas:

- A máquina ou *software* é simples o suficiente para o professor aprender?
- O *software* é versátil? Pode ser usado em mais de uma situação?
- O *software* vai motivar os estudantes?
- O *software* desenvolve habilidades conectadas com o que eu quero ensinar?
- A máquina ou *software* são confiáveis?
- Se o sistema falhar, vai haver alguém para consertá-lo?
- O tempo que o professor precisa investir em aprender a usar o sistema compensa em relação ao aprendizado do aluno?
- O uso dos computadores pelos estudantes vai enfraquecer minha autoridade?

Dessa forma, existe uma necessidade de conectar o desenvolvimento da capacidade de autoria dos professores com uma ferramenta sólida o suficiente para responder essas questões.

Além disso, a ferramenta deve alinhar o objetivo de ensino do professor com o interesse dos alunos por conteúdo relevante, de acordo com as necessidades de ambos e com o contexto tecnológico no qual estão vivendo. Em um exemplo de contexto vivido pelos estudantes, Gros (2003. apud SAVI, 2008) cita o exemplo dos jogos, que é "uma das principais formas de acesso ao mundo da tecnologia para crianças e jovens, pois geralmente o primeiro contato com equipamentos eletrônicos acontece por meio de um vídeo game". Porém, acrescenta a seguinte ressalva:

> Para serem utilizados como instrumentos educacionais os jogos devem conter algumas características específicas para atender as necessidades vinculadas à aprendizagem. Por isso os softwares educacionais, entre eles os jogos, "devem possuir objetivos pedagógicos e sua utilização deve estar inserida em um contexto e em uma situação de ensino baseados em uma metodologia que oriente o processo, através da interação, da motivação e da descoberta, facilitando a aprendizagem de um conteúdo". (PRIETO et al., 2005 apud SAVI, 2005)

Com o professor como o principal responsável pelo projeto pedagógico junto ao interesse dos alunos pela tecnologia e pelas mídias, faz-se interessante ter o professor junto à construção de objetos educacionais. Assim, é coerente facilitar a ele a autoria de seu próprio material educacional interativo.

A categoria de ferramentas que auxiliam as pessoas não-aptas em programação a criar seu próprio conteúdo interativo são as Ferramentas de Autoria. Ainda que existam diversas soluções de autoria disponíveis, ainda são encontrados problemas a serem resolvidos quanto ao desempenho do professor em executar projetos que atinjam às expectativas dos alunos.

Scratch é um programa que tem como objetivo o desenvolvimento de habilidades relacionadas à computação, tal como a lógica de programação, em uma ferramenta que permite aos seus usuários explorarem, experimentarem e criarem aplicações próprias por meio de uma interface amigável e de fácil aprendizado (FILDES, 2007). Seu público-alvo são principalmente as crianças e os jovens, que compõem a grande maioria dos usuários do programa e da comunidade virtual que existe em torno dele. Essa comunidade (http://scratch.mit.edu) agrupa todos os projetos que os chamados *Scratchers* (isto é, usuários do programa) compartilham com outros usuários do mundo todo. Resnick (2009) relata que em média são enviados 1.500 novos projetos todos os dias para a comunidade *online* do projeto. Além disso, segundo o autor, "a coleção de projetos do site é incrivelmente diversa, incluindo jogos, notícias interativas, simulações de ciência, *tours* virtuais, cartões de aniversário, concursos animados de dança, tutoriais interativos, etc., todos programados no *Scratch*" (RESNICK, op. cit., pg. 4).

Logo, com a aplicação sendo fonte de gama de projetos diversificados e contando com uma interface voltada à facilidade de aprendizado por crianças e jovens, tem-se uma aplicação potencial a suprir a necessidade dos educadores em criar materiais educacionais, se tornando mais interessante por ter relação próxima com a criação de aplicações midiáticas e interativas.

Com esse contexto em vista, dá-se o objetivo desse trabalho: expandir a ferramenta de autoria *Scratch* para suportar o uso de mídias de vídeo por meio da programação na linguagem *Squeak*, permitindo que a ferramenta esteja mais perto de suprir as necessidades de criação de materiais educacionais interativos usando mídias integradas.

A disposição do trabalho, então, apresenta-se na seguinte forma: o capítulo 2 comenta sobre Ferramentas de Autoria e discorre sobre exemplos de diferentes sistemas nos quais pessoas leigas em programação podem criar conteúdo interativo, citando aspectos positivos e negativos deles. No capítulo 3, disserta-se sobre o *Scratch*, a ferramenta desenvolvida pelo grupo *Lifelong Kindergarten* (2007), do MIT (*Massachusetts Institute of Technology*), comentando suas principais funcionalidades, aspectos de projeto e de interface, além da relação entre o *Scratch* e a educação. O capítulo 4, por fim, trata o suporte a mídias do *Scratch* e as modificações e acréscimos técnicos necessários de funcionalidades na ferramenta para suporte a vídeos, buscando melhorar a integração de mídias.

2 FERRAMENTAS DE AUTORIA

2.1 Conceito

Ferramentas de autoria, ou sistemas de autoria, se referem a sistemas baseados em computação que permitem a um grupo genérico (incluindo não-programadores) criar (i.e., autor) conteúdo para sistemas de tutoria inteligentes, sistemas especialistas que auxiliam o processo de tutoria dos alunos em um ambiente virtual de aprendizagem. No desenvolvimento de *softwares* para educação, um sistema de autoria é um programa que permite não-programadores criar conteúdo educacional facilmente com características de programação. Essas características de programação são integradas ou ocultas atrás de botões ou outras ferramentas e, dessa forma, o autor não possui a necessidade de saber como programar (AGARWAL, 2009). Um sistema de autoria possui por trás uma linguagem chamada Linguagem de Autoria. Essa linguagem é a base na qual reside o sistema de autoria, sendo através dela que se transcrevem os comandos em alto nível, definidos pelo usuário em comandos da linguagem de programação que foi utilizada no desenvolvimento do sistema para a criação de conteúdo interativo.

Diversas ferramentas podem ser chamadas de ferramentas de autoria, tais como editores para Web, Flash ou PowerPoint. Entretanto, somente um pequeno grupo de programas inclui suporte para padrões de conteúdo de *e-learning*, isto é, conteúdo próprio para ser criado e distribuído por meios eletrônicos especializados em aprendizagem. Exemplos de ferramentas de autoria são: Macromedia Authorware, ToolBook, NetObjects Fusion, CourseBuilder, Lectora, entre outros (ADOBE, 2010; SUMTOTAL, 2010; NETOBJECTS, 2010; COURSEBUILDER, 2010; TRIVANTIS, 2010).

2.2 Integração de Mídias

Uma característica muito importante que envolve qualquer ferramenta ou sistema de autoria para educação é o uso de mídias. “Mídia” pode ter diversos significados e se referencia a um complexo sistema de expressão de comunicação, sendo que, na atualidade, “mídias” é uma terminologia que indica “suporte de difusão e veiculação de informação para gerar informação” (e.g. rádio, televisão, jornal, máquina fotográfica, filmadora). Ela é organizada pela maneira como uma informação é transformada e disseminada. A palavra

escrita, o discurso, o som, a imagem estática ou em movimento formam o substrato da mídia (MORAN, 2009).

Dizard (1998) cita que a tecnologia transformou a forma de disseminar a informação em três principais momentos: primeiro, na introdução das impressoras a vapor para produção de jornais, livros e revistas em grande escala no séc. XIX; segundo, na vinda da transmissão por ondas eletromagnéticas resultando no rádio e na televisão nas décadas de 20 e 30 do séc. XX; por último, na produção, armazenagem e distribuição de informação e entretenimento estruturado pela revolução da informática no fim do séc. XX.

Nascida dentro dessa última conjuntura, a sigla TIC (Tecnologia de Informação e Comunicação), especificamente, envolve a aquisição, o armazenamento, o processamento e a distribuição da informação por meios eletrônicos e digitais, como rádio, televisão, telefone e computadores, entre outros. Ela resultou da fusão das tecnologias de informação, antes referenciadas como informática, e das tecnologias de comunicação, relativas às telecomunicações e à mídia eletrônica (MORAN, 2009). Quatro conceitos dão suporte para as TICs no novo contexto tecnológico trazido pelos computadores. Esses conceitos são descritos na logo abaixo (Tabela 2.1).

Tabela 2.1: conceitos relacionados às TICs. Adaptado de (MORAN, 2009)

Multimídia	Capacidade de um computador ou de um programa de usar elementos de várias mídias, como áudio, vídeo, ilustração, animação e texto. Também se refere às produções que articulam diversas mídias de maneira informatizada e com participação interativa dos seus usuários. Exemplos: almanaques ilustrados, jogos.
Hipertexto	Sistema para a visualização de informação cujos documentos contêm referências internas para outros documentos (chamadas de *hyperlinks* ou *links*) para a fácil publicação, atualização e pesquisa de informação. Exemplos: World Wide Web.
Telemática	Conjunto de tecnologias da informação e da comunicação resultante da junção entre os recursos de telecomunicações (telefonia, satélite, cabo, fibra ótica, etc.) e da informática (computadores, periféricos, *softwares*, sistemas de redes) que possibilita o processamento, armazenamento e comunicação de grandes quantidades de dados (texto, imagem, som, etc.) em curto prazo a qualquer distância do planeta.
Hipermídia	União dos conceitos de hipertexto, interface, multimídia e não-linearidade em uma

	única linguagem. O usuário pode iniciar uma leitura não-linear, escolhendo o início, meio e fim. Segundo Moran (2009 apud BUGAY, 2000), a hipermídia pode ser considerada uma extensão do hipertexto que inclui sons, animações e vídeos ao texto comum e de formas interativas, como com o clicar de um botão.
Hipermídia Adaptativa	Capacidade de um sistema TIC em facilitar para o seu usuário a acessibilidade da informação, selecionando e organizando o conteúdo e as conexões entre os conteúdos da forma mais adequada de acordo com o perfil do usuário ou com o contexto onde ele se encontra. O conceito também pode se estender para definir o conteúdo e suas conexões de acordo com o dispositivo sendo usado, por exemplo, conteúdo para computadores *desktop* ou dispositivos móveis.

As TICs revolucionaram a relação da sociedade com a informação trazendo, com isso, o desafio para a educação em o que fazer com essa informação e como orientar o aluno a usar a informação com autonomia. Compreender as diferentes formas de representação de comunicação disponíveis nas escolas e estabelecer o diálogo com as diferentes formas de mídia são necessidades no contexto mostrado (MORAN, 2009). A integração de mídias trata justamente disso, ao propor unir os diferentes tipos de mídias como texto, áudio e imagens ao processo de ensino-aprendizado.

Moran mostra ainda a forma como a integração das mídias atua no ensino e comenta a importância do objetivo da mídia como recurso pedagógico:

> A mídia impressa, a televisão, o vídeo, o rádio, a Internet, a hipermídia são ótimos recursos para mobilizar os alunos em torno de problemáticas, quando se intenta despertar-lhes o interesse para iniciar estudos temáticos, desenvolverem projetos ou trazer novos olhares para os trabalhos em andamento. Para tanto, é importante estabelecer quais os objetivos pedagógicos das atividades e quais as características principais das mídias disponíveis. Nesse último aspecto, os alunos são excelentes parceiros dos professores. (MORAN, op. cit.)

Concluindo com a afirmação que “uma mudança significativa na qualidade do ensino é atingida quando se consegue integrar o máximo possível de formas de mídia nesse processo, sejam elas telemáticas, audiovisuais, textuais, orais, musicais, etc.”.

2.3 Exemplos de Ferramentas de Autoria

2.3.1 ToolBook

ToolBook (SUMTOTAL SYSTEMS, 2010) é uma ferramenta de autoria desenvolvida pela empresa SumTotal Systems. Ela habilita profissionais e especialistas em ensino a criarem conteúdos de aprendizagem interativa como testes, avaliações e simulações de *software*.

O conteúdo criado pelo programa é exportado e distribuído em formato HTML, permitindo-o ser incorporado em diversos Sistemas de Gerenciamento de Aprendizado além de possuir compatibilidade com dispositivos móveis. O conteúdo pode ser criado com o uso de modelos prontos (*templates*), como páginas e estilos. Outra característica é a possibilidade de converter apresentações de PowerPoint em cursos de e-*learning*.

Outros exemplos de recursos da ferramenta, incluídos em sua última versão disponível (versão 10), são: melhor manipulação de mídias de imagem para produzir conteúdo *Web*; exercícios de ligar elementos com novo desenho (Figura 1); adição de som ao conteúdo, incluindo gravação de voz e recursos de controle de som (i.e. *play*, *stop*, *pause*, *restart*); importação de apresentações do *PowerPoint*; modificações na interface, bem como compatibilidade expandida para navegadores e plataformas *mobile*, com suporte para *Internet Explorer 8* e *Google Android*; (SUMTOTAL SYSTEMS, 2009).

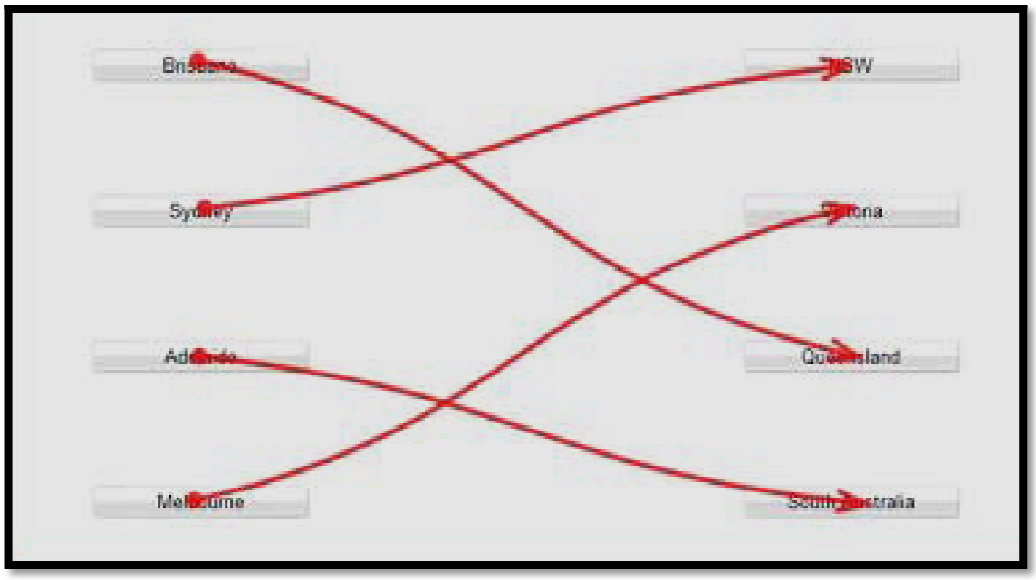

Figura 1: um exemplo de exercício de ligação de elementos montado usando a ferramenta de autoria *ToolBook*. (SUMTOTAL SYSTEMS, 2009).

Todavia, a análise de Darymple e Roy (2010) cita que o *ToolBook* “não é uma escolha adequada para o desenvolvimento rápido de materiais de E-Learning. A ferramenta requer muito treinamento para se alcançar velocidade no desenvolvimento”. Mais além, os autores comentam que para a necessidade do desenvolvimento rápido de cursos, há melhores escolhas. Outros aspectos negativos são citados no estudo, como alguns usuários comentarem a necessidade de programação, o programa não ser intuitivo, ter tendência a dar falhas durante seu uso, entre outras. (DARYMPLE, 2010).

Outro ponto negativo do *software* é a sua distribuição não-gratuita. O preço é disponibilizado mediante consulta à empresa desenvolvedora. Darymple e Roy (op. cit.) comentam que o custo do programa é muito alto, comparado a outros produtos similares. Esse fator inviabiliza a adesão e distribuição entre instituições ou redes de ensinos que possuem capital limitado para investimento em um sistema desse gênero, por exemplo.

2.3.2 HotPotatoes

HotPotatoes é uma suíte de aplicações de autoria gratuita, de código fechado, que permite aos usuários criar exercícios interativos para a Web. O conjunto inclui seis aplicações, cada uma delas relacionada a um tipo específico de exercício, que são: múltipla-escolha, resposta simples, ordenação de sentenças, palavras-cruzadas, exercícios de ordenação e ligação e de preenchimento de lacunas em textos. O programa é desenvolvido pela organização *Half-Baked Software Inc* e está atualmente em sua versão 6.3 de outubro de 2009 (HALF-BAKED SOFTWARE INC, 2009).

Arneil e Holmes (1999), proprietários da ferramenta, falam que um dos princípios que guiaram o desenvolvimento do *HotPotatoes* foi oferecer diferentes estilos de questionários e que respostas escritas pelos estudantes tivessem flexibilidade do *software* em aceitar pequenas variações, principalmente no caso de diferente capitalização de letras entre uma resposta e outra. Com essa escolha de projeto, os autores dizem querer “proporcionar a oportunidade dos educadores criarem exercícios no qual os estudantes podem efetivamente aprender (...) com flexibilidade suficiente para permitir certo número de variações na resposta.”. Uma importante escolha de desenvolvimento dos autores a ser citada é objetivo da plataforma em ser uma ferramenta de autoria qbue qualquer pessoa com habilidades básicas de uso de computadores possam efetivamente usá-la. Outras particularidades e decisões de design na construção da ferramenta podem ser encontradas em Arneil e Holmes (1999).

Um dos pontos negativos da suíte *HotPotatoes* para autoria é a limitação exclusiva a exercícios. Não é possível desenvolver conteúdo rico em mídia, com a inclusão de vídeos e som sem a edição do código gerado pelo programa. Apesar da possibilidade, essa solução se contrapõe à proposta de ser uma ferramenta acessível para leigos em programação ou pessoas menos habituadas ao uso dos computadores. Além disso, mesmo em sua versão seis de 2009, a interface é antiga e não pode ser personalidade facilmente (i.e. não há uma interface completa para essa demanda) e não há suporte para outras línguas a não ser o inglês. Abaixo, um exemplo de exercício gerado pela ferramenta (Figura 2).

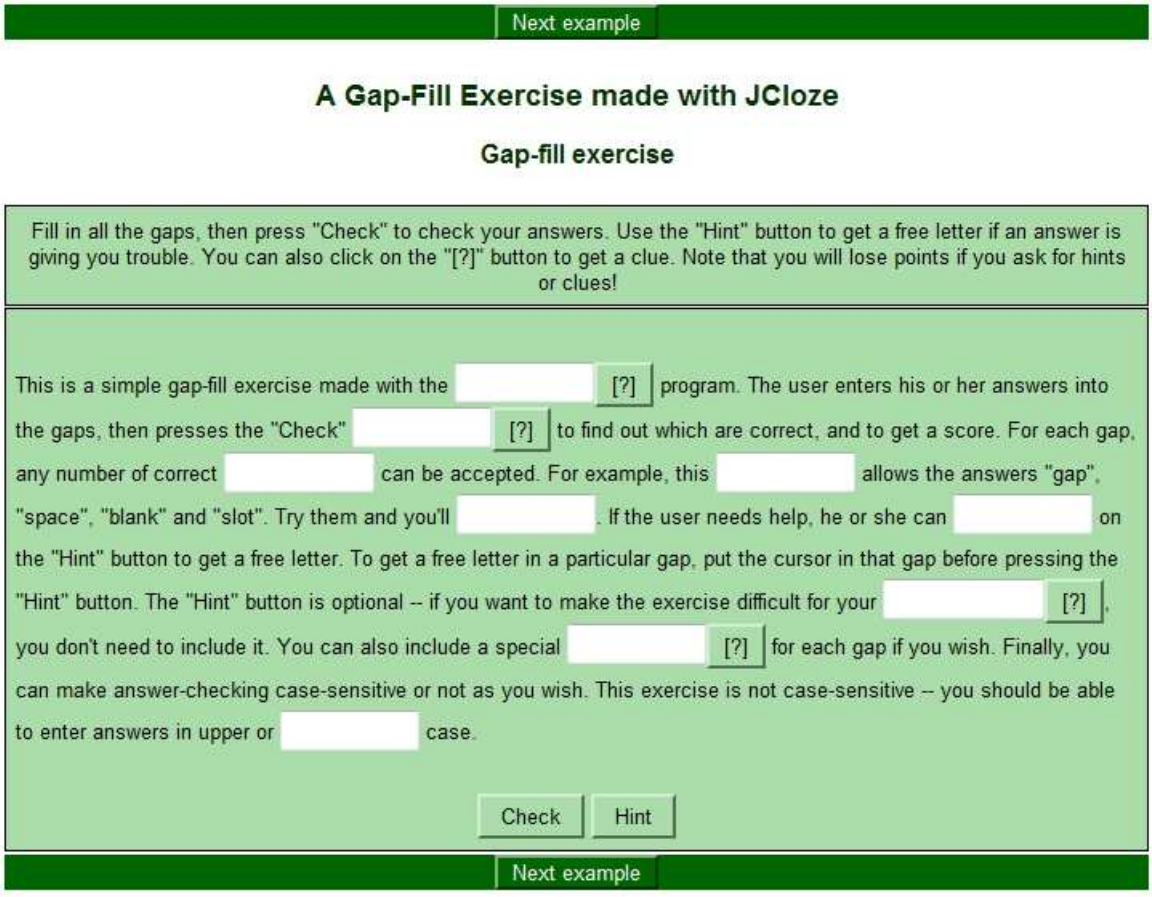

Figura 2: um exemplo de exercício de preenchimento de lacunas gerado pelo *JCloze*, uma das ferramentas da suíte *HotPotatoes* (HALF-BAKED SOFTWARE INC, 2009)

2.3.3 CourseBuilder

CourseBuilder (COURSEBUILDER, 2010) é uma ferramenta de autoria para *E-Learning* e, ao mesmo tempo, um Sistema de Gerenciamento de Aprendizado (LMS -

Learning Management System). Esse sistema busca oferecer rápida criação e implantação de cursos *online* e o desenvolvimento de conteúdo interativo. Por ser também um LMS, o *CourseBuilder* possui ferramentas gerenciamento de estudantes e relatórios de aprendizado.

Algumas características positivas a serem destacadas, listadas em CourseBuilder (op. cit.) são: possibilidade de *upload* de mídias, como imagens, vídeo e áudio; desenvolvimento de testes e exames com variados modelos de questões; agendamento de publicação dos cursos; os cursos podem ser criados, implantados e modificados de forma instantânea (i.e. por ser *online*); customização do visual da ferramenta de acordo com a organização; pré-requisitos para controlar o avanço dos alunos nos cursos; acompanhamento de competências; hospedagem e segurança do sistema incluída.

A plataforma possui uma interface agradável e moderna, com a edição contendo editores WYSIWYG (*What You See Is What You Get)*, forma de edição onde o que é editado no campo de edição texto é o que se obtém como resultado final, facilitando de forma geral a formatação do conteúdo. Os relatórios (e.g. o relatório de aprendizado de cada estudante) são bastante visuais e retornam informações relevantes aos educadores. A identidade visual também pode ser alterada de acordo com a organização contratante do serviço. Abaixo, um exemplo de apresentação da interface dessa solução de *E-Learning* (Figura 3).

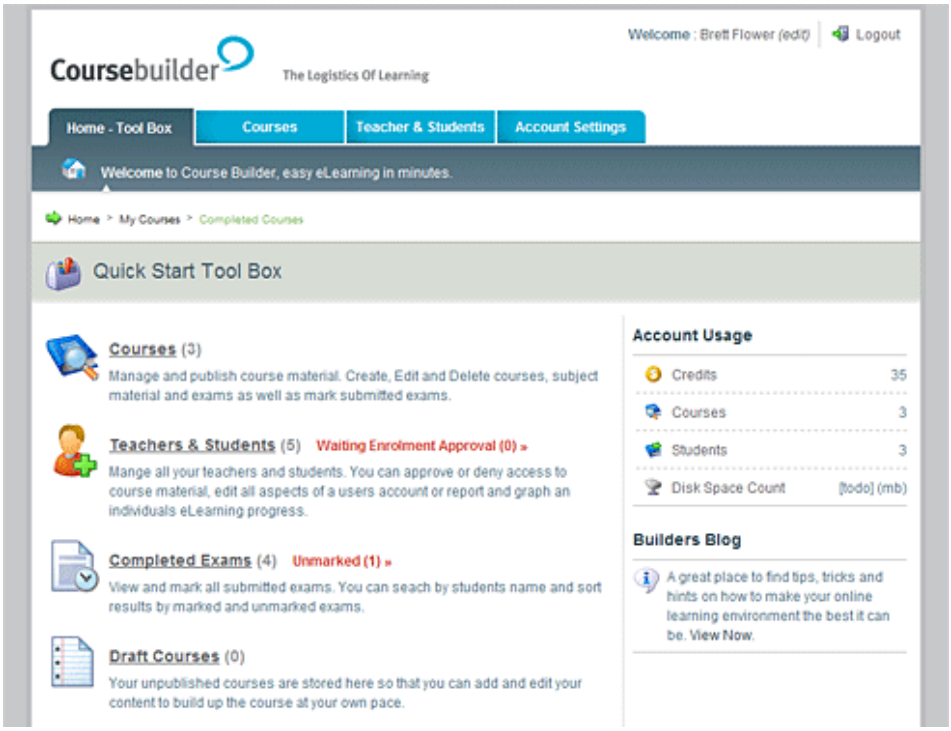

Figura 3: tela inicial da ferramenta *CourseBuilder* mostrando sua interface e visual. (COURSEBUILDER, 2010)

Porém, existem outros aspectos a serem levados em conta. Primeiro, o *CourseBuilder* é uma solução tendo como público-alvo principalmente o meio corporativo, como empresas e organizações. Seu conteúdo de autoria é majoritariamente texto, entremeado por vídeos e áudio embutidos, e sua interação se faz principalmente pelos modelos de questões as quais os alunos devem responder, ou seja, não há grande possibilidade de personalização da forma de interatividade. Além disso, essa ferramenta de autoria é paga, com um custo dependente do número de alunos envolvidos e a quantidade de cursos desenvolvidos.

2.3.4 NetObjects Fusion

NetObjects Fusion é uma ferramenta de *web design*, atualmente em sua versão 11, desenvolvida e distribuída pela companhia *NetObjects, Inc*. Possuindo uma interface gráfica para o usuário, o programa gera HTML ou XHTML de acordo com a modelagem feita pelo usuário do conteúdo (NETOBJECTS, 2010). Tal funcionalidade, assim como outras ferramentas do mesmo gênero (e.g. *Dreamweaver*, *Frontpage*), permite a criação de hipertexto para uso em *E-Learning*.

O programa conta com assistência para usuários novatos, além de modelos pré-prontos. Usuários mais avançados podem usar as diferentes ferramentas que o programa oferece para criar aplicações *online* mais sofisticadas. Além disso, uma suíte de componentes prontos permite a construção de conteúdo sem a necessidade de saber programação ou de construção de rotinas e, também, possui integrado diversas ferramentas úteis, como edição de imagem, FTP, banco de dados, editores CSS e HTML, controle de versões, animação, gerenciamento de tarefas, backup, entre outros. A interface com o usuário possui edição WYSIWYG (*What You See Is What You Get*), possibilitando inserir texto, gráficos e multimídia nas páginas sem grande dificuldade. (NETOBJECTS, op. cit.)

Um viés para o uso em educação é o fato de o programa ter como público usuários com conhecimentos prévios de *design*, jornalismo e autoria, com o programa provendo uma interface à qual esse tipo de profissional está acostumado (WILL-HARRIS, 2006). Outro ponto negativo para seu investimento em educação é a ferramenta ser proprietária, possuindo um custo de aquisição da licença (NETOBJECTS, op.cit).

3 A PLATAFORMA “SCRATCH”

3.1 Histórico

Scratch é a denominação para a plataforma gratuita de desenvolvimento de aplicações criada pelo grupo de pesquisa *Lifelong Kindergarten*, do Laboratório de Mídias no Instituto de Tecnologia de Massachusetts (MIT) - Estados Unidos. Essa plataforma de código aberto, lançada em maio de 2007, funciona como um ambiente de programação que tem como público principal usuários sem conhecimento de programação. A ferramenta permite a eles criar aplicações (programas) interativas e usando mídias, como som e imagem. Resnick (2009) cita que “*Scratch* tem sido chamado de *YouTube* de mídias interativas. A cada dia, *Scratchers* de todo o mundo enviam mais de 1.500 novos projetos para o site, com o código fonte de seus programas disponíveis livremente para serem compartilhados e reusados”. A figura 4 exibe uma das telas de um exemplo de projeto construído com o *Scratch*, de autoria do usuário “*Andrew123456789*”.

Figura 4: exemplo de um projeto interativo construído utilizando a ferramenta *Scratch* (LIFELONG, 2007).

No passado, iniciativas de inserir a programação no ambiente escolar começaram com entusiasmo, mas logo as escolas modificaram o contato com os computadores para outros usos, fazendo com que as máquinas fizessem parte constante da vida das crianças, mas com poucas aprendendo a programar. Alguns fatores contribuíram para isso ocorrer:

> Primeiro, as linguagens de programação eram muito difíceis de usar, fazendo com que muitas crianças não conseguissem aprender a sintaxe de programação; segundo, a programação era freqüentemente introduzida com atividades desconectadas com os interesses e experiências das pessoas (e.g. gerar listas de números primos); e terceiro, a programação era introduzida em contextos onde ninguém provia ajuda quando as coisas davam erradas ou encorajavam explorações mais profundas caso dessem certo. (RESNICK, 2009, p. 4)

Papert (1980, apud RESNICK, 2009, p. 4) argumenta que as linguagens de programação devem: ser fáceis para começar a trabalhar com elas (*low floor*); ter oportunidades de criação cada vez mais complexa ao longo do tempo (*high ceiling*); suportarem diversos tipos diferentes de projetos para que pessoas com diferentes interesses e estilos de aprendizado se engajem (*wide walls*). Segundo Papert (op. cit.), oferecer esses três aspectos em uma linguagem de programação não tem sido fácil.

3.2 Projeto

Para abordar melhor esses fatores que dificultam a introdução da programação às crianças, três princípios guiaram os autores no desenvolvimento do *Scratch*: ter uma utilização mais flexível (*tinkerable*), ter mais significado para o público (*meaningful*) e ser mais social que outros ambientes de programação.

O primeiro aspecto, ser mais flexível de usar, se traduz em um *design* inspirado na forma como as crianças brincam e criam com blocos de montar, como LEGO©. Os autores buscaram reinventar a forma de criação onde as crianças começam a juntar os blocos e, tão logo as estruturas começam a aparecer, novas idéias surgem, fazendo com que planos e objetivos se desenvolvam organicamente junto com as estruturas e histórias. Isso se traduz no *Scratch* como uma gramática baseada em uma coleção de blocos de programação gráficos, onde é possível juntá-los para criar programas. Não há nenhuma sintaxe obscura ou pontuação como nas linguagens tradicionais de programação. Os blocos do *Scratch* também têm uma

forma conveniente para facilitar a criação de rotinas, como laços de execução terem o formato de “C”, com os blocos a serem repetidos aninhados dentro deles (Figura 5).

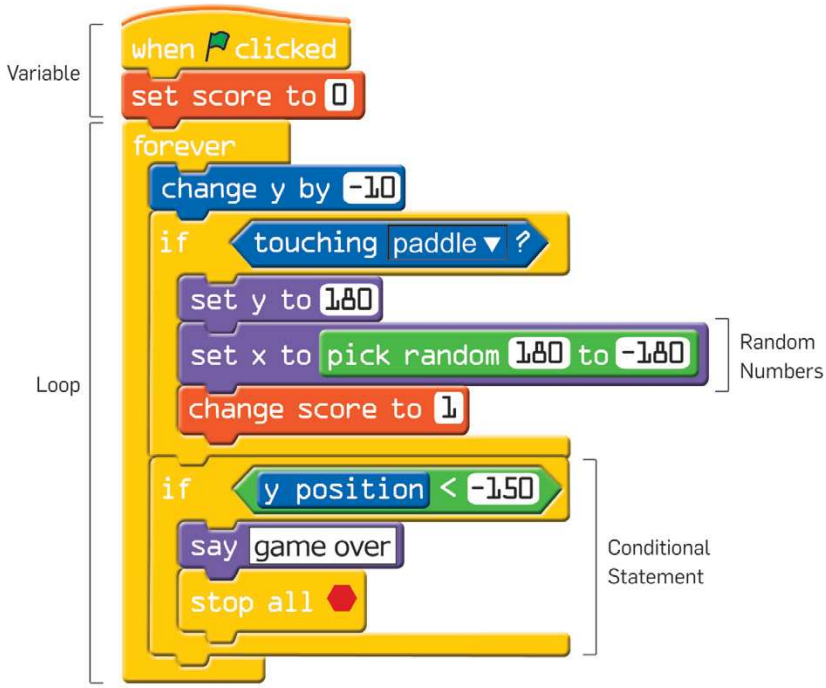

Figura 5: exemplo de rotina no *Scratch* mostrando conceitos computacionais e matemáticos. (RESNICK, 2009)

Resnick (2009) cita que alguns outros aspectos do ambiente trazem ainda mais facilidade de uso. Por exemplo, clicar em uma pilha de blocos faz com que eles executem imediatamente, sem compilação; é possível fazer modificações na pilha de blocos enquanto ela está executando e ver seu efeito; para montar processos paralelos somente é preciso criar duas ou mais pilhas de blocos que eles executarão em paralelo automaticamente. Além disso, a área de programação do *Scratch* procura ser como um *desktop* físico, não sendo necessário limpá-la de blocos desnecessários para o código executar. Ao contrário da maior parte das linguagens e cursos de programação que privilegiam o planejamento *top-down*, onde o projeto deve ser planejado como um todo primeiro, os autores buscam instigar para os usuários a experimentação *bottom-up*, onde a construção dos blocos iniciais de código instiga o usuário a criar e acrescentar progressivamente cada vez mais funcionalidades ao seu programa.

Para alcançar mais significado para o público, o *Scratch* se baseia em dois critérios: diversidade e personalização. O primeiro suportando diversos tipos de projetos (histórias,

jogos, animações, simulações, etc.), permitindo pessoas com interesses variados trabalhar em projetos os quais lhes convém, e personalização, facilitando o uso de mídias como fotos, música e criação de gráficos para que os projetos se tornem pessoais e trazendo à tona a necessidade da integração de mídias na experiência de aprendizado (MORAN, 2009).

Por último, para ter apelo social, uma comunidade *online* foi desenvolvida em torno do website (http://scratch.mit.edu/), onde os usuários podem compartilhar seus projetos e colaborar, criticar ou dar suporte ao projeto de outras pessoas. Na própria interface do programa existe um botão para a publicação de projetos, que rodam a partir do navegador. Com isso, Resnick (2009) aponta que em 27 meses após o lançamento do *Scratch*, 500.000 projetos foram compartilhados na comunidade. A grande audiência é uma fonte de motivação para as pessoas compartilharem seus projetos e, por meio disso, receberem *feedbacks* ou conselhos em contrapartida dos outros usuários.

Algumas outras características relacionadas ao *design* do projeto merecem destaque, tais como a manipulação de mídias, suporte a múltiplas línguas e integração com o mundo físico (RESNICK et al., 2003).

Relacionado ao “significado” para o público, o foco em manipulação de diferentes mídias permite que seus usuários se envolvam com projetos conectados mais fortemente com seus interesses. Entre as mídias suportadas estão os principais formatos de imagens e alguns formatos de áudio, porém atualmente não existe qualquer suporte a vídeos.

Para o suporte a múltiplas línguas, o *Scratch* foi desenvolvido com o propósito de permitir seus usuários trabalharem e pensarem na sua linguagem mais confortável, além de permitir que qualquer um possa ler e modificar código compartilhado na sua comunidade, mesmo que esse tenha sido feito do outro lado do mundo. Essa propriedade permite fazer do *Scratch* um ambiente multicultural, agregando pessoas de todo o mundo.

Um último conceito de *design* é ligado ao mundo físico, com o ambiente de programação sendo projetado para permitir a integração com objetos reais e programáveis, tais como motores, luzes ou sintetizadores MIDI (*Musical Instrument Digital Interface*). Atualmente, o programa conta com certas funções de programação de motores para o brinquedo LEGO© *Mindstorms*, funções essas provenientes na última versão do programa (versão 1.4), lançada no primeiro semestre de 2010.

3.3 Interface

A interface do programa possui 4 painéis principais. São eles: biblioteca de funções, biblioteca de *sprites*, painel de programação e mídias e o palco (Figura 6).

A biblioteca de funções contém todas as funções de programação do programa. Ela se subdivide em categorias para organizar e direcionar o usuário para funções relacionadas a um objetivo específico, como todas as operações de movimento ou todos os operadores de lógica relacional. Cada categoria tem a sua cor específica para facilitar a usabilidade. Nessa seção do ambiente também é possível declarar variáveis na categoria "Variáveis", que podem ser usadas como argumentos em funções específicas, como "Mova X passos", sendo 'X' o argumento esperado da função. Algo importante a notar é que novas funções não podem ser criadas pela interface, somente editando o código-fonte do programa.

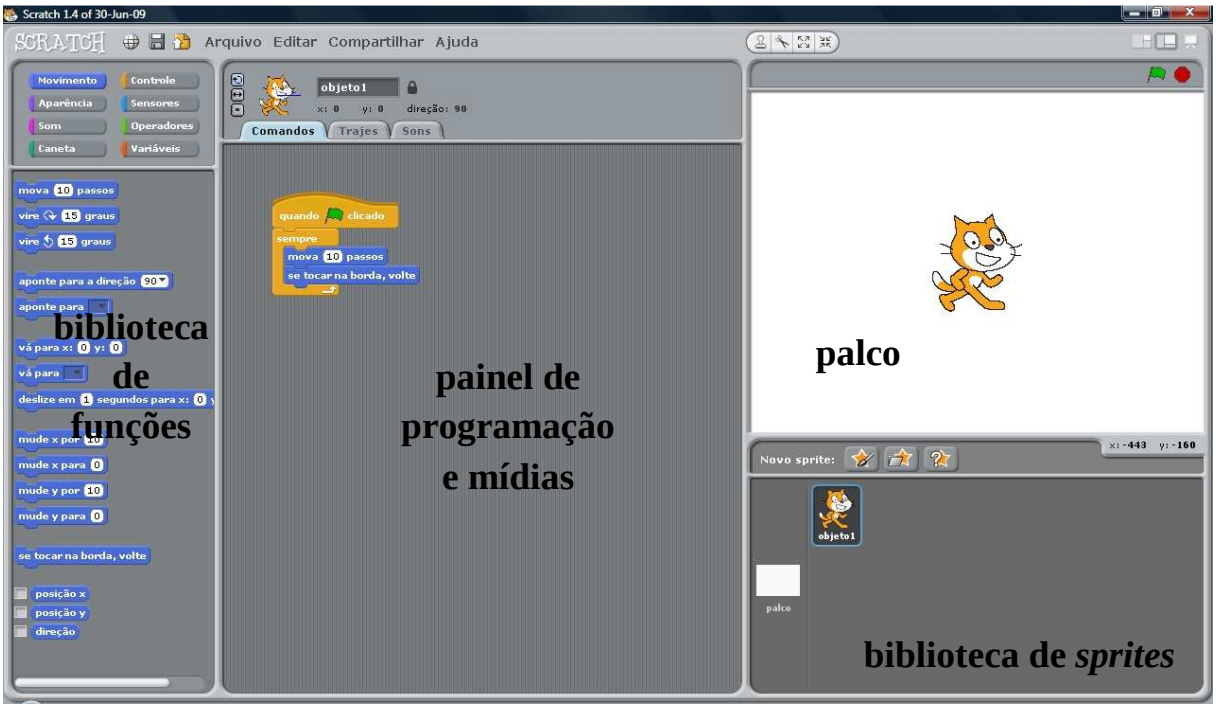

Figura 6: interface do *Scratch* com os principais painéis.

Na biblioteca de *sprites* se encontram todas as imagens que são ou podem ser usadas como objetos a serem manipulados pelas funções. Um operador "Mova 10 passos" age diretamente sobre uma dessas imagens, movendo-a na direção atual em 10 unidades de espaço (*pixels*). É possível importar imagens em formato *.jpeg*, *.png*, *.bmp* e *.gif*. No entanto, imagens *.gif* com vários *frames* não têm animação, ficando estáticas em um quadro. O palco é um *sprite* especial que atua como fundo da tela da aplicação, podendo ser modificada de acordo.

Por último, a biblioteca de *sprites* permite também que o próprio usuário desenhe imagens para serem usadas como objetos através de um editor de imagens embutido no *Scratch*.

O painel de programação e mídias tem esse nome por ser o local de manipulação das rotinas de programação e de manipulação das mídias, como imagens e áudio. Um *sprite* da biblioteca é programado nessa seção, tendo comandos que atuam sobre ele ou sobre sua "roupagem" (*costume*). Também pode ser determinado seu rótulo e estilo de rotação. Para cada objeto, incluindo o palco, é possível anexar recursos de áudio para serem executados através das funções de manipulação de som que existem na biblioteca, permitindo, assim, a obtenção de efeitos sonoros e musicais nas aplicações. Os formatos disponíveis de arquivo são *.aif*, *.aiff*, *.wav* e *.mp3*. Ao incluir um arquivo *.mp3*, o áudio é convertido em mono não-compactado. Abaixo, o painel com a aba 'Sons' selecionada (Figura 7).

Figura 7: painel de programação e mídias com a aba 'Sons' selecionada.

Em relação a mídias, uma falta é o suporte a vídeos ou animações. Nenhum formato de vídeo é suportado. Na aba "trajes" (*costumes*), onde são armazenadas todas as imagens de um determinado *sprite* para simular animações de personagens ou de *sprites*, há um recurso de acesso à *webcam* do computador. No entanto, com essa ferramenta, só é possível capturar fotos para serem usadas como roupagem. Não há forma de capturar vídeos. Adaptar o *Scratch* para permitir suporte a vídeos é um dos objetivos desse trabalho.

Por último, há o palco (*stage*). Ele representa o estado atual do programa e mostra o que será o resultado final da aplicação. Nele é possível ver o resultado da programação das imagens que compõem a aplicação do usuário e acompanhar o estado das variáveis. Existem botões anexados que aumentam o tamanho do palco ou para deixá-lo em formato tela cheia. Da biblioteca de *sprites* é possível adicionar as imagens ao palco.

3.4 "Scratch" e Educação

O *Scratch* foi concebido primariamente com o intuito de permitir aos seus usuários alcançarem uma maior fluência com as tecnologias de informação. Ou seja, irem além das atividades básicas relacionadas aos computadores (RESNICK et al., 2003). A audiência da ferramenta, incluindo juntas a plataforma de desenvolvimento e a comunidade, é composta principalmente de jovens com uma faixa de etária entre oito e 16 anos, além de um considerável grupo de adultos que também participa ativamente criando seus projetos pessoais ou mesmo comentando os projetos de outros usuários.

A proposta demonstra que, enquanto os usuários programam e compartilham seus projetos interativos, eles aprendem importantes conceitos matemáticos e computacionais, além de habilidades essenciais para o século 21, tais como pensar criativamente, raciocinar sistematicamente e trabalhar colaborativamente. Resnick (2009) comenta que o objetivo não é preparar pessoas para carreiras como programadores profissionais, mas sim nutrir uma nova geração de pensadores criativos e sistemáticos, confortáveis em usar a programação para expressar suas idéias.

Rusk, Resnick e Maloney (2008) sintetizam as nove competências do século XXI, identificadas pelo *Partnership for the 21st Century* (http://21stcenturyskills.org) no relatório *Learning for the 21st century*, e, divididas em três áreas, traçam um paralelo com o desenvolvimento delas através da plataforma *Scratch* (Tabela 3.1).

Tabela 3.1: competências do séc. XXI identificadas no relatório *Learning for the 21st century*. (RUSK et. al., 2008)

Área	**Competência**	**Desenvolvimento através do *Scratch***
Informação e Comunicação	Literácia para Informação e Mídia	Aprender a selecionar, criar e gerir múltiplas formas de mídias, tais como textos, imagens e registros de áudio. Com o tempo se tornam perspicazes e críticos com a

		mídia que observam a sua volta.
	Comunicação	Escolha, manipulação e integração de uma grande variedade de mídias para se expressarem de forma criativa e persuasiva, além da capacidade de ler e escrever.
Raciocínio e Resolução de Problemas	Raciocínio Crítico e Pensamento Sistêmico	Para construir projetos, é necessário coordenar o tempo e interação entre diversos objetos na tela. Essa capacidade de programar proporciona uma experiência direta de detecção, retroalimentação e outros conceitos fundamentais sobre sistemas.
	Identificação, Formulação e Resolução de Problemas	Criar um projeto requer que se pense numa idéia, para depois dividir o problema em passos menores e concretizá-los através dos blocos de programação. Ao longo do processo os alunos envolvem-se com experimentação e resolução de problemas iterativamente.
	Criatividade e Curiosidade Intelectual	O programa encoraja o pensamento criativo. Envolve o usuário na procura por soluções inovadoras em problemas que vão surgindo à medida que se trabalha, não apenas saber resolver um problema pré-definido.
Interpessoais e de Autodirecionamento	Interpessoais e Colaboração	O código é mais legível, acessível e partilhável que outras linguagens de programação. Os objetos visuais e código modular possibilitam o trabalho em conjunto em projetos, ou intercâmbio de código, de forma fácil.
	Autodirecionamento	Descobrir como programar uma idéia requer persistência e prática. Com projetos que consideram importantes e significativos, eles geram motivação para superar os desafios no processo de resolução de problemas.
	Responsabilização e Adaptabilidade	Ter em mente o público-alvo e como ele reagirá ao projeto. Alterar seus projetos em função da reação do público.
	Responsabilidade Social	Através do compartilhamento, promover discussões de assuntos importantes do ambiente de aprendizado pessoal ou mesmo do ambiente internacional da comunidade.

Tendo essa perspectiva educacional, voltada para o desenvolvimento de fluência com a tecnologia de informação e comunicação, o *Scratch* vem sendo usado por educadores no mundo todo para o ensino de conceitos básicos computacionais e matemáticos. De fato, existe uma comunidade específica para o compartilhamento de recursos e para discussões entre educadores que utilizam o *Scratch* como ferramenta de ensino (BRENNAN, 2009).

A pesquisa envolvendo *Scratch* e Educação, assim como as discussões presentes na comunidade educacional por trás do *Scratch* (BRENNAN, op. cit.), são em sua maioria voltadas às experiências de educadores aplicando a metodologia de ensino através do *Scratch*

em salas de aula e, a partir daí, extraindo resultados de aprendizado dos seus alunos. Existe pouca ou nenhuma pesquisa envolvendo a parte técnica do programa ou a criação de objetos de aprendizagem por meio dele.

4 INCLUSÃO DE SUPORTE A VÍDEOS NO “SCRATCH”

4.1 Avaliação Inicial

Como visto anteriormente, o *Scratch* não possui suporte a mídias de vídeo para as aplicações criadas nele. Levando em conta os benefícios do uso de diferentes mídias para o aprendizado, essa falta limita as possibilidades de construção de aplicações completas no quesito “integração de mídias”. Esse capítulo trata as modificações feitas no programa para permitir também o uso de vídeos junto com as outras mídias suportadas, como som e imagem.

Uma questão a ser considerada inicialmente é a licença do código-fonte do programa. O código é distribuído gratuitamente no *website* do desenvolvedor (LIFELONG, 2009), onde se comenta sobre a licença de uso para pesquisa.

> “Como um projeto de pesquisa, o *Scratch* explora formas de usar programação visual e projeto de interface para usuários para tornar a programação mais fácil para não-especialistas. Existem diversas possíveis direções para pesquisas futuras, como modificar o *software* para rodar em dispositivos móveis, adicionar novos tipos de dados ou estruturas para a linguagem ou aplicar as idéias de programação do *Scratch* em outros contextos. (...) Uma vez que o time do *Scratch* não possui formas de explorar todas as direções potenciais de pesquisa, estamos disponibilizando o código-fonte do *Scratch* para que outros possam também explorar e experimentar. Note, no entanto, que a licença do código-fonte inclui algumas restrições para evitar confusões entre o produto *Scratch* e os projetos de pesquisa construídos baseado no código-fonte do *Scratch*. (...)”. (LIFELONG, 2009).

A licença permite a distribuição de trabalhos derivados baseados no código-fonte para uso não-comercial, desde que sujeitos às seguintes condições:

- As notas de *copyright* e de permissão devem ser incluídas em todas as cópias do *software* e dos trabalhos derivados.
- A palavra “*Scratch*” não deve ser usada para se referir a trabalhos derivados a não ser na frase “Baseado no *Scratch* do MIT Media Lab”, desde que tal frase não seja usada para promover o trabalho derivado ou para dar entendimento de que o MIT o apóia.
- Os trabalhos derivados não devem usar o Logo do *Scratch* ou a imagem padrão do gato do “*Scratch*” usado no *software* original, distribuído em forma binária pelo grupo de pesquisa Lifelong Kindergarten, do laboratório de mídias do MIT.
- Trabalhos derivados não devem implementar ou ativar qualquer funcionalidade que permita o envio de projetos do *Scratch* para qualquer site do *Scratch* do MIT.

- O código-fonte de qualquer trabalho derivado baseado no *software* deve ser disponibilizado para leitura do público, idealmente em um *website*, sem custos.

4.2 Uso de Mídias

O *Scratch* trabalha com três tipos de mídia: imagem, som e texto. As imagens são os objetos programados pelo usuário, como os personagens ou elementos do cenário no caso de um jogo, ou então outros elementos visuais de acordo com a necessidade da aplicação sendo feita. Os sons são músicas ou efeitos sonoros, também programados para agirem de acordo com a vontade do usuário. Os formatos de imagem que o programa permite serem trabalhados são: *.gif*, *.jpeg* (ou *.jpg*), *.bmp* e *.png*. Os formatos de áudio são: *.aif*, *.aiff*, *.wav* e *.mp3*. O texto é usado principalmente nos balões de fala, em caixas de texto ou na entrada de informação textual digitada quando o aplicativo pede que o usuário escreva algo. Entre essas, os principais tipos de mídias a serem discutidos são os sons e as imagens.

Toda mídia incluída na aplicação é vinculada a um objeto programável padrão do *Scratch* chamado *Sprite*. Isso é, para incluir uma imagem ou som, é preciso adicionar a mídia a um objeto já existente ou esse objeto ser criado no momento em que a mídia é importada para dentro da aplicação. Por exemplo: ao clicar no botão para adicionar um *Sprite*, o *Scratch* vai abrir uma caixa de diálogo solicitando que o usuário escolha uma mídia de imagem para representar esse novo objeto que está sendo criado (o *Sprite*). Após uma imagem ser selecionada, o novo objeto é criado e uma instância dele é inserida automaticamente no palco do programa. A imagem selecionada pelo usuário dá a aparência visual desse novo objeto. Ela é listada na aba de mídias de imagem chamada “Trajes” (*Costumes*, em inglês). A figura a seguir mostra o *Sprite* número um (nomeado “objeto1”) com a segunda aba selecionada, mostrando as mídias de imagem que esse objeto contém em sua coleção de mídias (Figura 8). Diferentes imagens com pequenas modificações entre si permitem simular animações, caso o objeto seja programado para trocar seus “trajes” em rápida sucessão.

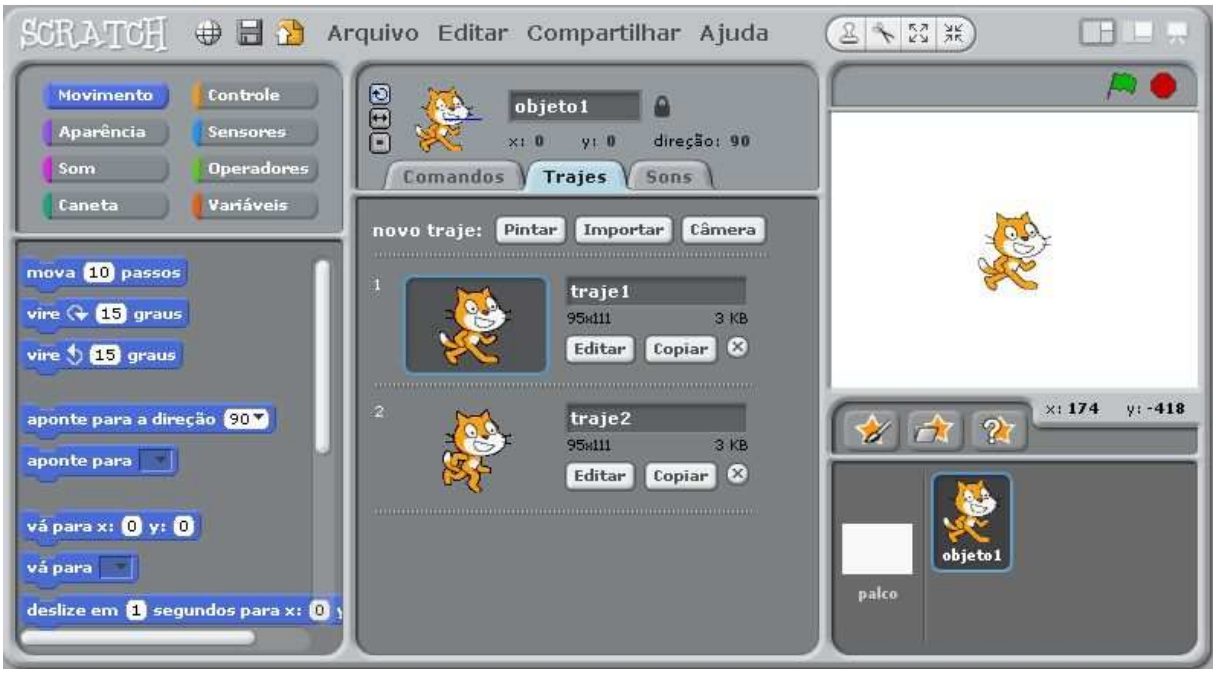

Figura 8: um objeto (*Sprite*, em inglês) de nome “objeto1” selecionado na biblioteca (canto inferior-direito). Ao centro a aba de mídias “Trajes” selecionada, mostrando a coleção de mídias do tipo imagem para esse objeto. Uma instância do objeto aparece no palco, no canto superior-direito.

Assim como na aba “Trajes” se encontram mídias do tipo “imagem” vinculadas a esse objeto, na aba “Sons” é possível adicionar elementos sonoros que irão fazer parte da coleção completa de mídias do objeto (isto é, todos os sons mais todas as imagens). Para um som genérico que não precisa estar vinculado a um objeto em específico, como uma música de fundo, existe o objeto “Palco”. O “Palco” é um objeto que não pode ser deletado e que atua normalmente como outros objetos. Nele também é possível adicionar mídias, como músicas (ou sons) ou imagens representando cenários, caso a aplicação em desenvolvimento seja um jogo interativo.

Quando um arquivo de som *.mp3* é adicionado, sua trilha de áudio é convertida automaticamente para formato *mono* e seu tamanho total após a conversão resulta em aproximadamente três vezes mais KB (*Kylobytes*). Uma ferramenta no menu “Editar” do programa permite comprimir o áudio final para um tamanho menor. Na experiência desse trabalho, um arquivo original de 2150 KB (*.mp3*) foi importado ao programa e, após a conversão, totalizou 6079 KB. Após isso, a compressão “normal” resultou em um total de 1520 KB, indicando uma possível perda de qualidade. Existem quatro níveis de compressão de áudio: alta, normal, baixa e muito baixa. Para as imagens, no mesmo menu “Editar” existe

uma opção no menu para a compressão de imagens JPEG, possibilitando ao usuário definir um nível de compressão das imagens entre 0 e 100 dentro do padrão de compressão.

4.3 Squeak

A linguagem de programação escolhida pelos autores para o desenvolvimento do *Scratch* é a linguagem *Squeak*, uma variação *open-source* da linguagem *Smalltalk-80* (INGALLS et al. 1997). A linguagem conta com um ambiente de programação onde o código é interpretado por meio da máquina virtual *Squeak*. Para abrir o código-fonte é preciso copiar o arquivo da aplicação *Scratch* (*Scratch.exe*) para o diretório onde o código-fonte do programa se encontra. Carrega-se, então, o arquivo "*ScratchSourceCode1.4.image*" no arquivo executável do *Scratch*. O ambiente de programação *Squeak* vai iniciar, possibilitando a leitura e escrita do código-fonte.

O *Scratch* utiliza a versão 2.8 modificada do *Squeak* em sua versão 2.8. Entre as modificações observadas está a retirada de algumas classes e algumas mudanças na interface do ambiente integrado de desenvolvimento (IDE - *integrated development environment*). A linguagem *Squeak* possui características próprias de programação. Por exemplo, todos os elementos do ambiente são objetos, incluindo as próprias classes. Outra característica é o ambiente de programação ser integrado e todas as classes poderem ser livremente instanciadas em qualquer ponto do código-fonte sem dificuldades ou sintaxe adicional. O estilo de programação, uma vez compreendido, facilita o entendimento (leitura) do código, resultando em uma linguagem prática e que usa poucas linhas para a criação de um método.

4.4 Reengenharia e Mapas de Classes

Antes de entrar na discussão técnica do capítulo, é importante citar uma dificuldade encontrada no desenvolvimento das modificações no projeto do *Scratch*, que foi a escassez de documentação do código. Apenas alguns pequenos trechos de documentação existem e estão embutidos no código-fonte do programa como pequenos comentários em meio ao código-fonte. Os autores comentam esse fato:

> Como o código-fonte do *Scratch* não se destina ao consumo público, não se deve esperar um nível de comentários ou documentação de API que se têm usando um sistema como Java, onde todo o ponto é ajudar desenvolvedores a construir suas próprias bibliotecas de classes. Francamente, se

> nós esperássemos para lançar o código do *Scratch* até tudo estar tudo perfeitamente comentado, ele nunca seria liberado. (LIFELONG, 2007)

Dessa forma, não tendo em mãos uma documentação que explicasse a fundo o funcionamento do programa, a solução foi o estudo e o mapeamento do código-fonte para ter um entendimento completo do funcionamento do programa. Esse artifício pode ser visto como uma reengenharia do programa, onde muitas vezes, por exemplo, foram realizados testes de modificações do código-fonte baseados em tentativa e erro em tempo de execução.

Uma forma de auxiliar essa tarefa foi o uso do mapeamento de classes. Essa atividade consiste na montagem de diagramas para auxiliar a visão macro da execução do programa, descrevendo as funções das classes assim como a forma como elas se relacionam. No total, foram feito quatro diagramas para auxiliar o desenvolvimento do trabalho: um para analisar de descrever a função de todas as categorias e classes do programa; o segundo para listar todas as modificações feitas no código-fonte; o terceiro para anotar as etapas da resolução do problema da inserção de objetos; e o último para dar suporte à resolução dos problemas advindos do uso da classe `MovieMedia`, mostrado mais adiante nesse capítulo. Abaixo é mostrado o trecho de um dos diagramas usado (Figura 9).

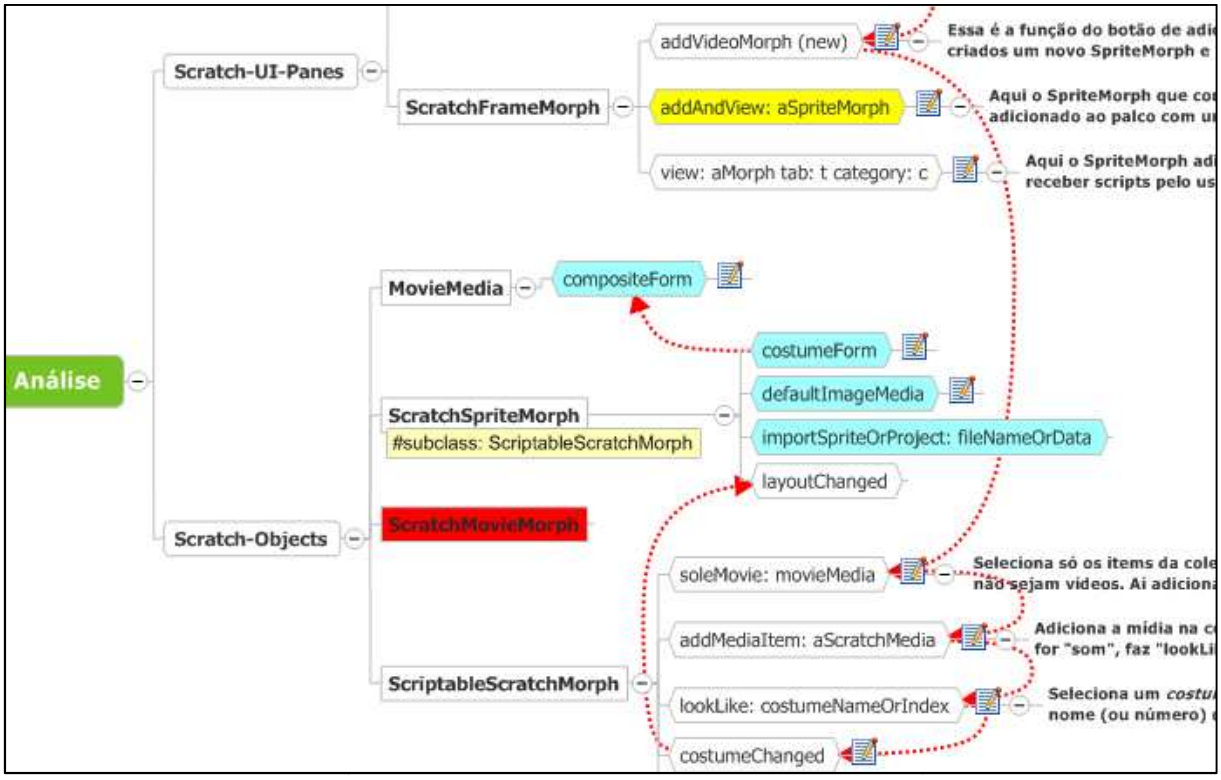

Figura 9: trecho do terceiro diagrama de classes, auxiliando a solução de um problema.

4.5 Categorias

Todas as classes no *Squeak* são organizadas em categorias. Existem categorias que agrupam, por exemplo, todas as classes relativas aos componentes principais da linguagem *Squeak*, ou então categorias para todas as classes que tratam os gráficos, rede, sons, formas, entre outros. Dentro do escopo de programação do *Scratch*, são especificadas doze categorias que organizam todas as classes e métodos do programa (Tabela 4.1).

Tabela 4.1: categorias para organização das classes do *Scratch* de acordo com seu propósito.

Categorias de Classes	**Descrição**
Scratch-Objects	Mídias e objetos programáveis. Nessa categoria é definida a mídia genérica `ScratchMedia` e as subclasses `ImageMedia` e `SoundMedia`. Também aqui é definida a classe genérica para objetos programáveis (`ScriptableScratchMorph`).
Scratch-Blocks	Todos os blocos de programação do *Scratch* utilizados pelo usuário. Duas classes principais derivam todas as outras. São elas: `BlockMorph` para os blocos em si e `ArgMorph` para os argumentos que são comumente usados como parâmetros dos blocos de programação.
Scratch-Execution Engine	Motor de execução. São definidas quatro classes: eventos, processos, pilha de execução e *Thread*.
Scratch-Object IO	Trata a serialização dos objetos e mídias para salvar projetos em arquivos ou para transmissão pela rede.
Scratch-UI-Dialogs	Classes para suporte à interface que estabelecem “diálogo” com o usuário. Exemplos nessa categoria são: a janela de escolha de um arquivo externo, caixas de ajuda ou a pergunta de confirmação de saída do programa.
Scratch-UI-Panes	Painéis (formas) principais que formam o visual do programa. `FrameMorph` é o painel básico inicial que dá início a todos os outros. `LibraryMorph` é o painel da biblioteca de objetos; `ScriptEditorMorph` define o painel central de mídias e programação de um objeto; `ScratchViewer` é o painel que mostra as categorias dos blocos de programação.
Scratch-UI-Watchers	Classes de controle de eventos da interface.
Scratch-UI-Support	Classes de suporte à interface, usadas pelos painéis principais.

Scratch-Paint	Ferramenta de edição de imagens interna do *Scratch*.
Scratch-Sound	Ferramenta de edição de sons *midi* e funções de áudio.
Scratch-Translation	Classes para manipulação da tradução da interface.
Scratch-Networking	Classes para manipulação de funções de rede e Internet.

As categorias que definem os objetos básicos do programa, os blocos e os painéis visuais são as principais candidatas a sofrerem modificações para adicionar o suporte a vídeos no programa. Todas as categorias do *Scratch* são identificadas pelo prefixo "*Scratch-*" para serem diferenciadas das outras categorias de classes do ambiente *Squeak*. Um dos objetos dessa diferenciação tem por objetivo saber qual código-fonte está sob qual licença de distribuição, se do *Scratch* ou da linguagem *Squeak*.

4.6 Avaliação Inicial do Código-Fonte

4.6.1 Classes de vídeo

Verificar o suporte a vídeos da linguagem *Squeak* é o primeiro passo. Com a organização da biblioteca de classes da linguagem, é possível verificar que três categorias são promissoras dentro do ambiente da versão 2.8, a versão usada pelo *Scratch*. As categorias são `Movies-Kernel`, `Movies-Player` e `Morphic-Movies`.

As categorias de classes que possuem o prefixo "*Morphic*" contêm toda definição das formas básicas do *Squeak*. Essas formas representam objetos gráficos interativos. A classe `Morph` define características comuns a todas as formas do *Squeak* e todas as outras formas são derivadas dessa classe. Outra classe importante é chamada `HandMorph` e determina o comportamento do cursor (*mouse*). Por último, também entre as mais importantes, está a classe `WorldState`, que mantém o estado atual de todas as formas que compõem o "mundo", ou seja, todas as formas que têm instâncias na tela.

Em `Morphic-Movies` encontramos quatro classes relacionadas a vídeos. São elas: `JPEGMovieFile`, `MPEGFile`, `MovieDisplayMorph` e `MoviePlayerMorph`.

A primeira classe, `JPEGMovieFile`, explora um recurso da linguagem que permite um tipo de formato de vídeo chamado "Vídeos JPEG". Esse tipo de formato (extensão "*.jmv*") contém um cabeçalho, uma coleção de *frames* comprimidos em formato JPEG do arquivo de

vídeo e, possivelmente, uma ou mais trilhas de áudio. Os quadros (*frames*) do arquivo de vídeo em formato de imagem (JPEG) são mostrados um após o outro em uma taxa constante para simular o efeito da imagem em movimento de um vídeo. A classe contém métodos para converter filmes MPEG em filmes JPEG. No entanto, esse formato após convertido tende a ser 1,5 a três vezes o tamanho do vídeo MPEG original de qualidade similar, excluindo ainda o tamanho do áudio que também é substancialmente maior que o áudio original do arquivo MPEG. Logo, o uso dessa classe não é adequado para os propósitos do trabalho.

`MPEGFile` é uma classe-interface para a classe `LibMPEG3` (Willians, 2010). Segundo comentário no código-fonte, a classe original foi modificada em setembro de 2000 para funcionar no ambiente *Squeak*. O código *Squeak/Smalltalk* da classe produz como resultado um código em linguagem C para manipular arquivos formato *.mpeg*. Ela é usada para o manuseio de dados de arquivos *.mpeg* entre o sistema operacional e o *Squeak*, permitindo sua utilização por outras classes. `MovieDisplayMorph` é classe que representa uma forma elementar para servir de base à visualização de um arquivo de vídeo (por exemplo, uma instância de `MPEGFile`). Por último, `MovieDisplayMorph` é uma classe que monta uma interface de usuário genérica com botões de *playback*, como “iniciar filme” (*play*), “parar filme” (*stop*) ou “voltar ao início” (*rewind*). Essa classe também utiliza `MPEGFile` para gerenciar os dados do arquivo de vídeo.

Nenhuma outra classe no ambiente do *Squeak* (na versão 2.8, a usada pelo *Scratch*) trata formatos de vídeo. A pesquisa sobre outras soluções mostra que existem algumas poucas outras formas de manipulação de vídeos, mas, no entanto, esses recursos não têm o código-fonte aberto ou não possuem licença de distribuição gratuita (DECK et al., 2004).

4.6.2 Mídia de vídeo

Uma análise mais detalhada do código do *Scratch* mostra que existe uma classe chamada `MovieMedia` dentro da categoria `Scratch-Objects` (onde estão algumas das principais classes do sistema). Mesmo com o programa não oferecendo suporte a vídeos em sua versão final para os usuários, a descrição da classe indica que ela define objetos do tipo “mídia de filme” (*movie media*) e, em conjunto com as classes `ImageMedia` e `SoundMedia`, totaliza o conjunto de classes que descrevem as mídias do *Scratch*. Essas três classes são derivadas da classe-pai `ScratchMedia`, que estabelece características comuns a todos os tipos de mídia do programa (Figura 10).

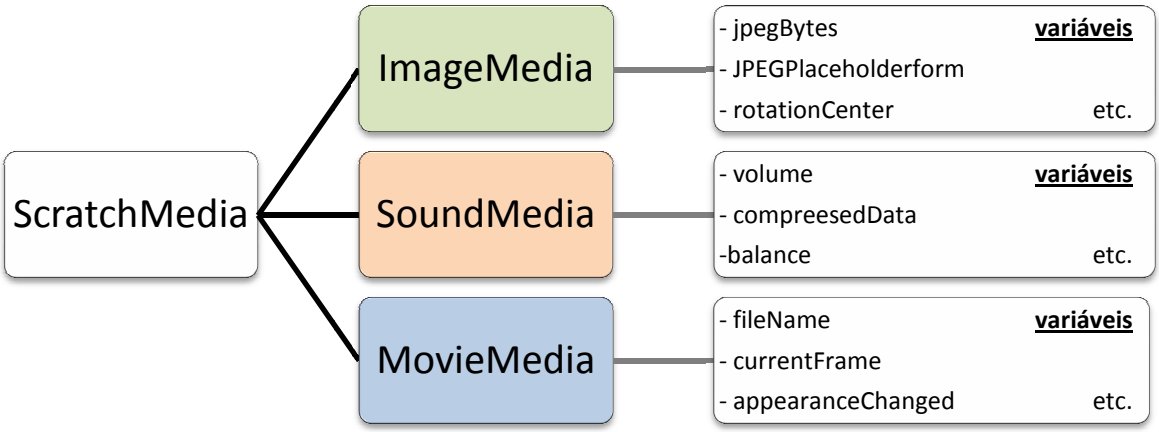

Figura 10: diagrama mostrando a relação subclasses para mídias de imagem, som e vídeo derivados da classe básica `ScratchMedia`. Alguns atributos específicos de cada classe são listados à direita.

Essa classe forma um ponto de partida para o entendimento do código para, posteriormente, permitir modificações no programa para obter suporte a mídias de vídeo. É preciso entender como essas classes de mídias são organizadas e utilizadas no sistema. Usando os mapeamentos de classes citados anteriormente, observa-se a relação entre classes-pai e classes-filho e o objetivo de cada classe no sistema (Figura 11).

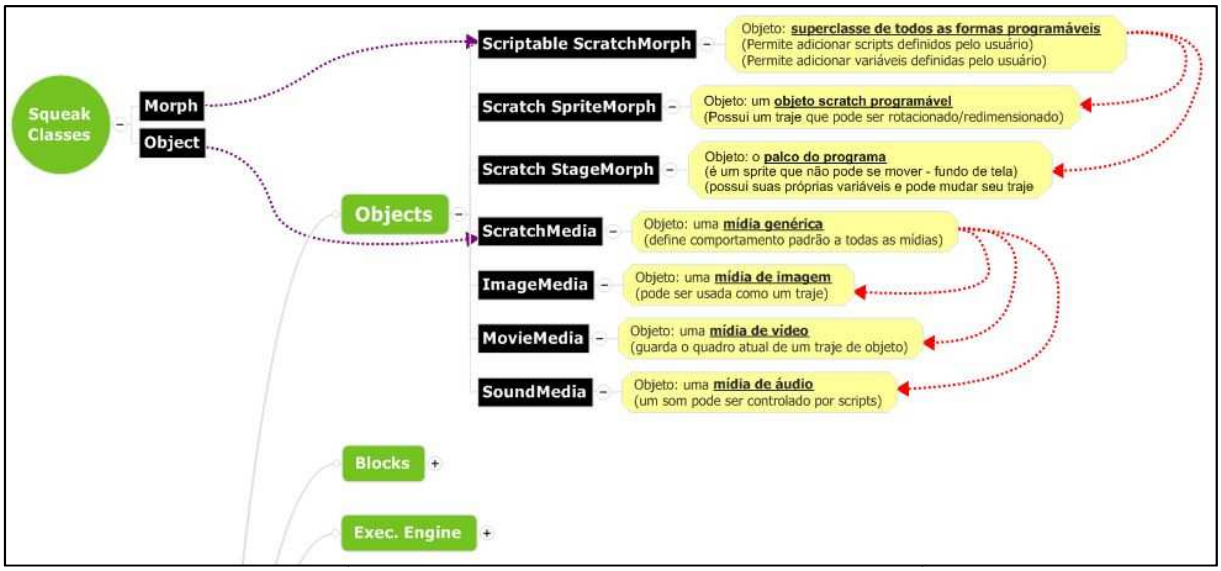

Figura 11: parte do primeiro diagrama de classes feito para a análise inicial do código-fonte do *Scratch*.

4.6.3 Objetos do “Scratch”

As mídias que o *Scratch* usa nativamente nos seus objetos são imagens e sons. Todo o conteúdo midiático inserido no programa é vinculado a algum objeto. Um personagem de um jogo, por exemplo, pode ter diferentes imagens o representando em diferentes posições (trajes) e diferentes vozes ou efeitos sonoros (sons). Existem três classes que ditam o comportamento dos objetos (*sprites*): uma classe superior que define genericamente todo objeto programável do *Scratch*, chamada `ScriptableScratchMorph`, e duas subclasses, `ScratchSpriteMorph` e `ScratchStageMorph`. Entre as duas últimas, a primeira representa todos os objetos comuns e a segunda o objeto especial “*Stage*”, que representa o palco do programa.

Os objetos do *Scratch*, chamados de *Sprites*, e o palco são essencialmente formas. Essas formas possuem o visual relacionado ao conjunto de mídias de imagem que possuem, sendo uma dessas imagens mostrada na tela por vez. O conjunto de imagens de um objeto é mostrado na aba “Trajes”, no painel principal. O palco, por exemplo, possui um traje inicial branco chamado *background* (ou fundo de tela).

Com esse formato do *Scratch* em vista, faz-se lógico que uma mídia de vídeo também seja atrelada a um objeto programável, ou seja, a uma forma (*sprite*). Desse jeito, um vídeo aberto no palco poderia aceitar do usuário os mesmos blocos programáveis que criam o comportamento das outras formas no palco. Além disso, a mídia de vídeo também receberia as mesmas funcionalidades, como poder ser redimensionada e ter o menu de contexto acessível por meio do botão direito do *mouse*.

4.7 Painel de mídias

A primeira tentativa de alteração no programa foi a inclusão de uma nova aba para mídias de vídeo no painel central do programa. É nesse painel que os usuários programam o objeto selecionado escolhendo blocos de programação. No painel, a primeira aba tem o título “Comandos” (*Scripts*, em inglês) e é onde se organizam os códigos criados pelos usuários. A segunda e a terceira aba têm os títulos “Trajes” e “Sons”, respectivamente, e indicam as coleções de mídias de imagem e de som que o objeto selecionado possui.

Tal painel é uma instância da classe `ScratchScriptEditorMorph`. A classe constrói todo o painel central do programa, instanciando as formas que o compõem. Entre

essas formas estão as abas, sendo esse o ponto inicial de análise do código. O método `initialize` inicia os atributos da classe e chama as funções de construção (Figura 12).

```
ScratchScriptEditorMorph methodsFor: 'initialization'
initialize
	super initialize.
	self
		initFrontFromForm: (ScratchFrameMorph skinAt: #scriptPaneFrameTransparent2)
		topSectionHeight: 90.
	self color: (Color r: (149/255) g: (154/255) b: (159/255)).

	thumbnailMorph _ ScratchThumbnailMorph new.
	self addMorph: (thumbnailMorph position: self position + (37@16)).

	self addNameBox.

	pageViewerMorph _ ScrollFrameMorph2 new
		growthFraction: 0.1;
		color: ScratchFrameMorph scriptsPaneColor.
	self addMorph: (pageViewerMorph position: (self left @ (self top + topSectionHeight))).

	rotationButtons _ #().
	readoutMorphs _ #().
	self target: nil.
	thumbnailMorph extent: 50@50.
	self extent: 300@400.
	self createTabPane.
```

Figura 12: código-fonte do método `initialize`, classe `ScratchScriptEditorMorph`.

No final do código-fonte, o método "`self createTabPane`" indica para a instância da classe `ScratchScriptEditorMorph` que traga para execução o método `createTabPane`. O código mostra que o método tenta criar o painel das abas e adicioná-lo ao painel central.

A linha 16 do método possui um comentário indicando o conjunto de instruções que cria as abas. O código na linha 17, logo abaixo, indica um novo vetor de elementos (chamado no *Squeak* de *collection*, ou coleção) inserido em um laço de execução, onde, em cada iteração, é criada uma forma de aba para cada elemento do vetor. Também são iniciados alguns atributos no laço. Nesse ponto, a primeira inserção no código é feita, com a inclusão de um novo elemento denominado "*Videos*" no vetor (Figura 13).

```
ScratchScriptEditorMorph methodsFor: 'initialization'
createTabPane

    | tabOnForm tabOffForm tabID tabLabel |

    "create tab pane"
    tabPaneMorph _ ScratchTabPaneMorph new.
    tabPaneMorph
        borderWidth: 0;
        color: Color transparent;
        targetPane: self.

    tabOnForm _ (ScratchFrameMorph skinAt: #tabOn).
    tabOffForm _ (ScratchFrameMorph skinAt: #tabOff).

    "add the tabs"
    #(Scripts Costumes Sounds Videos) do: [:spec |
        tabID _ spec asString.
        tabLabel _ tabID localized.
        tabPaneMorph
            createTab: tabID
            withLabel: tabLabel
            onForm: tabOnForm
            offForm: tabOffForm].

    "set current tab and add to frame"
    tabPaneMorph currentTab: 'Scripts'.
    self addMorph: tabPaneMorph.
```

Figura 13: código-fonte modificado do método `createTabPane`, classe `ScratchScriptEditorMorph`.

Imediatamente uma nova forma de aba aparece no painel de abas. O resultado, no entanto, é falho, pois selecionar a aba recém-criada não resulta em nada diferente no aspecto visual do programa. De fato, a aparência no painel central após clicar na nova aba continua sendo igual ao da aba selecionada por último (as abas Comandos, Trajes ou Sons).

Nesse momento, o código-fonte do método não dá nenhuma pista adicional para avançar. A penúltima linha mostra o método `currentTab:`, que apenas ajusta aspectos visuais do painel das abas. Um novo caminho é retornar e pesquisar os métodos da classe `ScratchScriptEditorMorph`.

Na busca, o método `currentCategory:` traz informações relevantes: o parâmetro da classe é testado sucessivamente contra cadeias de texto contendo o nome exato das categorias, *Scripts* (comandos), *Costumes* (trajes) e *Sounds* (sons). Caso uma dessas categorias satisfaça o texto condicional, a instância de classe `pageViewerMorph` recebe o método que diz que sua variável `contents` deve ser atualizada para o resultado de uma dos seguintes métodos: `costumesPage:`, `soundsPage:` ou `blocksBin`.

A variável citada acima, `pageViewerMorph`, é uma referência (também chamado de ponteiro) a uma instância da classe `ScrollFrameMorph2`. Essa classe é da categoria que dá suporte à interface do usuário e representa uma forma (*frame*) que, no caso de ter mais objetos que seu tamanho comporta, mostra uma barra de rolagem para o usuário navegar pelo seu conteúdo. Essa é uma forma-filho instanciada pela já citada classe `ScratchScriptEditorMorph` (a classe que representa o painel central do programa, chamado de painel de programação e de mídias) onde é criada uma forma para cada uma das suas abas. `ScrollFrameMorph2` é uma classe de suporte à interface de usuário e é muito usada por diversos painéis do *Scratch*.

Uma modificação é feita no método para ele ter um novo teste condicional de categoria para o texto “*Videos*”, nome da categoria criada anteriormente, indicando como alvo de conteúdo o resultado do método `moviesPage: xOffset` (Figura 14). A execução no programa indica um erro: “message not understood: ‘*moviesPage:*’”, indicando que o método não existe.

Nesse momento, surge uma dificuldade do ambiente de desenvolvimento: para criar o novo método é imprescindível o conhecimento dos outros métodos `costumesPage:` e `soundsPage:`, no entanto, uma busca simples não encontra esses métodos na classe atual `ScriptsEditorMorph`. Como a classe-pai também não possui esses métodos, existe uma dificuldade na busca por métodos no ambiente, uma vez que não há uma ferramenta de busca de métodos que abranja todas as classes nessa versão do *Squeak*. Uma alternativa no momento é instituir que a aba “*Videos*” referencie temporariamente o conteúdo de `soundsPage:`, um método que existe. Essa solução funciona, e a nova aba de mídias de vídeo passa a dar certo, referenciando, temporariamente, o mesmo conteúdo da aba dos sons.

```
ScratchScriptEditorMorph methodsFor: 'accessing'
currentCategory: aString

    | xOffset |
    currentCategory _ aString.
    self target ifNil: [^ self].
    xOffset _ 0.

    World activeHand newKeyboardFocus: nil.

    currentCategory = 'Scripts' ifTrue: [
        pageViewerMorph contents: self target blocksBin].
    currentCategory = 'Costumes' ifTrue: [
        pageViewerMorph contents: (self target costumesPage: xOffset)].
    currentCategory = 'Sounds' ifTrue: [
        pageViewerMorph contents: (self target soundsPage: xOffset)].
    currentCategory = ‘Movies’ ifTrue: [
        pageViewerMorph contents: (self target moviesPage: xOffset)].

    pageViewerMorph contents color: ScratchFrameMorph scriptsPaneColor.

    self world ifNotNil: [self world startSteppingSubmorphsOf: pageViewerMorph contents].
```

Figura 14: código-fonte do método `currentCategory:`, classe `ScratchScriptEditor`.

4.8 Importando mídias de vídeo

Deixando em segundo plano as abas do painel de mídias, a importação de mídias é outro ponto a ser tratado. No *Scratch*, as mídias externas ao programa são trazidas ou pelo botão de importação presente em cada aba no painel de programação e mídias, ou pelo botão da biblioteca que cria um novo objeto (*Sprite*) a partir da imagem selecionada nos sistema de arquivos do sistema operacional. O botão para criar um novo objeto importa mídias porque a criação de um novo objeto necessita ao menos uma imagem para representar esse objeto no mundo. Clicar no botão para adicionar um objeto chama uma caixa de diálogo para o usuário escolher um arquivo, mas é visível que há um filtro para mostrar apenas diretórios ou arquivos do tipo imagem (*.jpg*, *.gif*, *.png*, etc.).

Inspecionando a instância da caixa de diálogo, vê-se que a classe que define a forma se intitula `ScratchFileChooserDialog`, uma classe da categoria `Scratch-UI-Dialogs`. A análise da inicialização da classe não traz nenhuma informação. Todos os

métodos nela indicam rotinas de construção de formas para compor o visual da caixa de diálogo.

Ao invés de analisar a caixa de diálogo, o painel da biblioteca mostra em sua lista de métodos um denominado `makeNewSpriteButtons:`. Nele, uma seção do código-fonte mostra a definição das características dos três botões que inserem um novo objeto programável no *Scratch*. O primeiro botão desenha um novo objeto, o segundo permite ao usuário escolher um novo objeto do arquivo e o terceiro adiciona um objeto randomicamente do diretório padrão de imagens do *Scratch*. Cada botão é especificado com um ícone, uma mensagem de dica quando o ponteiro do *mouse* pousa sobre eles e um seletor, isto é, a rotina que é ativada por cada botão (Figura 15).

```
ScratchLibraryMorph methodsFor: 'initialization'
makeNewSpriteButtons: aScratchFrameMorph
    "Return a morph containing a set of new sprite buttons."

    | panel buttonSpecs buttons button butonExtent x |
    panel _ Morph new color: Color transparent.

    buttonSpecs _ #(
        "icon name          selector              tooltip"
        (newSpritePaint     paintSpriteMorph      'Paint new sprite')
        (newSpriteLibrary   addSpriteMorph        'Choose new sprite from file')
        (newSpriteSurprise  surpriseSpriteMorph   'Get surprise sprite')
    ).

    buttons _ buttonSpecs collect: [:spec |
        button _ ToggleButton new
            onForm: (ScratchFrameMorph skinAt: (spec at: 1))
            offForm: (ScratchFrameMorph skinAt: (spec at: 1)).

    […]

    ^ panel
```

Figura 15: código-fonte do método `makeNewSpriteButtons:`, classe `ScratchLibraryMorph`.

O seletor do segundo botão mostra que o método `addSpriteMorph` é chamado quando o botão é clicado. A hierarquia das classes mostra classes de onde esse método pode

derivar. Analisando os métodos da instância da classe-pai da biblioteca, vemos que entre eles existe o método apontado.

Analisando o código-fonte do método, nota-se que nele se iniciam as operações para a inserção da mídia do sistema operacional para dentro do ambiente *Scratch*. Primeiramente, uma instância do painel de diálogo para seleção do arquivo é criada e, em retorno, devolve o endereço completo da mídia selecionada pelo usuário no diretório de arquivos. A seguir, uma nova forma é criada para abrigar a mídia, funcionando de certa forma como uma "moldura". Nesse momento, tendo o endereço do arquivo e a moldura criada a partir das dimensões da mídia, é criada uma instância da classe `ImageMedia` recebendo como argumento a forma e o endereço. Essa instância da classe abrigará as informações sobre a mídia que está sendo importada ao programa. Finalmente, um novo objeto *Scratch* é criado (`ScratchSpriteMorph`) e a nova mídia é inserida como única mídia de representação visual do objeto. Por fim o objeto é adicionado à biblioteca e ao palco (Figura 16).

```
ScratchFrameMorph methodsFor: 'menu/button actions'
addSpriteMorph

    | result f m el |
    self world activeHand toolType: nil.
    self paintingInProgress ifTrue: [^ self beep].

    result _ ScratchFileChooserDialog chooseSpriteCostumeFor: self.
    result = #cancelled ifTrue: [^ self].
    (result asLowercase endsWith: '.sprite')
        ifTrue: [^ self importSpriteOrProject: result].

    [f _ Form fromFileNamed: result] ifError: [^ self].
    el _ ImageMedia new form: (ScratchFrameMorph scaledFormForPaintEditor: f).
    m _ ScratchSpriteMorph new soleCostume: el.
    el mediaName: (m unusedMediaNameFromBaseName: (FileDirectory localNameFor: result)).

    self addAndView: m.
```

Figura 16: código-fonte do método `addSpriteMorph`, classe `ScratchFrameMorph`.

Nesse momento, é possível ver que a mídia do novo objeto deve ser uma instância da classe `MovieMedia`, e não `ImageMedia`. Isso parte do pressuposto que um vídeo, assim como uma imagem, possui uma aparência para ser mostrada no palco, ao contrário de uma mídia de som que não possui. Cria-se, então um novo método semelhante chamado `addVideoMorph`, para ser invocado como seletor de um botão. Seu código possui as

seguintes diferenças: primeiramente, a classe de mídia usada é a `MovieMedia`. Em seguida, analisando a seqüência de métodos chamados quando `MovieMedia` é instanciado, vê-se que não é necessário criar uma nova forma para abrigar a mídia. Isso decorre do método `loadFile:` automaticamente criar uma forma baseado nas dimensões originais do arquivo de mídia passado como argumento (Figura 17).

```
ScratchFrameMorph methodsFor: 'menu/button actions'
addVideoMorph
	| result el m |

	self world activeHand toolType: nil.
	self paintingInProgress ifTrue: [^ self beep].

	result _ ScratchFileChooserDialog chooseMovieCostumeFor: self.
	result = #cancelled ifTrue: [^ self].
	(result asLowercase endsWith: '.sprite')
	    ifTrue: [^ self importSpriteOrProject: result].

	el _ MovieMedia new loadFile: result.
	m _ ScratchSpriteMorph new soleMovie: el.
	el mediaName: (m unusedMediaNameFromBaseName: (FileDirectory localNameFor: result)).

	self addAndView: m
```

Figura 17: código-fonte do novo método `addVideoMorph`, classe `ScratchFrameMorph`.

Duas necessidades surgem: a classe `ScratchFileChooserDialog` não possui o método `chooseMovieCostumeFor:` definido e a classe `ScratchSpriteMorph` não tem o método `soleMovie:`. O primeiro inicia atributos para a caixa de diálogo de seleção de arquivo, além da importante função de definir quais extensões de arquivo serão mostradas. O segundo inicia e limpa uma coleção de mídias para o objeto `ScratchSpriteMorph` criado, adicionando como única mídia a essa coleção a mídia selecionada.

Definir `chooseMovieCostumeFor:` consiste em criar um método semelhante ao seu equivalente para imagens. A diferença fica por conta da linha "`extensions: #(mpg mpeg sprite);`" onde a variável `extensions` é atribuída a uma coleção (vetor) de extensões permitidas. Terminado esse passo, `soleMovie` sofre modificação parecida. O método é definido na classe `ScriptableScratchMorph`, superclasse da instância

`ScratchSpriteMorph` onde ele é chamado. A mudança consiste em verificar se a mídia é do tipo correto para limpar a coleção de mídias do objeto e adicioná-la como a única mídia.

Volta-se para modificar o código-seletor dos botões. Uma solução rápida é substituir o seletor do terceiro botão (`surpriseSpriteMorph`) pelo novo método seletor, `addVideoMorph`. Entre os três botões, o escolhido é o que possui menor utilidade por selecionar um objeto aleatório do diretório de imagens padrão do *Scratch*. Substitui-se a linha "`(…) surpriseSpriteMorph 'Get surprise sprite'`" pela linha "`(…) addVideoMorph 'Choose new vídeo from file'`".

A conclusão dessa seqüência lógica de modificações resulta no novo objeto contendo a mídia de vídeo sendo adicionado à biblioteca e uma instância do objeto sendo adicionada ao palco do *Scratch*. No entanto sua aba de mídias ainda não funciona corretamente.

4.8.1 Seleção de arquivos

A seção anterior mostrou a lógica de criação do novo objeto *Scratch* contendo a mídia selecionada. A classe `ScratchFileChooserDialog`, que constrói a interação (diálogo) para que o usuário selecione o arquivo de mídia, no entanto, teve que sofrer diversas alterações para permitir a seleção de arquivos de vídeo. Uma dessas alterações foi citada anteriormente e consiste no novo método `chooseMovieCostumeFor:` definido para formatos de arquivo *.mpg* e *.mpeg*. Essa subseção cobre as outras alterações.

Primeiramente, `chooseImageFileType:title:` é um método da classe muito parecido com o método criado anteriormente, `chooseMovieCostumeFor:`. A diferença é que o método recebe como primeiro parâmetro um tipo. O tipo é uma variável especial do *Scratch* que usa o artifício da linguagem *Squeak* chamado símbolo. Um símbolo é uma cadeia de texto precedida pelo identificador "#" que garante que ele seja único globalmente. O *Scratch* identifica seus tipos de objetos dessa forma. Exemplos são `#image`, `#sound`, `#project`, `#sprite`, entre outros. Como esse método provavelmente é usado na importação de mídias, possivelmente no painel central de mídias e comandos, uma réplica dele é criada com o nome `chooseMovieFileType:title:`. Modificações são os formatos de arquivo especificados (somente *.mpg* e *.mpeg*) e a retirada das rotinas de importação da mídia para o editor de imagens do *Scratch* (a nova função importa vídeos).

O método `getDefaultFolderForType:title` também é alterado para refletir um possível diretório dedicado para mídias de vídeo. Se o tipo (variável `type`) for `#movie`,

o diretório padrão do diálogo de busca de arquivos é redirecionado a um diretório específico. Nesse caso, a mudança no método é adicionar um teste condicional para caso o tipo seja vídeo. Se existir um diretório com nome "*Movies*" o resultado da função será o caminho até esse diretório definido como padrão para os arquivos de vídeo.

Um novo método é criado, chamado `scratchMovies`, e retorna como resultado justamente o caminho devolvido pela função `getDefaultFolderForType:title`. Esse método é usado pelos botões de atalho (*home*, *computer*, *desktop*, etc.) inseridos no painel de diálogo de busca de arquivo `ScratchFileChooserDialog`.

Nesse estágio do trabalho uma grande dificuldade era descobrir como o novo tipo `#movie` poderia ser definido. Enquanto isso não era descoberto, uma saída foi determinar no método `addVideoMorph` que a mídia de vídeo não fosse escolhida através da caixa de diálogo padrão porque ela usaria o tipo `#movie`. A solução momentânea para fim de testes foi inserir como resultado explícito o caminho (*path*) até um arquivo de vídeo específico, posicionado convenientemente na área de trabalho do computador. Dessa forma, o botão de adicionar mídia de vídeo temporariamente só poderia adicionar esse arquivo.

4.9 A Categoria "Vídeos"

Levando em conta que o controle dos vídeos necessita novos comandos de controle de execução (tal como *play*, *stop*, entre outros), uma necessidade prevista era a criação de uma nova categoria de comandos no painel de comandos à esquerda do programa. A categoria teria como nome "*movies*", assim como a categoria "*sound*" possui comandos para controle do som pelo usuário.

O painel de categorias é uma instância da classe `ScratchViewerMorph`. O método de inicialização `initialize` mostra que parte das principais funções é definida na classe-pai `ScratchFrameMorph`, o painel principal do programa. Outro método da categoria, no entanto, já tem informações mais concretas, `rebuildCategorySelectors` inicia a lista de categorias através de uma coleção contendo o nome de todas as categorias. Insere-se aí o nome da futura nova categoria "*Movies*".

O aspecto do *Scratch* permanece igual. Em geral, mudanças no código são atualizadas na hora devido à característica dinâmica da linguagem *Squeak*. A conclusão é que a inicialização das categorias ocorre uma única vez na inicialização do programa. Dessa forma, fecha-se o *Scratch* e é aberta uma nova instância por meio do menu de contexto do *Squeak*,

aberto com o menu do botão esquerdo sobre o painel de programação. A inicialização do *Scratch* traz à tona um erro em tempo de execução: "*key not found*". Por se tratar de um erro desconhecido, usa-se o *debugger* do *Squeak* para analisar o contexto de execução (Figura 18).

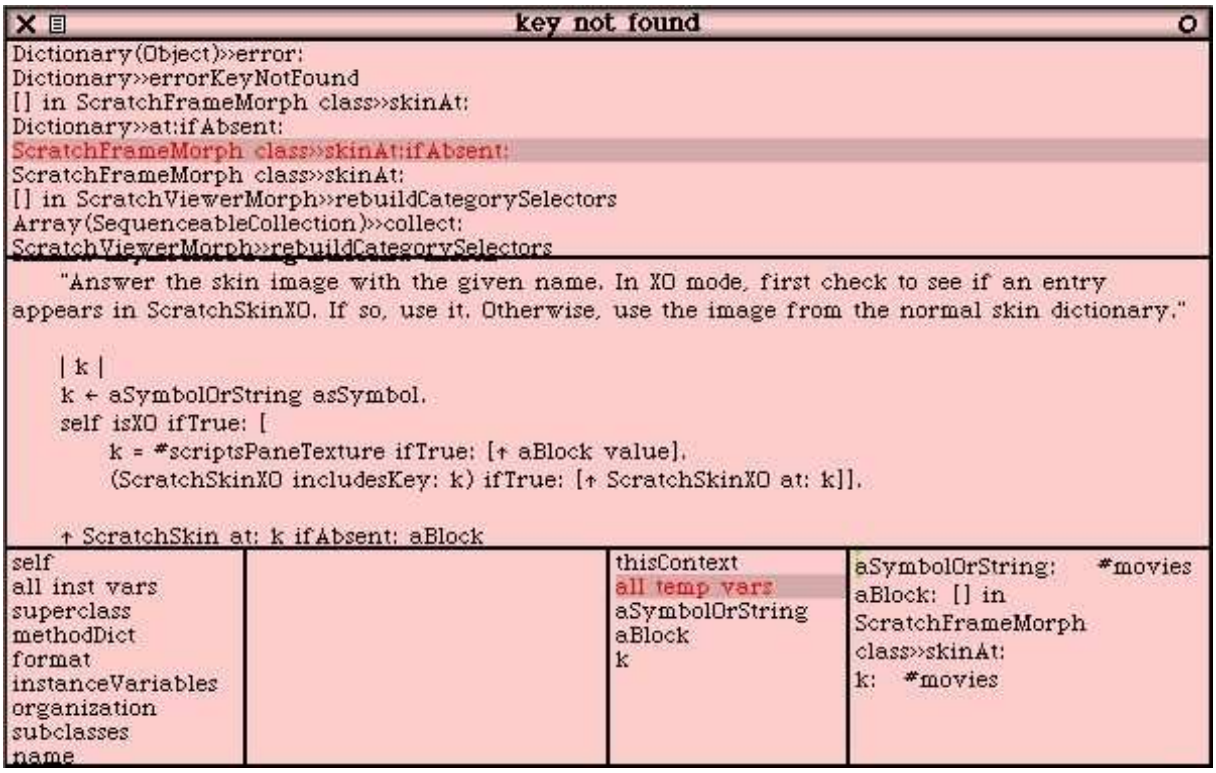

Figura 18: *debugger* do *Squeak* aberto mostrando a pilha de execução no topo e o contexto de execução no canto inferior direito.

A análise da pilha de execução que retorna o erro mostra que o símbolo `#movies` não foi encontrado. Pode-se concluir que a criação das categorias define alguns dos símbolos globais demandados por alguns métodos anteriores, como `chooseImageFileType:` da classe `ScratchFileChooserDialog`.

Observa-se que o erro acontece no tempo de execução de um método da instância-pai de `ScratchViewerMorph`. Esse método tem o nome `skinAt:ifAbsent:`. Após uma análise profunda do código, entende-se que o erro de execução é a ausência de imagens para a construção da forma da categoria. Essas imagens devem estar localizadas no diretório padrão de arquivos imagens da aparência do *Scratch*. Três arquivos de imagem são criados seguindo o modelo das imagens já existentes: *movies.gif*, *moviesOver.gif* e *moviesPressed.gif*. Cada uma delas representa o botão da categoria em um estado diferente de acordo com a ação do

mouse. O resultado dessa ação é a abertura do programa com a nova categoria criada, no entanto com problemas na aparência do programa. A forma da nova categoria excede o espaço e fica situada sobre a barra divisória do painel.

A solução é analisar a geometria do painel no código-fonte. Após uma revisão do código-fonte e uma série de tentativas e erro para correções da aparência, os métodos modificados são `rebuildCategorySelectors` e `drawBackgroundOn:`. As seções dos métodos, finalizadas com as mudanças, se encontram abaixo (Figura 19) (Figura 20).

```
[...]
self width: catButtonsExtent x.
        pageViewer position: self position + (0@(catButtonsExtent y + 20)).
        topSectionHeight _ catButtonsExtent y + 16.
```

Figura 19: trecho do código-fonte do método `rebuildCategorySelectors`, classe `ScratchViewerMorph`.

```
ScratchViewerMorph methodsFor: 'drawing'
drawBackgroundOn: aCanvas
        "Draw my background."

        color isTransparent ifTrue: [^ self]

        aCanvas
            fillRectangle: (self topLeft extent: (self width @ (catButtonsExtent y +20)))
            color: color.
```

Figura 20: método `drawBackgroundOn:`, classe `ScratchViewerMorph`.

Terminadas essas modificações, a nova categoria surge com o *Scratch* novamente tendo a aparência correta. A seleção da categoria, no entanto, traz um conjunto vazio de instruções para o usuário. A próxima seção aborda essa questão.

4.10 Blocos de programação

Criar comandos de controle de execução (*playback*) para os vídeos inseridos no *Scratch* é essencial para tornar útil essa adição ao programa. O primeiro estágio para conseguir isso é a busca pela classe e métodos relacionados às funções existentes. Lembrando que `ScriptableScratchMorph` é a classe básica de todo objeto programável, analisam-se seus métodos.

A construção dos blocos é feita no método de classe `blockSpecs`. Ele retorna uma coleção de blocos que são comuns a todos os objetos. Diversos tipos de argumentos e variáveis de definição de blocos são explicados no método. Eles são chamados de *flags* (Tabela 4.2).

Tabela 4.2: seletores dos tipos de blocos existentes.

-	Sem *flags*.
b	Bloco de retorno de dado booleano.
c	Bloco em forma de “C” contendo uma seqüência de comandos.
r	Bloco de retorno de dado (*reporter*)
s	Forma especial de comando com sua própria regra de avaliação.
t	Comando de tempo, como “espere”.
E	Disparador (*Hat*) do tipo Evento/Mensgem
K	Disparador (*Hat*) do tipo Tecla do Teclado
M	Disparador (*Hat*) do tipo Clique do Mouse
S	Disparador (*Hat*) do tipo Início da Aplicação

A nova categoria é referenciada e dois novos blocos *Scratch* são criados. Um para começar a execução da mídia de vídeo e o segundo para parar a execução (Figura 21). Precisam-se construir as rotinas `playMovie:` e `stopMovie:` que não existem. Para isso os novos métodos referenciarão os métodos genéricos da classe de mídias `ScratchMedia` que controlam a execução. Caso a mídia seja de uma subclasse que implemente esses métodos para a execução (como é o caso das mídias de vídeo), os novos métodos cumprirão seu propósito. Um desses novos métodos é mostrado abaixo (Figura 22), o outro método criado possui uma estrutura semelhante.

```
'movies'
        ('play movie %V'                -   playMovie:)
        ('stop movie %V'                -   stopMovie:)
'sound'
        ('play sound %S'                -   playSound:)
        ('play sound %S until done'     s   doPlaySoundAndWait)
        ('stop all sounds'              -   stopAllSounds)
        -
```

Figura 21: trecho do código-fonte do método de classe `blockSpecs`, classe `ScriptableScratchMorph`.

```
ScriptableScratchMorph methodsFor: 'movie ops'
playMovie: movieName

    | snd |
    snd _ self movieNamed: movieName ifAbsent: [^ self].
    snd startPlaying.
```

Figura 22: código-fonte do método `playMovie`, classe `ScriptableScratchMorph`.

Um problema inesperado é o tipo de argumento da definição do bloco. Precisa-se criar um novo seletor para trazer ao bloco a lista das mídias de vídeo para o usuário selecionar. O processo é similar ao bloco de controle de som. Resolver essa questão envolve novamente uma análise a fundo da relação entre as classes para determinar onde as variáveis dos argumentos são definidas. Após a busca, descobre-se que a classe `CommandBlockMorph`, principal classe da categoria `Scratch-Blocks`, explicita a função desses argumentos no método `uncoloredArgMorphFor:`. A modificação no código para incluir consiste em selecionar uma letra ainda não usada para determinar um símbolo e apontar qual conjunto de instruções ela retorna. O novo símbolo criado usa a letra V maiúscula para as mídias de vídeo (Figura 23).

```
$V = code ifTrue: [^ ChoiceOrExpressionArgMorph new getOptionsSelector: #movieNames; choice: ''].
```

Figura 23: código-fonte do novo símbolo criado em `uncoloredArgMorphFor:`, classe `CommandBlockMorph`.

Para finalizar toda a criação das categorias e dos blocos, falta conferir o seletor `#movieNames`, usado no último método citado. O novo método `movieNames` é semelhante ao que retorna para o bloco o nome das mídias de áudio de um objeto. A mudança consiste em um teste condicional para ter como retorno somente o nome das mídias que sejam do tipo vídeo. Esse novo método deve fazer parte de qualquer objeto que tenha uma coleção de mídia, logo, ele é criado na classe `ScriptableScratchMorph`. Por último, como o bloco criado contém um campo para selecionar o argumento, o método `choice:` da classe `ChoiceArgMorph` precisa ser complementado com as referências ao seletor `#movieNames` (Figura 24).

```
ChoiceArgMorph methodsFor: 'accessing'
choice: aSymbol

	| frame palette block doLocalization label |
	frame _ self ownerThatIsA: ScratchFrameMorph.

	((#movieNames = getOptionsSelector) and:
	 [aSymbol = ('record' localized, ScratchTranslator ellipsesSuffix)]) ifTrue: [
	    frame ifNotNil: [^ frame newMovie]].

(…)

	choice _ label _ aSymbol.
	label isUnicode ifFalse: [label _ label asString].
	doLocalization _ (#(costumeNames soundNames movieNames varNamesMenu listVarMenu)
includes: getOptionsSelector) not.

(…)
```

Figura 24: trechos do código-fonte do método `choice:`, classe `ChoiceArgMorph`.

A última referência necessária é no menu de contexto invocado pelo método `presentMenu` do bloco. Adicionar o novo método `movieNames` à coleção finaliza toda a construção da categoria e dos blocos de programação.

4.11 Concluindo o painel de mídias

Voltando ao painel de mídias e comandos, é preciso arrumar a nova aba que dá a relação das mídias de vídeo de um dado objeto. Como visto anteriormente, foi resolvido que a seleção da aba "*Videos*" mostrasse temporariamente a relação das mídias de som, como se fosse uma cópia da aba "*Sounds*". Solucionar essa questão remete a buscar os métodos `costumesPage:` ou `soundsPage:` para ir adiante à criação de uma página para a relação de mídias de vídeo.

Uma nova metodologia para a busca dos métodos foi usada a partir de certo ponto de desenvolvimento do trabalho e, em especial, foi útil nessa tarefa apresentada. Como citado, o ambiente da versão 2.8 do *Squeak* não possui uma busca horizontal de métodos entre as diversas classes. Uma saída foi abrir em modo texto toda a imagem do código-fonte usando o programa Notepad++ (Ho et al., 2010). Dessa forma se fez possível buscar por todas as instâncias de um determinado trecho de código. Essa nova metodologia traz muito mais agilidade ao processo de análise do sistema.

Descobrindo que os métodos buscados fazem parte da classe `ScriptableScratchMorph`, sabe-se onde criar o novo método `moviesPage:` baseado nas classes semelhantes. As principais modificações são: remoção do botão de gravação, condições baseadas na mídia ser um tipo vídeo e a criação de um novo seletor para o botão de importação de mídias, denominado `#importMovie`. Diversas outras mudanças foram feitas, mas todas relacionadas à geometria e disposição dos objetos na tela. Um novo método `importMovie` para o seletor citado anteriormente é também criado na classe (Figura 25).

Um segundo método também relacionado à importação de mídias é modificado para suportar extensões do formato de vídeo. O método tem como nome `importMedia:` e reage de acordo com o tipo de arquivo sendo importado, ou seja, de acordo com a extensão do caminho do arquivo que é recebido como argumento. As inserções no código são mostradas a seguir (Figura 26).

```
ScriptableScratchMorph methodsFor: 'media' stamp:
importMovie
	"Import a new movie from a file and add it to my media."

	| result el newName |
	result _ ScratchFileChooserDialog
		chooseExistingFileType: #movie
		extensions: #(mpg mpeg)
		title: 'Import Movie'.
	result = #cancelled ifTrue: [^ self].

	el _ [MovieMedia new loadFile: result]
		ifError: [:err :rcvr |
			DialogBoxMorph warn: err.
			nil].
	el ifNil: [^ self].

	newName _ self mediaNameFromFileName: result default: 'movie'.
	el mediaName: (UTF8 withAll: (self unusedMediaNameFromBaseName: newName)).
	media addLast: el.
	self updateMediaCategory.
```

Figura 25: código-fonte do método `importMovie`, classe `ScriptableScratchMorph`.

```
ScriptableScratchMorph methodsFor: 'media'
importMedia: fileName

(…)

(#(mpg mpeg) includes: extension) ifTrue: [
		baseName := self mediaNameFromFileName: fileName default: 'movie'.
		[elList addLast: (MovieMedia new loadFile: fileName)] ifError: [^ self]].

(…)

isFirst _ true.
	elList do: [:el |
		el mediaName: (self unusedMediaNameFromBaseName: baseName).
		media addLast: el.
		isFirst ifTrue: [
			isFirst _ false.
			el isSound ifFalse: [self lookLike: el mediaName].
			el isMovie ifFalse: [self lookLike: el mediaName]]].

	self updateMediaCategory.
```

Figura 26: trecho do código-fonte do método `importMedia:`, classe `ScriptableScratchMorph`.

No painel central do programa, denominado painel de mídias de comandos, essas modificações resultam no aparecimento com sucesso da relação das mídias de vídeo de um dado objeto ao ser selecionada a nova aba “*Videos*”.

4.12 Execução

Todos os passos anteriores buscam atingir como resultado a inclusão no Scratch de um novo objeto programável possuidor de uma mídia de vídeo. Após diversas tentativas e abordagens diferentes, conseguiu-se ter mídias de vídeos funcionando da forma esperada. Muitos problemas foram encontrados nessa tarefa, entretanto. Essa seção relata toda a análise feita para a execução da mídia de vídeo.

4.12.1 Estado inicial

Após todas as modificações apontadas nas seções anteriores, chegou-se à seguinte situação: o terceiro botão da biblioteca traz uma caixa de diálogo onde o usuário seleciona um arquivo de mídia de vídeo do sistema de arquivos; um novo objeto programável é criado e sua coleção de mídias contém somente o arquivo de mídia selecionado; sua aparência (*costume*) é definida como sendo a do vídeo; uma instância do objeto é inserida no palco e seu aspecto visual é o quadro (*frame*) número um da coleção de quadros do arquivo de vídeo.

O primeiro quadro do vídeo surge na tela. A quarta aba “Videos”, com a coleção das mídias de vídeo do objeto, mostra apenas um item: a mídia de vídeo escolhida pelo usuário (Figura 27). Tem-se a garantia de que a mídia realmente é um vídeo pelo simples fato dela ser listada nessa aba. O método `isMovie` seleciona da coleção de mídias do objeto somente as que forem *MovieMedia*, de acordo com as extensões suportadas.

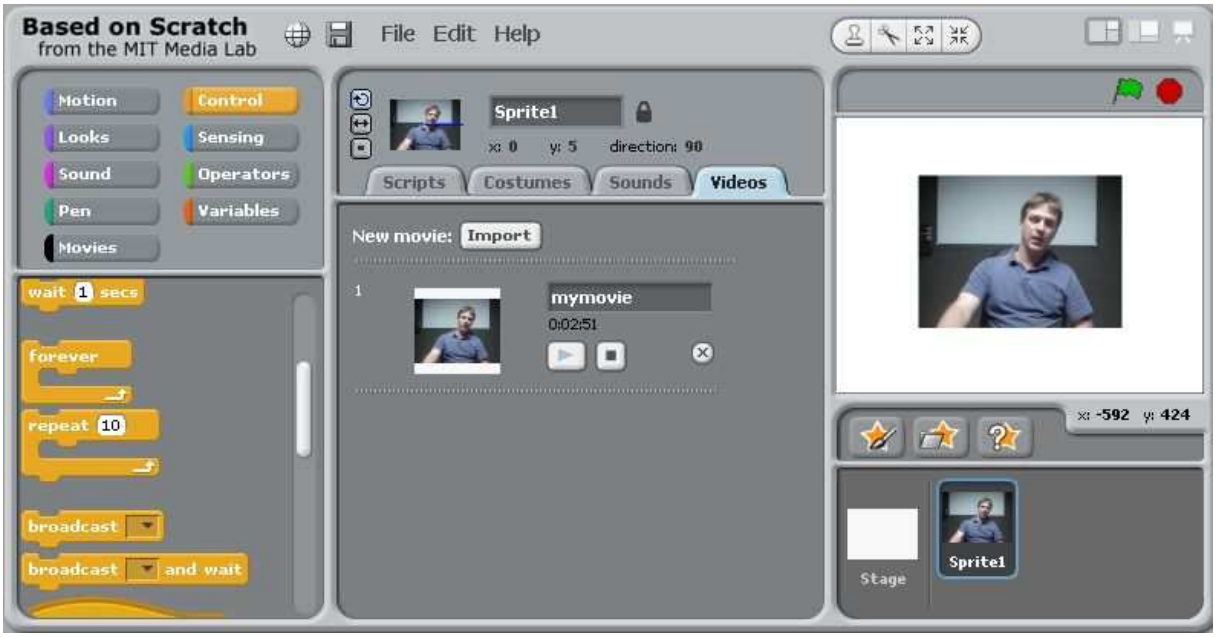

Figura 27: um objeto recém-criado contendo uma mídia de vídeo. A aba "Videos" está em destaque ao centro mostrando as mídias de vídeo que o objeto contém. À direita, no topo, uma instância do objeto no palco.

O objeto no palco também pode ser redimensionado e rotacionado como qualquer outro objeto e é possível programá-lo usando os blocos do *Scratch* (Figura 28). Essa característica pode ser vista como um diferencial do programa para execução e uso de vídeos, considerando que a vasta maioria dos visualizadores ou programas de autoria exibe a forma horizontal tradicional, sem qualquer inclinação ou efeito de imagem em tempo real.

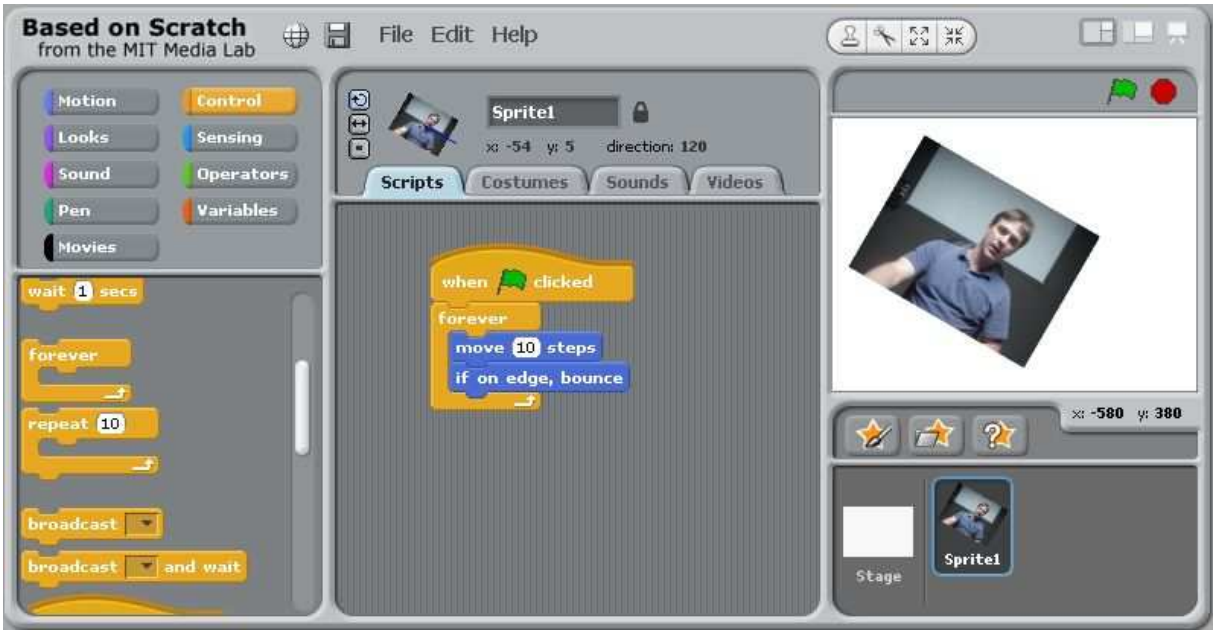

Figura 28: o objeto com mídia de vídeo rotacionado e redimensionado. A rotina definida pelo usuário, ao centro, indica ao objeto para, quando a aplicação iniciar, ele se mover 10 unidades de espaço e retornar quando encostar-se à borda do palco.

Por último, a aparência do objeto pode ser trocada de uma mídia de vídeo para uma mídia de imagem e vice-versa (Figura 29).

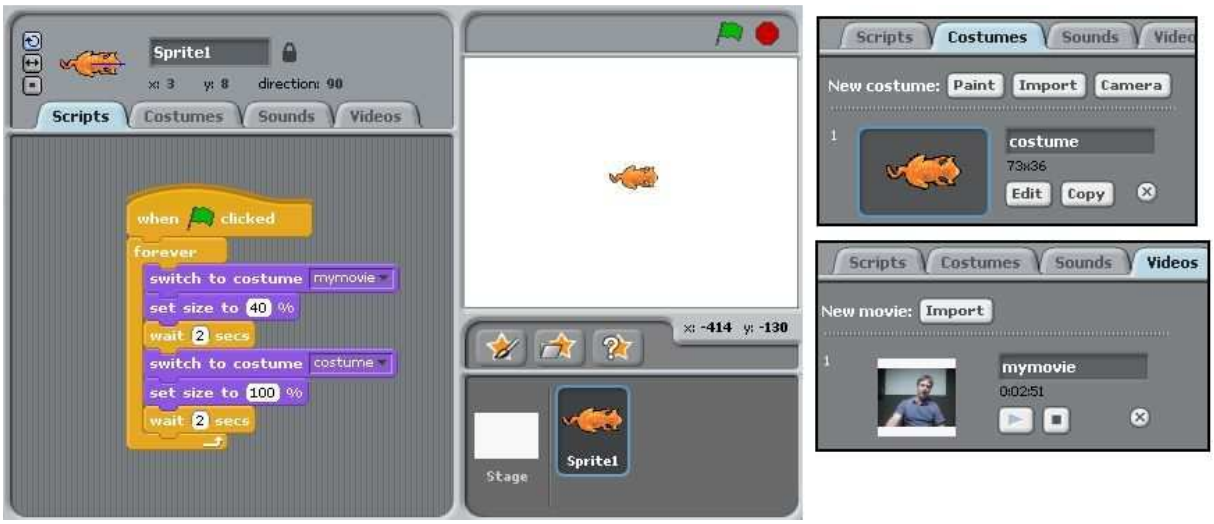

Figura 29: a figura mostra o objeto um (*Sprite1*) com uma rotina definida pelo usuário para trocar sua aparência entre uma mídia de imagem e uma mídia de vídeo a cada dois segundos. No destaque, as abas *Costumes* e *Videos* mostrando as mídias de cada tipo na coleção de mídias do objeto.

4.12.2 Problema de execução

Tendo toda a estrutura necessária, existem duas possibilidades para iniciar o vídeo: o botão "*play*" na aba de mídias e o novo bloco de programação "*play movie*". Os dois convocam o mesmo método `startPlaying`, definido na classe `MovieMedia`, que faz a mídia iniciar sua execução.

No entanto, nenhum dos dois possui o efeito esperado. O áudio inicia sua execução, porém a imagem permanece estática no primeiro quadro do vídeo. Ao inspecionar a mídia, nota-se que os atributos da instância estão sendo atualizados normalmente pelos métodos de controle. O atributo `currentFrame` (o quadro atual) iniciado com o valor "1" avança a valores posteriores, `appearanceChanged` troca do valor "falso" para "verdadeiro"

mostrando que o aspecto mudou e `lastFrameMSecs`, um atributo para cálculos do tempo de execução, marca o ponto de início da execução cada vez que *play* é chamado (Figura 30).

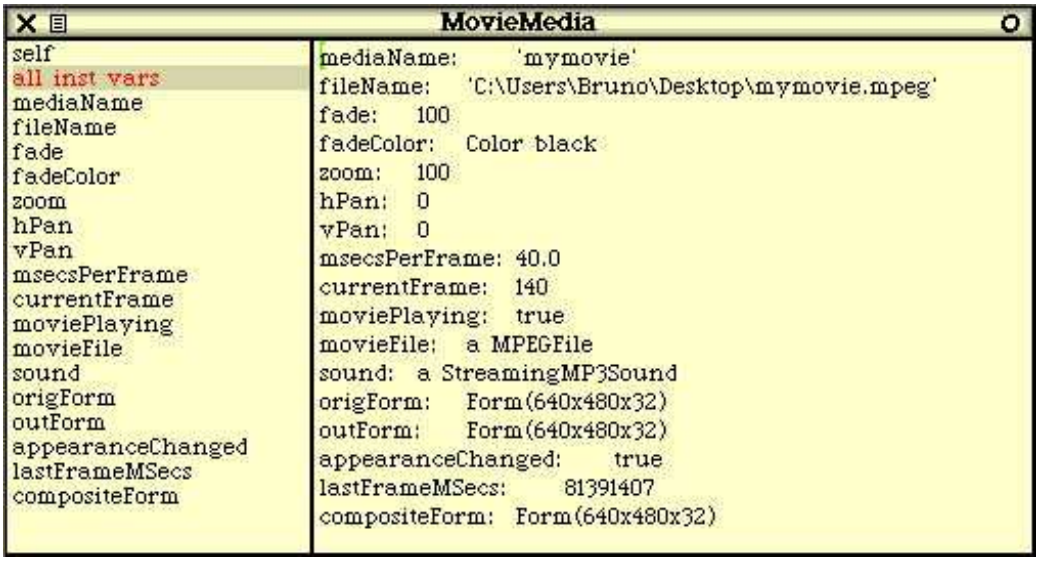

Figura 30: inspeção de `MovieMedia` em tempo de execução. Os atributos `currentFrame`, `moviePlaying`, `appearanceChanged` e `lastrameMSecs` reagem da forma esperada.

4.12.3 Delimitando o problema

Um fato indica um possível ponto de partida para a solução: quando o novo objeto é criado, ele imediatamente muda sua aparência para a imagem do primeiro quadro da mídia. Isso ocorre no método `loadFile:`, na última linha "`showFrame: 1`".

Como visto `loadFile:` é executado no método `addVideoMorph`. Esse método está presente na classe `ScratchFrameMorph` e é chamado quando o botão de adicionar um vídeo (o terceiro botão do painel da biblioteca) é clicado e selecionamos um novo arquivo de mídia desse tipo. Uma instância de `MovieMedia` é criada e o método `loadFile:` é executado com seu argumento sendo o caminho do diretório até a mídia selecionada.

O método, portanto, executa uma vez na sua inicialização e traz "`showFrame: 1`" à tona, executado com sucesso. Uma análise dos outros métodos da classe `MovieMedia` mostra que todos os outros métodos de avanço e execução da mídia também utilizam esse método para mostrar o novo quadro a ser mostrado na tela. A dificuldade está em descobrir por que o método funciona quando chamado na inicialização e por que não funciona durante seus passos de execução.

4.12.4 Rastreamento a partir da inicialização

Em primeiro lugar, `loadFile:` chama o método `basicLoadFile:`. Nesse método a variável privada `movieFile` é iniciada com uma instância da classe de sistema `MPEGFile`. Duas formas são criadas, `origForm` e `outForm`. A primeira é criada a partir da largura e altura da imagem do arquivo de vídeo (`movieFile`) e seu *depth* (o número de bits disponível para as cores). A segunda é uma cópia da primeira usando o método `deepCopy`. É feito o rastreio das variáveis através de impressões em *console* do estado das variáveis. Os resultados indicam que o método está funcionando de acordo com o esperado. Volta-se ao método `loadFile:` para investigar a fundo do método "`showFrame: 1`".

Sabe-se que esse método funciona porque, ao modificar o valor "1" por valores diferentes (como "500" ou "1000") na abertura de um mesmo arquivo de vídeo, o quadro inicial mostrado corresponde ao número estabelecido. Ou seja, o método funciona e mostra o quadro indicado, porém somente na inicialização da classe junto ao carregamento do arquivo. Tira-se disso que as formas e a geração da imagem funcionam, e que o problema deve residir na classe `MovieMedia`.

O método `showFrame:` indica à instância de `MovieMedia` que sua execução deve ir para o parâmetro indicado no método. No caso da inicialização, o parâmetro é "1" para o primeiro quadro (*frame*) no vídeo ser mostrado. Esse método também é invocado a cada passo da execução por diversos outros métodos para fazer o aspecto do vídeo ser modificado de acordo com o quadro atual. Nele, dois outros métodos são executados em relação à variável privada `movieMedia`: o primeiro método se chama `videoSetFrame:currentFrame`, enquanto o segundo se denomina `videoReadFrameInto: origForm`. Após isso, o método chama por último `updateOutForm` que, pelo nome, indica uma boa possibilidade de aí residir o problema.

Os dois métodos relacionados à variável `movieFile` não possuem falhas. A conclusão advém do fato que elas são métodos da classe de sistema `MPEGFile`, utilizada por outras classes do sistema já comentadas anteriormente, como `MovieDisplayMorph`. Nela, arquivos de vídeo funcionam corretamente, porém fora do sistema *Scratch*. Dessa forma, outra conclusão lógica é que o problema não reside no tratamento do arquivo MPEG do *Squeak*, mas sim na manipulação de dados ou referências de `MovieMedia` ou outra classe ligada a ela.

Retornando à “`showFrame: 1`”, examina-se o método `updateOutForm`. Nele, a variável booleana `appearanceChanged` é mudada para o valor “*true*” logo de início, indicando que a aparência foi modificada (ou seja, um *flag*). Um teste é feito: caso o *zoom* seja 100% e as formas `outForm` e `origForm` sejam iguais, executa-se a linha “`origForm displayOn: outForm.`”. Ela simplesmente mostra a imagem dentro da forma criada para gráficos. Como o método é de uma classe do sistema (`Form`) que é classe-pai de muitas outras classes, no máximo podemos ter as referências `origForm` e `outForm` erradas, e não o método. O resto do código de `updateOutForm` não traz outras pistas sobre o problema em questão.

4.12.5 Rastreamento a partir da execução

Como a busca pelas classes invocadas por “`showFrame: 1`” não trouxe resultados, outro ponto de partida é analisar o código-fonte a partir da inicialização da execução, o método `startPlaying`.

Nesse método a impressão de algumas variáveis locais mostram que a execução entra no laço principal através das condições `hasAudio` e `hasVideo` da classe `MPEGFile`. Dentro do laço, o ponto de execução do som e do vídeo é calculado para, logo depois, serem iniciados a partir desse ponto. O método `showFrame:` novamente é chamado com seu parâmetro sendo o número calculado do quadro que se quer mostrar na tela. No final do código desse método a variável `lastFrameMSecs` é atualizada com o valor de tempo corrente do relógio do programa (em milissegundos) e a variável `moviePlaying`, uma *flag*, é acionada para o valor “*true*”.

Novamente um ponto sem saída. Nenhum método declarado em `startPlaying` leva a uma nova possibilidade de solucionar esse problema. Outra alternativa é analisar a partir do método `step`.

4.12.6 Rastreamento a partir do método “step”

O método `step` (ou passo) é um método especial. Presente em várias classes do sistema, ele define rotinas que devem ser feitas em todo o ciclo de execução do programa. Cada classe pode ter o seu método `step`. Todas as instâncias de classes no programa que

tiverem o método `step` são colocadas em uma lista de execução e chamadas uma após outra. A classe `MovieMedia` também possui seu método `step`.

Na classe, o método testa se a instância de `MovieMedia` está referenciando um arquivo de vídeo e se ele está em execução. Caso essas condições sejam satisfeitas, o método avança para o próximo quadro quando estiver no tempo correto. O método é chamado sucessivas vezes pelo laço principal do programa, só avançando quando o quadro do vídeo estiver no tempo dado pelos cálculos internos da classe `MovieMedia`.

Sua estrutura se dá da seguinte maneira: primeiramente a mídia precisa estar rodando, ou seja, a variável `moviePlaying` precisa ter como valor “verdadeiro” (*true*). Após isso, caso o arquivo possua som, parte-se para o método `advanceFrameWithSound` para sincronizar a taxa de avançado com o som. Depois, o novo quadro desejado é calculado a partir do quadro atual mais a diferença entre o tempo no momento que o método é chamado menos o tempo no momento que o último quadro foi mostrado na tela. Por último, o valor desse novo quadro calculado vira argumento para a função `showFrame:`, novamente chamada para mostrar o quadro desejado por meio da função `updateOutForm`, presente no final do seu código-fonte.

Como é possível observar, todas as execuções da classe convergem para o método `showFrame:`, que recebe como parâmetro o quadro que deve ser mostrado na tela, e sua chamada a `updateOutForm`, para atualizar na forma esse novo quadro.

4.12.7 Solução

Tendo o problema sido delimitado no método `updateOutForm`, faz-se uma análise detalhada de todo o seu contexto de execução para procurar por possíveis incoerências entre as referências de seus atributos.

O ponto inicial é a linha “`origForm displayOn: outForm.`”. Essa rotina exibe no segundo elemento o conteúdo do primeiro. Como verificado antes, essas formas são iniciadas genericamente no método de inicialização da classe e depois recebem os dados de imagem provenientes da execução do método `showFrame:`. Uma nova forma de visualizar a questão é necessária.

Uma possibilidade é inspecionar o objeto que contém a mídia. A lógica é que se a mídia possui formas para tratar suas imagens, o objeto `ScratchSpriteMorph` também deve possuir uma forma, afinal é nele que são definidas as funções de redimensionamento e

rotação. No objeto vemos que existe uma forma referenciada pela variável `rotatedForm`. Em contraste, a classe `MovieMedia` possui um total de três formas: `origForm`, `outForm` e `compositeForm`.

É necessário descobrir o conteúdo dessas formas em tempo de execução. Para tanto, inicia-se a execução do programa e em dado momento, pausa-se para realizar uma inspeção dos elementos. Essa inspeção consiste em mostrar de alguma maneira o conteúdo de imagem da forma na tela. Se o conteúdo de uma delas mostrar um quadro diferente do quadro travado número um (o quadro inicial que está sendo exposto no objeto) sabe-se que os quadros estão correndo da maneira certa, porém não estão sendo atualizados corretamente.

O método usado para isso é `displayAt:`, da classe `Form`. Ele é um método auxiliar que mostra na tela a imagem em formato puro, sem uma forma para lhe servir de base (permitindo redimensionamento, movimentação, etc.). Logo de início descobre-se que as três formas de `MovieMedia` estão mostrando o quadro corrente da execução do vídeo. Inspecionando a forma do objeto `ScratchSpriteMorph`, tem-se a confirmação de que há uma falta de atualização entre a imagem da classe da mídia e da classe do objeto.

Analisando a variável `rotatedForm` da classe `ScratchSpriteMorph`, vemos que ela é definida no método de inicialização pela rotina "`rotatedForm := self costumeForm.`". O método `costumeForm` se encontra na classe-pai e devolve como resultado o valor do atributo de mesmo nome `costumeForm` do traje atual do objeto. No caso específico do objeto em análise, esse traje é a mídia de vídeo referenciada por `MovieMedia`. Logo, a conclusão é que a imagem que aparece na tela é a imagem que a forma `compositeForm`, da classe `MovieMedia`, está detendo.

A solução de todo o problema consiste em simplesmente atualizar `compositeForm` com o valor de `outForm` da mesma classe. Valor esse que é constantemente atualizado pelo laço de execução do vídeo. A modificação consiste em uma pequena linha de código, definida por "`outForm displayOn: compositeForm`", no método `updateOutForm` (Figura 31).

```
MovieMedia methodsFor: 'private'
updateOutForm
        "Update my output form."

        | srcRect |

        appearanceChanged _ true.
        origForm ifNil: [^ self].  "do nothing during initialization"

        ((zoom = 100) & (hPan = 0) & (vPan = 0) and:
         [outForm extent = origForm extent])

                ifTrue: [  "optimization: no scaling"
                        origForm displayOn: outForm.
o                       compositeForm ifNotNil: [
                                outForm displayOn: compositeForm]]
                 ifFalse: [  "scaling needed"
                        srcRect _ Rectangle
                                center: (origForm boundingBox center + (hPan@vPan negated))
                                extent: (origForm extent / (zoom asFloat / 100.0)).

                                fillColor: Color black;

        (…)
        self changed.
```

Figura 31: trecho do código-fonte `updateOutForm`, classe `MovieMedia`, com o código criado que resolve o problema da execução das mídias de vídeo em destaque.

A modificação apontada resolve a questão da execução do vídeo e atinge o objetivo proposto de integrar mídias de vídeo ao *Scratch*.

4.13 Outras Modificações

Algumas mudanças e inserções secundárias de código-fonte no programa ficaram de fora do conteúdo relatado nas seções acima. Essa seção tem como finalidade referenciar essas modificações.

4.13.1 Botões da aba “Vídeos”

Três botões são definidos para cada item no painel de mídias de vídeo: um para iniciar a execução, um para parar a execução e um terceiro para deletar a mídia da coleção do objeto *Scratch*. Seus respectivos métodos são `addPlayButton`, `addStopButton` e

`getDeleteButton`. A classe é a `MediaItemMorph` e a alteração consiste em acrescentar um teste condicional para, dado o tipo da mídia, a rotina selecionar o conjunto de instruções adequado. O método `addStopButton` é bastante semelhante ao método `addPlayButton` (Figura 32). Outro método na mesma classe a sofrer modificações foi `buildRightSideMorph`, porém, suas mudanças são simples questões de geometria.

```
MediaItemMorph methodsFor: 'initialization'
addPlayButton

    | h |
    playButton _ ResizableToggleButton2 new
        offForm: (ScratchFrameMorph skinAt: #btn)
        onForm: (ScratchFrameMorph skinAt: #btnPressed).

    media isSound ifTrue: [
        playButton
            icon: (ScratchFrameMorph skinAt: #scratchSoundPlay);
            target: self;
            actionSelector: #startPreviewSound;
            toggleMode: false].

    media isMovie ifTrue: [
        playButton
            icon: (ScratchFrameMorph skinAt: #scratchSoundPlay);
            target: self;
            actionSelector: #startPreviewMovie;
            toggleMode: false].

(…)
```

Figura 32: trecho do código-fonte do método `addPlayButton`, classe `MediaItemMorph`.

4.13.2 Nome do arquivo de vídeo

Para contornar um problema intermediário no carregamento de uma mídia de vídeo para o ambiente do *Scratch*, foi preciso especificar no método `addVideoMorph`, da classe `ScratchFrameMorph`, o caminho para o arquivo da mídia a ser carregado e também o nome da mídia. Essa solução não é a ideal, uma vez que engessa o usuário a editar esse

caminho de dados através da programação ou editar o nome do arquivo para permitir o seu carregamento no programa.

Essa saída temporária foi adotada por ser um problema que tangencia o problema central do carregamento e execução do vídeo. Através dos estudos e análises nesse trabalho, não foi possível descobrir o que ocasiona o problema com o caminho de dados (*path*) da função durante a execução da caixa de seleção do arquivo.

Ao trazer o programa para outro contexto de diretórios, deve-se especificar por meio da edição desse método que seja feita a referência ao diretório onde se encontra o vídeo que se quer carregar, assim como ao nome do arquivo. Para a edição do método, deve-se carregar o arquivo da imagem do *Scratch* (denominada *tg-bruno.image*) no executável do programa, *Scratch.exe* e, após, editar a linha com o caminho de diretório do arquivo. A solução desse problema específico de seleção e carregamento de arquivos, assim como melhorias para a janela de seleção de arquivos, pode ser trabalhada como uma sugestão de trabalho futuro.

5 CONCLUSÃO

Este trabalho apresentou a plataforma *Scratch* como uma nova possibilidade de solução para a criação de objetos interativos de aprendizagem utilizando mídias. Através da expansão da ferramenta para comportar vídeos em seus tipos de mídia suportados, foi mostrada a sua capacidade de trabalhar com mídias de diferentes formatos como imagens, áudio, vídeos e texto de maneira interativa e com alto nível de personalização e controle. Esse objetivo teve como base a pesquisa e revisão de literatura, onde foi visto que um potencial programa para ser usado por pessoas sem base técnica para criação de conteúdo educacional (chamados de ferramentas de autoria) precisa ter uma interface simples de entender e com um grande potencial de desenvolvimento com integração de mídias. Como visto, o *Scratch* apresenta uma sólida base nesses quesitos, advinda da sua proposta de uso e de seu projeto.

A principal contribuição desse trabalho consiste em trazer a idéia do uso do *Scratch* como ferramenta de autoria para criação de objetos de aprendizagem, uma necessidade cada vez maior à medida que mais pessoas têm acesso à tecnologia e à comunicação. Outra contribuição é o acréscimo de mídias de vídeo ao programa, até então não suportadas, sendo essa outra fundamental contribuição por ser um trabalho técnico da área da Ciência da Computação.

O *Scratch* possui como principal frente de pesquisa pela comunidade de educadores a sua aplicação com alunos na sala de aula. Este trabalho sugere a possibilidade de uma nova abordagem, onde a sua proposta de fácil aprendizado, alto potencial e diversificação, possa ser utilizada por educadores na criação dos seus próprios materiais educacionais midiáticos e interativos.

O foco do trabalho foi a avaliação técnica e desenvolvimento para integração de mídias, uma necessidade nas ferramentas de autoria. O aspecto educacional de um estudo de caso para a completa avaliação do seu potencial foi deixado de fora do escopo desse trabalho, sendo uma sugestão de trabalho futuro a aplicação da ferramenta junto a um professor em uma turma de alunos. Outra sugestão de trabalho futuro é a personalização da interface para o educador em estudos de usabilidade para facilitar a tarefa para esse público específico, uma vez que a ferramenta foi desenvolvida tendo como principal alvo os jovens e as crianças.

Outras frentes de desenvolvimento técnico futuro podem incluir: integração de novas extensões de vídeo ou tipos de mídia (como animações em *flash*), tendo como base a execução deste trabalho, ou a expansão da linguagem para diferentes funções e recursos de

programação. Algumas linhas específicas de pesquisa não são recomendadas, como a criação de rotinas e o suporte para plataformas móveis, por já existirem trabalhos em desenvolvimento nessas áreas.

6 REFERÊNCIAS

ADOBE SYSTEMS INCORPORATED. **Adobe Authorware**. Estados Unidos, 2010. Disponível em: <http://www.adobe.com/products/authorware/>.

AGARWAL, M. **Authoring Tool for E-Learning**. Requerimento parcial para graduação em Mestrado de Tecnologia. Índia: Computer Science and Engineering, ITT Bombay, 2009.

ARNEIL, S. HOLMES, M. **Juggling hot potatoes**: decisions and compromises in creating authoring tools for the Web. Canada: University of Victoria, 1999.

BALASUBRAMANIAN, N. WILSON, B. G. **Games and Simulations**. Society of Information Technology and Teacher Education, 2005.

BRENNAN, K. **ScratchEd**: learn, share, connect. Lifelong Kindergarten Group at MIT Media Lab, OHO Interactive, 2009. Disponível em: < http://scratched.media.mit.edu/>

COURSEBUILDER. **CourseBuilder:** The Logistics Of Training. Sydney, 2010. Disponível em: < http://www.coursebuilder.com.au/>

DARYMPLE, D. ROY, K. **ToolBook Instructor**. Training Media Review. 2010. Disponível em: <http://www.tmreview.com/Review.asp?ID=1625>

DECK, D. G. REIMONDO, A. OHSHIMA, Y. MUSA, J. **VideoFlow:** Video and Image Processing. 2004. Disponível em: < http://wiki.squeak.org/squeak/2411>

DIZARD, W. J. **A nova mídia:** a comunicação de massa na era da informação**.** Tradução [da 2ªed.]: Edmond Jorge. Rio de Janeiro: Jorge Zahar Ed. 1998.

DUFFIN, J. W. **Theory for Authoring Tools that Support Teacher Adaptation of Mathlets**. Requerimento parcial para graduação em Doutor de Filosofia. Estados Unidos: UTAH State University, Logan, 2004.

FILDES, J. **Free Tool offers 'easy' coding**. BBC News, 2007. Disponível em: <http://news.bbc.co.uk/2/hi/technology/6647011.stm>

GRAEBIN, C. **Critérios pedagógicos, ambiente educacional, programa curricular e os aspectos didáticos:** critérios relevantes na avaliação de softwares educacionais. Santa Maria: Especialização em Mídias na Educação. UFSM, 2009.

HALF-BAKED SOFTWARE INC. **HalfPotatoes**. Canadá, 2009. Disponível em: < http://hotpot.uvic.ca/>

HO, D. et al. **Notepad++**. 2010. Disponível em: < http://notepad-plus-plus.org/>

INGALLS, D. KAEHLER, T. MALONEY, J. WALLACE, S. KAY, A. **Back to the Future:** The Story of Squeak, A Practical Smalltalk Written in Itself. OOPSLA '97: Proc. of the 12th ACM SIGPLAN Conference on Object-oriented Programming, 1997.

KEARSLEY, G. **Authoring systems in computer based education**. Communications of the ACM, 1982. Disponível em: <http://portal.acm.org/citation.cfm?id=358557.358569>

LIFELONG KINDERGARTEN GROUP. **Scratch:** Imagine, Program, Share. MIT Media Lab, 2007. Disponível em: <http://scratch.mit.edu/>

MORAN, J. M. et al. **Mídias na Educação**: Módulo Introdutório Integração de Mídias na Educação. Brasil: Ministério da Educação, 2009. Disponível em: <http://www.eproinfo.mec.gov.br/webfolio/Mod83230/index.html>

NETOBJECTS, INC. **NetObjects Fusion**. Redwood City, EUA, 2010. Disponível em: <http://netobjects.com/html/fusion_11.html>

RESNICK, M. et al. **Scratch: Programming for All**. Communications of the ACM, 2009. Disponível em: <http://web.media.mit.edu/~mres/papers/Scratch-CACM-final.pdf>

RESNICK, M. KAFAI, Y. MAEDA, J. **A Networked, Media-rich Programming Environment to enhance Technological Fluency at After-School Centers in**

Economically-Disadvantaged Communities. Proposal to National Science Foundation, 2003. Disponível em: <http://web.media.mit.edu/%7Emres/papers/scratch-proposal.pdf>

RUSK, N. RESNICK, M. MALONEY, J. **21st Century Learning Skills**. Lifelong Kindergarten Research Group, MIT, 2008.

SAVI, R. ULBRICHT, V. R. **Jogos Digitais Educacionais:** Benefícios e Desafios. Novas Tecnologias na Educação, CINTED-UFRGS, Dezembro de 2008. Disponível em: <http://www.cinted.ufrgs.br/renote/dez2008/artigos/4b_rafael.pdf>

SUMTOTAL SYSTEMS. **ToolBook 10**. Mountain View, EUA, 2010. Disponível em: < http://www.sumtotalsystems.com/products/toolbook-elearning-content.html>

SUMTOTAL SYSTEMS. **ToolBook 10 New Features**. Mountain View, EUA, 2009. Disponível em: <http://www.idea.se/filer/TB10_new-features.pdf>

TRIVANTIS. **Lectora Pro Suite**. Cincinnati, EUA, 2010. Disponível em: < http://www.trivantis.com/uk/lectora-pro-suite>.

WIKIPEDIA, THE FREE ENCYCLOPEDIA. **Authoring System**. 2010a. Disponível em: <http://en.wikipedia.org/wiki/Authoring_system>

WILL-HARRIS, D. **How to build a better web site using NetObjects**. Dot Com: Siteseeing. [S.l], 2006. Disponível em: < http://efuse.com/Start/dot_com__meet_dot_com.html>

WILLIAMS, A. **LibMPEG3**. 2010. Disponível em: < http://heroines.sourceforge.net/libmpeg3.php>