DA ARTE GENERATIVA AO PENSAMENTO COMPUTACIONAL
Uma análise comparativa das plataformas de aprendizagem

H.C. Sant'Anna[+], F. Gatti[*], J. C. Carmo[*], M. A. Rocha[*], S. R. O. Rangel[§] e V. B. Neves[*]

1. Introdução

Este artigo consiste em uma análise comparativa de plataformas gratuitas orientadas ao desenvolvimento do *Computational Thinking* (CT) ou Pensamento Computacional, tendo a arte computacional como catalisadora e mediadora do processo. Para tanto, o percurso foi organizado em três etapas: primeiramente discutimos os conceitos, práticas e perspectivas do CT. Em seguida, abordamos as noções de arte computacional e arte generativa, suas possibilidades de aproximação com o CT, e apresentamos seis perspectivas de projeto em arte computacional a partir da análise das características de dez plataformas que podem ser utilizadas por artistas iniciantes no tema. Por fim, introduzimos um quadro comparativo elaborado com o intuito de esses artistas iniciantes na escolha de uma plataforma de projeto.

2. Pensamento Computacional

O conceito de Pensamento Computacional (CT) foi definido por Wing (2006) como um conjunto de competências e habilidades tradicionalmente restritas aos profissionais da Ciência da Computação, mas que poderiam ser utilizadas na resolução criativa de problemas de qualquer área do conhecimento. Esse conceito tem sido abordado em áreas cada vez menos ligados à Informática (Resnick et al, 2009) e em outros níveis para além do ensino superior (Stephenson & Barr, 2011), dada a onipresença dos computadores nos mais diversos segmentos da atividade humana (Greenfield, 2006) e, principalmente, pelo crescente número de problemas computacionais decorrentes dessa intensificação no uso desses dispositivos.

Do ponto de vista pedagógico, Brennan (2011) e Resnick & Brennan (2012) propõem que as oportunidades de aprendizado dos princípios do CT poderiam ser estruturadas a partir de *conceitos*, *práticas* e *perspectivas computacionais*: os *conceitos* seriam aqueles utilizados pelos aprendizes enquanto programam – sequências, laços, paralelismo, eventos, condicionais, operadores e dados; as *práticas* seriam desenvolvidas pelos aprendizes enquanto programam – ser iterativo e incremental, testar e depurar, reusar e se apropriar do trabalho de outros, abstrair e modularizar; e as *perspectivas* estariam relacionadas à emergência de capacidades do aprendiz para se expressar, conectar-se a outros aprendizes e para fazer perguntas sobre o mundo que o cerca a partir do empoderamento oferecido pelo domínio da Computação e das suas ferramentas.

Entidades como a *Association for Computing Machinery* – ACM[1] e Sociedade Brasileira de Computação – SBC (SBC, 2011), empresas como o Google[2] e sites sem fins lucrativos

[+] Doutorando em Psicologia, professor do Departamento de Desenho Industrial da Universidade Federal do Espírito Santo – UFES e Coordenador do Laboratório de Psicologia da Computação – LabPC/UFES.
[*] Estudante do curso de graduação em Desenho Industrial da UFES e pesquisador júnior do LabPC/UFES.
[§] Bacharel em Comunicação Social pela Universidade Estácio de Sá do Espírito Santo e pesquisadora do LabPC/UFES.
[1] ACM CSTA - Computer Science Teachers Association. Disponível em http://csta.acm.org. Acesso em 10 de setembro de 2012.

como *Khan Academy*[3] e *Codecademy*[4] realizam esforços para criar oportunidades de aprendizagem dos conceitos citados anteriormente por qualquer pessoa. O desafio dessas iniciativas, em termos gerais, consiste em recomendar plataformas com seus respectivos guias de primeiros passos para os iniciantes, oferecer atividades e projetos motivadores além de criar uma comunidade ativa e receptiva onde os usuários possam aprender uns com os outros, respeitando os diferentes perfis de aprendizes.

3. Arte: computação e generatividade

A pesquisa e o desenvolvimento em arte computacional a partir da segunda metade do séc. XX apresentaram-se como terrenos férteis tanto para a exploração das possibilidades estéticas quanto para o aprendizado dos princípios do Pensamento Computacional, seja na categoria das "imagens de síntese" (Venturelli e Burgos, 1999) ou pela exploração da "estética generativa" conforme descrita por Max Bense (1968). Na década de 1960, enquanto pioneiros como Georg Ness, Frieder Nake, Vera Molnar, Michael Noll, Hiroshi Kawano e Manfred Mohr experimentaram a criação artística baseada em algoritmos (Lieser 2010), Seymour Papert e seus colaboradores do Instituto de Tecnologia de Massachusetts (MIT) investigaram como crianças poderiam aprender os fundamentos da programação de computadores desenvolvendo arte computacional com a linguagem LOGO (Solomon, 1976). Apesar dos objetivos, motivações e ferramentas distintas, artistas e crianças pioneiras no uso do computador como meio de expressão artística compartilhavam o interesse por dois processos que, segundo Wing (2008), representariam a essência da Computação: a estruturação de operações computacionais na forma de algoritmos (*abstração*) e a consequente construção desses algoritmos nos computadores (*automação*).

As manifestações de arte computacional que se enquadram na categoria das imagens de síntese não diferem daquelas baseadas em programas especialistas apenas pelas escolhas das ferramentas de trabalho do artista – algoritmos e linguagens de programação versus paletas digitais e recursos de modelagem, respectivamente – mas também pela cessão do controle parcial ou total do resultado estético para o sistema que o realiza. Galanter (2003) sugere que essa forma estética, na qual os algoritmos são intencionalmente elaborados de maneira a proporcionar certo grau de autonomia à obra, pertenceria ao domínio da *arte generativa*.

Pode-se sugerir que as manifestações da arte computacional seriam espaços privilegiados para a exploração dos princípios relacionados ao CT, numa perspectiva muito similar à que originou a linguagem LOGO (Papert & Solomon, 1971). O desenvolvimento da capacidade de abstração lógico-matemática do artista e o uso que ele faz dos recursos das linguagens de programação para automatizar abstrações de finalidade estética pertencem ao mesmo universo cognitivo das habilidades e competências típicas de um profissional da Ciência da Computação, de forma que as ferramentas utilizadas por este também poderiam ser adotadas como ponto de partida para os artistas computacionais iniciantes.

---

[2] http://www.google.com/edu/computational-thinking/. Acesso em 10 de setembro de 2012.
[3] http://www.khanacademy.org/cs. Acesso em 10 de setembro de 2012.
[4] http://www.codecademy.com. Acesso em 10 de setembro de 2012.

4. Plataformas de aprendizagem

O processo de aprendizagem dos princípios da Computação por artistas iniciantes no tema começaria pela escolha de uma plataforma que permita a exploração dos conceitos, práticas e perspectivas do CT. Por *plataforma* entendemos a distribuição completa de uma determinada linguagem de programação, incluindo as ferramentas para edição, correção, intepretação ou compilação de código[5].

No presente estudo, selecionamos dez plataformas com base nos seguintes critérios: a) enquadrar-se na definição de *plataforma* mencionada; b) ser gratuita, de código aberto ou possuir licença de uso flexível; c) apresentar recursos para a criação de imagens de síntese de natureza generativa; d) ser de *download* e instalação simples, com o mínimo de dependências adicionais. Uma vez selecionada, cada plataforma foi analisada a partir de oito dimensões:

1) Princípios pedagógicos subjacentes e oportunidades para a discussão dos conceitos, práticas e perspectivas do CT, caso declarados formalmente pelos autores e mantenedores da plataforma;
2) Natureza e características da linguagem de programação utilizada, partindo dos mecanismos descritos por Abelson & Sussman (1996) para a expressão de ideias por meio de linguagens de programação – expressões primitivas, meios de combinação e meios de abstração.
3) Possibilidades de projetos envolvendo computação física, especialmente pelo suporte oferecido a placas controladoras Arduino ou similares;
4) Suporte ao desenvolvimento para *tablets* e celulares;
5) Disponibilidade nos diferentes sistemas operacionais, com suporte mínimo a Linux, Microsoft Windows ou Apple Mac OS X;
6) Tipos de mídias suportadas – geração e importação de imagens, áudio e vídeo;
7) Existência de documentação online, comunidades, fóruns, ambientes de apoio ao processo de aprendizagem e demais recursos de tutoria multimídia;
8) Bibliografia disponível preferencialmente em português e inglês.

O processo de seleção chegou às plataformas App Inventor, Context Free, DrawBot, Impromptu, Nodebox, Processing, Pure Data, Scratch, ShoeBot e vvvv, dentre as quais seis serão discutidas em detalhe no presente trabalho e as demais terão suas análises em cada uma das dimensões apresentadas no quadro comparativo final, dadas as limitações de espaço para o texto. Cabe esclarecer que as plataformas detalhadas foram escolhidas apenas por exemplificarem perspectivas distintas no universo das ferramentas investigadas.

4.1 Scratch

O Scratch[6] é ambiente programação multimídia em rede, desenvolvido originalmente visando aumentar a fluência tecnológica de jovens e crianças frequentadoras de espaços que oferecem atividades fora do horário escolar em comunidades de risco social e econômico (Resnick et al, 2003). Além do ambiente, disponível para Linux,

[5] Segundo Dale e Lewis (2010, p.415), compiladores são programas que traduzem uma linguagem de alto nível em código de máquina. Já interpretadores são programas que tem como entrada um programa em uma linguagem de alto nível e direciona o computador a executar as especificadas em cada instrução (idem, p.419).

[6] http://scratch.mit.edu. Acesso em 10 de setembro de 2012.

Microsoft Windows e Apple Mac OS X, o Scratch está integrado a uma comunidade *online* com projetos compartilhados pelos próprios usuários, tutoriais multimídia, guias de primeiros passos para os aprendizes e uma base de informações de apoio aos professores que desejarem utilizar o aplicativo. Tanto o site quanto o conteúdo de apoio didático está disponível em diversos idiomas, incluindo o português.

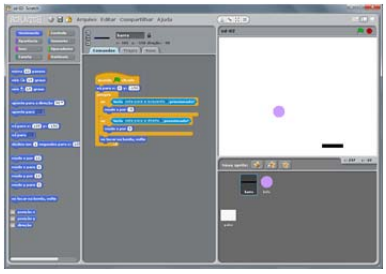

Fig. 1-A: Interface do Scratch

Fig. 1-B: Programa escrito com blocos

O Scratch é uma ferramenta essencialmente pedagógica, relacionada conceitualmente e institucionalmente à linguagem LOGO e outras iniciativas construcionistas (idem, p.5). Foi planejada de forma a explorar formalmente os conceitos, práticas e perspectivas do CT na educação de tecnológica de crianças e jovens (Brennan, 2012). Os elementos primitivos da linguagem, situados na parte esquerda da interface (Fig. 1-A), são blocos organizados conforme sua função – movimento, aparência, som, caneta de desenho, controle, sensores, operadores e variáveis – e os programas (parte central da interface) são escritos por meio do encaixe apropriado dos blocos que controlam os objetos na tela de saída, chamada *palco* (canto superior direito). As cores e desenhos dos encaixes dos blocos (Fig. 1-B) sugerem as regras de combinação, facilitando o aprendizado das relações entre os elementos primitivos.

A produção de imagens de síntese no ambiente Scratch, sejam animadas ou estáticas, é feita a partir da combinação dos blocos das categorias caneta, controle e operadores. O ambiente também pode ser utilizado em projetos de computação física com os kits de robótica LEGO Mindstorms NXT[7], placas controladoras Arduino[8] e PicoCricket[9].

4.2 App Inventor

Criado originalmente pelo Google e atualmente mantido pelo MIT Center for Mobile Learning, a App Inventor[10] é uma linguagem de programação visual dedicada ao projeto de aplicativos para celulares Android (Gray et al, 2012). A App Inventor é executada a partir de um navegador comum da Web nos três sistemas operacionais investigados. A única dependência externa é o Java Runtime Environment – JRE, precisando apenas ser instalada no Microsoft Windows. O computador precisa estar necessariamente conectado à Internet para utilizar o App Inventor, apesar dos aplicativos desenvolvidos poderem ser utilizados no modo *off-line*. A interface, executada diretamente no navegador, possui duas janelas: na primeira (Fig. 2-A), o usuário organiza os componentes do aplicativo em telas (*screens*), gerencia os arquivos

[7] http://www.mindstorms.com. Acesso em 15 de agosto de 2012.
[8] http://www.arduino.cc. Acesso em 15 de agosto de 2012.
[9] http://www.picocricket.com/. Acesso em 15 de agosto de 2012.
[10] http://appinventor.mit.edu/. Acesso em 16 de agosto de 2012.

e exporta o projeto para o celular ou simulador; na segunda (Fig. 1-B), o editor de blocos, o usuário define a lógica e a programação propriamente dita do aplicativo.

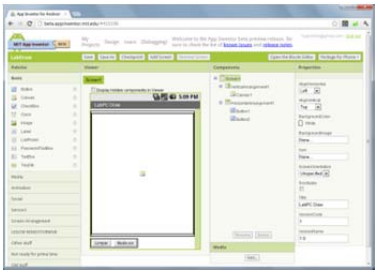

Fig. 2-A: Componentes e Telas

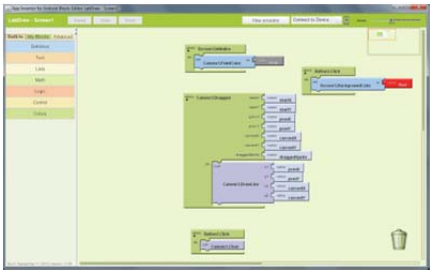

Fig. 2-B: Editor de Blocos

Assim como no ambiente Scratch, os elementos primitivos da linguagem são blocos organizados conforme sua função – declaração de variáveis, tratamento de texto, listas, funções matemáticas, operadores lógicos, estruturas de controle e definições de cores – que são combinados por meio de encaixes particulares.

No que tange ao desenvolvimento de projetos que envolvem computação física, a App Inventor conta com componentes para a comunicação via *Bluetooth* com kits de robótica LEGO Mindstorms NXT ou controladoras Arduino.

Aqueles interessados em aprender ou ensinar com apoio da App Inventor podem partir do site da linguagem, cujo conteúdo está organizado em três áreas: 1) *Teach*, direcionado a educadores; 2) *Explore*, orientado a aprendizes; e 3) *Invent*, que carrega o ambiente de programação. Não foram encontradas referências em português para a App Inventor além de alguns blogs brasileiros e vídeos que oferecem introduções à linguagem. Os mantenedores do projeto editaram um livro em inglês intitulado "*App Inventor: Create your own Android Apps*" (Wolber et al, 2011), que apresenta uma visão geral da linguagem e aborda conceitos da Ciência da Computação por meio de design de jogos e de aplicações gráficas 2D. A página de recursos para educadores[11] do projeto contém outros títulos lançados recentemente que também servem de introdução à linguagem.

4.3 Context Free

Criado por Chris Coyne, Mark Lentczner e John Horigan, Context Free[12] é um ambiente para a edição, rasterização e exploração de conjuntos de regras não determinísticas para a produção de imagens usando uma linguagem denominada *Context Free Design Grammar* - CFDG. Essa linguagem é similar a uma gramática formal livre de contexto, apresentando regras de produção, símbolos não terminais e símbolos terminais para a produção de sentenças, com a adição de operadores de transformação ou regras de ajuste (Christensen, 2009). O ambiente possui o código-fonte aberto e está disponível para Linux, Microsoft Windows XP ou superior e Apple Mac OS X. Para máquinas rodando o versões desatualizadas do Microsoft Windows, a instalação do Microsoft .Net Framework versão 4 ou superior pode ser necessária. A interface do ambiente é

[11] http://tinyurl.com/livroappinventor. Acesso em 16 de agosto de 2012.
[12] http://contextfree.org. Acesso em 17 de agosto de 2012.

dividida de tal forma que a digitação do código acontece à esquerda da saída gráfica do ambiente (Fig. 3).

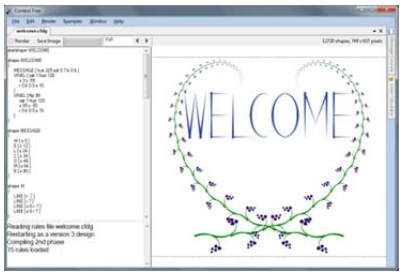

Fig. 3: Editor de código e saída gráfica.

Na linguagem CFDG, há diversos grupos de elementos primitivos[13]: variáveis, funções (matemáticas, de animação, especiais e de configurações das definições da imagem), curvas, estruturas de controle (laços, condicionais), operadores de transformação e clonagem, além dos símbolos terminais *quadrado* (SQUARE), *círculo* (CIRCLE) e da diretiva *startshape* que define o símbolo raiz do desenho. Os meios de combinação da linguagem articulam os elementos primitivos em grupos de instruções rotuladas que definem os símbolos não terminais, permitindo a construção de ideias mais complexas pela abstração de ideias mais simples.

A prática da programação no Context Free começa pela criação do símbolo raiz (*startshape*) que será expandido por meio das regras de produção até encontrar um símbolo terminal. Colocando de outra forma, as imagens são geradas por substituição: símbolos não terminais são definidos nos termos de outros símbolos e a renderização procede pela substituição em árvore dos símbolos mais abstratos em direção aos mais concretos. O processo de desenho é recursivo, podendo gerar um número infinito de terminais, de forma que o Context Free termina a execução das regras de produção sempre que as primitivas tornam-se muito pequenas para serem vistas.

A linguagem não oferece suporte a outros tipos de mídia além de imagens nem apresenta interfaces para projetos de computação física. O foco da ferramenta é a geração de imagens de síntese, inclusive em resoluções extremamente altas, partindo da lógica das gramáticas livre de contexto.

Os conteúdos do site do projeto, disponíveis exclusivamente em inglês, consistem em uma introdução detalhada à linguagem, uma galeria de programas CFDG enviados pelos usuários e um fórum de discussão. Os autores da linguagem publicaram um livro em inglês intitulado "*Community of Variation*" (Lentczner, 2008) que apresenta os melhores trabalhos desenvolvidos por 29 artistas nos dois primeiros anos e meio do projeto. Até o momento do fechamento desta pesquisa, este foi o único título encontrado sobre a plataforma e praticamente não há citações em língua portuguesa ou em sites brasileiros sobre a linguagem.

4.4 Nodebox

O ambiente NodeBox[14] foi construído a partir do DrawBot[15], mantendo a linguagem Python como base dos scripts e incorporando novos recursos. Além das funções de desenho, o NodeBox possui bibliotecas organizadas nas categorias *conhecimento* (bancos de dados, processamento e mineração de textos, grafos, funções linguísticas,

[13] http://tinyurl.com/cfdghowto. Acesso em 16 de agosto de 2012.
[14] http://nodebox.net. Acesso em 15 de setembro de 2012.
[15] http://drawbot.com. Acesso em 15 de setembro de 2012.

busca e recuperação de conteúdos na Web, redes semânticas), *pixels* (processamento e captura de imagens, áudio e vídeo), *curvas* (manipulação e edição de vetores), *sistemas* (simulação do comportamento de organismos vivos e sistemas naturais), *design* (construção de grids e manipulação avançada de cores), *tipografia* (simulação de caligrafia e manipulação de glifos de fontes digitais) e *tangíveis* (uso de interfaces e protocolos de comunicação diversos). Os responsáveis pelo ambiente enfatizam sua utilidade em projetos de arte generativa, descrevendo possibilidades que empregam algoritmos genéticos, sistemas emergentes, processamento de linguagem natural, entre outros (De Smedt, Lechat & Daelemans, 2011), diversificando as perspectivas do CT a serem desenvolvidas pelos aprendizes.

O ambiente NodeBox está disponível apenas para Apple Mac OS X e seu código-fonte é aberto. A interface também é organizada em uma área de código e uma saída gráfica (Fig.4). Embora haja suporte para projetos que envolvam computação física, utilizando a biblioteca do protocolo Open Sound Control – OSC[16], não é possível gerar aplicações para celulares ou *tablets* utilizando o NodeBox.

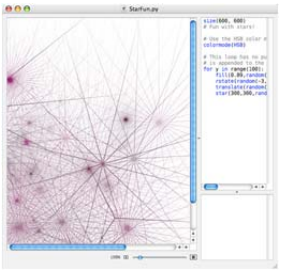

Fig. 4: Editor de código e saída gráfica.

Quando comparado ao DrawBot, o NodeBox dispõe de uma estrutura mais completa de apoio aos aprendizes: o site do projeto oferece uma introdução teórica aos fundamentos da arte generativa, tutoriais ilustrados em cinco categorias (básicos, dados, estratégias, específicos e avançados), a referência completa da linguagem com exemplos, uma galeria com trabalhos enviados pela comunidade de usuários e um fórum de discussão. Em outro site[17] indicado pelos mantenedores do projeto, é possível encontrar uma série de artigos em inglês e holandês que exploram aplicações variadas do NodeBox. O acervo dos dois sites compensa, em certa medida, a ausência de livros dedicados ao ambiente.

4.5 Processing

Criado por Casey Reas e Ben Fry sob orientação de John Maeda no MIT Media Lab, Processing[18] integra uma linguagem de programação baseada na linguagem Java[19], um ambiente de desenvolvimento e uma metodologia de ensino. Segundo os autores, a ferramenta foi criada para ensinar os fundamentos da programação de computadores em um contexto visual orientado à geração e processamento de imagens, podendo ser utilizada por estudantes, artistas, designers e pesquisadores tanto como para esboçar e prototipar ideias quanto para o desenvolvimento de aplicações completas (Reas & Fry, 2007). O código-fonte do ambiente é aberto e a distribuição disponível para os sistemas operacionais Linux, Microsoft Windows e Apple Mac OS X, tendo como única dependência externa o JRE. A interface do ambiente de desenvolvimento consiste em um editor de código, um console para verificação de erros e saída textual dos

[16] http://opensoundcontrol.org/. Acesso em 16 de setembro de 2012.
[17] http://research.nodebox.net/index.php/Abstracts. Acesso em 16 de setembro de 2012.
[18] http://processing.org. Acesso em 17 de agosto de 2012.
[19] http://java.com. Acesso em 17 de agosto de 2012.

programas escritos na linguagem (Fig. 5-A) e uma janela gráfica de tamanho variável que é aberta durante a execução dos programas (Fig. 5-B).

A linguagem se propõe a facilitar o aprendizado de técnicas de computação gráfica e de interação humano-computador por meio de elementos primitivos para desenho vetorial e matricial (2D e 3D), processamento de imagens e cores, eventos de teclado e mouse, comunicação em rede e programação orientada a objetos. Há uma série de bibliotecas adicionais criadas pela comunidade[20] que diversificam as possibilidades de uso da linguagem em áreas como computação física, processamento e geração de áudio e vídeo, visão computacional, simulações físicas, protocolos de comunicação, além da possibilidade de exportar os programas escritos em Processing como aplicações nativas para celulares e *tablets* Android.

Fig. 5-A: Editor de Código e console

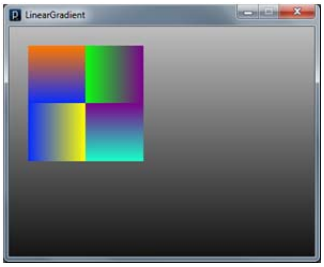

Fig. 5-B: Saída gráfica

No que concerne aos elementos primitivos, meios de combinação e de abstração da linguagem, a Processing se apresenta como uma versão simplificada da linguagem Java. Programadores experientes podem escrever códigos complexos utilizando todas as boas práticas do Java em Processing ao mesmo tempo em que iniciantes podem explorar os métodos e funções com o apoio de atalhos e tratamentos no código que o editor realiza de forma transparente, minimizando a ocorrência de erros.

Os códigos-fonte escritos em Processing, denominados *sketches*, podem ser compostos por uma sequência simples de funções da linguagem que são executadas uma única vez ou podem seguir uma estrutura especial que permite que os programas rodem continuamente: uma função de configuração *setup()* invocada no início da execução, um laço de desenho *draw()* invocado conforme a taxa de quadros por segundo, uma função de atualização de variáveis e do estado do sistema *update()* com a mesma taxa de atualização do *draw()*, e uma série de gerenciadores de eventos para teclado, mouse ou outras interfaces de entrada e saída de dados. A escrita de código orientado a objetos é opcional, podendo ser utilizada ou não conforme o interesse do aprendiz e demanda dos projetos. Nesse sentido, a linguagem Processing oferece uma situação para o desenvolvimento conceitos do CT conforme o aprofundamento do aprendiz nas práticas e perspectivas abertas pelo uso do ambiente.

O site do projeto é o principal ponto de partida para os interessados em aprender a linguagem, concentrando a referência oficial da linguagem, tutoriais ilustrados sobre conceitos básicos ou por tópicos, uma lista atualizada de livros específicos sobre o tema, uma galeria com projetos selecionados pelos mantenedores e um fórum de

[20] http://processing.org/reference/libraries/. Acesso em 17 de agosto de 2012.

discussão para os usuários. No período de levantamento de dados desta pesquisa, havia 17 livros disponíveis sobre a linguagem em inglês, com um deles editado em português[21]. Dentre todas as linguagens investigadas, esta apresenta a comunidade mais ativa, com o maior número de referências e a maior diversidade de usos, de modo que outras iniciativas como NodeBox e Context Free manifestam terem sido influenciadas pela linguagem Processing e buscam oferecer uma experiência semelhante por meio de outros conceitos, práticas e perspectivas do CT.

4.6 Pure Data

A linguagem de programação visual Pure Data[22] foi criada por Miller S. Puckette no final dos anos 1990 e faz parte da genealogia das linguagens de programação musical derivadas do Max, desenvolvido originalmente no IRCAM (Puckette, 1996). A linguagem Pure Data possui código aberto e a distribuição está disponível para os sistemas Linux, Microsoft Windows 2000 ou superior e Apple Mac OS X. Um dos principais usos da linguagem está no processamento de mensagens MIDI entre computadores e instrumentos musicais que suportam o protocolo.

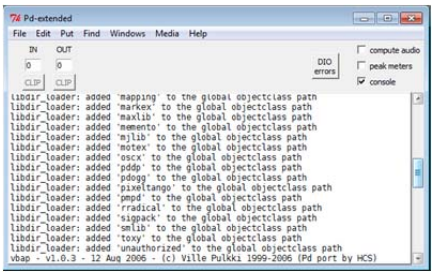

Fig. 6-A: Registro das operações

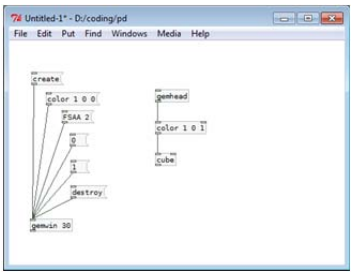

Fig. 6-B: Janela de código

A interface do ambiente é composta por um conjunto de janelas, sendo uma dedicada ao registro das operações (Fig. 6-A) e as demais utilizadas para comportar o código (Fig. 6-B) ou para dar saída a gráficos OpenGL.

Os elementos primitivos da linguagem estão divididos em três grupos: *módulos* (objetos, mensagens, números e símbolos), *elementos de interface* (botões, gatilhos, controles deslizantes, telas), *dispositivos de visualização de dados* (gráficos e mostradores) e um objeto específico para criar *cadeias* de outros objetos. Os meios de combinação são baseados no roteamento de sinais (*dataflow programming*) que são transmitidos através dos elementos primitivos conectados em sequência, de forma similar à operação de equipamentos do universo da música eletrônica. Abstrações são possíveis na linguagem pela criação de objetos caixas-pretas que transformam entradas em saídas sem revelar seus roteamentos internos.

A programação por roteamentos apresenta resultados em tempo real para o usuário, de forma que as práticas do CT na linguagem Pure Data estão alinhadas à vertente do *live coding* (Collins, 2007) ou programação ao vivo, onde o artista-programador cria a experiência visual ou auditiva na presença do público por meio da composição de códigos em tempo real. O suporte nativo a mensagens MIDI e ao protocolo OSC

[21] http://tinyurl.com/processingpt. Acesso em 18 de agosto de 2012.
[22] http://puredata.info. Acesso em 20 de agosto de 2012.

possibilita a utilização da linguagem em projetos de computação física e interações com dispositivos móveis[23].

A produção de imagens de síntese na linguagem Pure Data é baseada na biblioteca OpenGL e depende do pacote *Graphics Environment for Multimedia* – GEM, incluso na distribuição padrão (*extended*) do ambiente. A adição dos recursos do pacote GEM permite a criação de objetos Pure Data geométricos ou *geos* (primitivas 2D e diversos sólidos 3D) e não geométricos ou *non-geos*, responsáveis pela transformação e manipulação dos *geos*, controle da janela e das propriedades gráficas.

O site da Pure Data oferece uma série de referências em inglês para os iniciantes, incluindo a documentação da linguagem, galeria de projetos, publicações, links para listas de discussão, comunidades online e outras formas de interação com os demais usuários. Dentre as publicações específicas, o livro distribuído gratuitamente por Miller Puckette intitulado *The Theory and Technique of Electronic Music* (2007) pode ser utilizado como texto introdutório. A iniciativa FLOSS[24] elaborou um tutorial em inglês da linguagem contendo uma seção específica sobre o pacote GEM.

5. Discussão: perspectivas para projetos de arte computacional

As plataformas analisadas no presente artigo representam seis diferentes perspectivas para projetos de arte computacional e generativa:

1) Uso de dispositivos móveis como meio – App Inventor e Processing;
2) Geração de imagens por regras de produção e gramáticas – Context Free;
3) Inspiração na natureza como pesquisa de possibilidades estéticas – NodeBox;
4) Programação ao vivo (*live* coding) – Pure Data, Impromptu e vvvv;
5) Ambientes de programação para crianças e jovens – Scratch e App Inventor;
6) Linguagens simples associadas a ambientes minimalistas orientados à produção de imagens – NodeBox, DrawBot e ShoeBot;

Cada uma das perspectivas introduz diferentes problemáticas pedagógicas para o desenvolvimento das habilidades e competências que compõem o Pensamento Computacional, de forma que a escolha de uma entre as plataformas disponíveis consiste em uma decisão orientada por múltiplos fatores – perfil e interesses do aprendiz, demandas do projeto, disponibilidade de referências e documentação, sistemas operacionais suportados e, principalmente, questões estéticas da arte computacional e generativa a serem exploradas.

A Tabela 1 foi elaborada visando auxiliar o artista iniciante no processo de escolha, comparando horizontalmente as plataformas nas oito dimensões de análise empregadas ao longo deste levantamento. Embora nem todas as plataformas presentes no quadro tenham sido discutidas no presente trabalho, todas se inscrevem em pelo uma das seis perspectivas citadas acima.

---

[23] A biblioteca gratuita *libpd* disponibiliza os recursos de síntese de áudio da Pure Data em celulares e *tablets* baseados nos sistemas Android, iOS e Maemo, apesar de ainda não oferecer suporte à geração de imagens. Disponível em http://libpd.cc/. Acesso em 15 de setembro de 2012.

[24] http://en.flossmanuals.net/pure-data/. Acesso em 12 de setembro de 2012.

Devido às limitações de espaço do presente estudo, não foi possível abordar as diferenças na aplicação dos conceitos do CT de acordo os elementos primitivos, meios de combinação e de abstração de cada linguagem. Pretendemos detalhar essas diferenças em trabalhos futuros e sugerir estratégias para o acompanhamento e avaliação do desempenho do aprendiz ao longo do processo.

Tabela 1 – Comparativo das plataformas segundo as dimensões de análise

	Princípios Pedagógicos	Elementos primitivos, meios de combinação e abstração	Suporte a projetos de computação física	Suporte a *tablets* e celulares	Disponibilidade	Tipos de mídias	Documentação e recursos de aprendizagem	Bibliografia em inglês ou português
App Inventor	Criação de aplicativos	Componente e Blocos	LEGO Arduino	Sim	Web (Java)	Imagens, áudio e vídeo	Sim	Inglês
Context Free	Imagens de síntese	Gramática livre de contexto	Não	Não	Linux Windows Mac OS X	Imagens	Sim	Inglês
DrawBot	Imagens de síntese	Python	Não	Não	Mac OS X	Imagens	Sim	Inglês
Impromptu	Animações 3D e música	Scheme	OSC MIDI	Não	Mas OS X	Imagens, áudio e vídeo	Sim	Inglês
NodeBox	Imagens de síntese, animações 2D/3D e interatividade	Python	OSC TUIO WiiMote	Não	Mac OS X	Imagens, áudio e vídeo	Sim	Inglês
Processing	Imagens de síntese, animações 2D/3D e interatividade	Java	LEGO Arduino OSC	Sim	Linux Windows Mac OS X (Java)	Imagens, áudio e vídeo	Sim	Inglês e Português
Pure Data	Animações 2D/3D, música e instalações	Programação Dataflow	OSC MIDI	Apenas áudio via libpd	Linux Windows Mac OS X	Imagens, áudio e vídeo	Sim	Inglês
Scratch	Imagens de síntese, música animações 2D e interatividade	Programação com blocos	Crickets LEGO	Não	Linux Windows Mac OS X	Imagens e áudio	Sim	Inglês e Português
ShoeBot	Imagens de síntese	Python	Não	Não	Linux Mac OS X	Imagens	Sim	Inglês
vvvv	Animações 2D/3D, música e instalações	Programação Dataflow	OSC MIDI	Não	Windows	Imagens, áudio e vídeo	Sim	Inglês

6. Referências

Abelson, H., & Sussman, G. J. (1996). *Structure and Interpretation of Computer Programs*, 2nd ed. Cambridge: MIT Press.

Bense, M. (1968). *Pequena Estética*. São Paulo: Perspectiva.

Brennan, K. (2011). *Creative computing: A design-based introduction to computational thinking*. Disponível em http://tinyurl.com/brennancc. Acesso em 16 de agosto de 2012.

Brennan, K., & Resnick, M. (2012). New Frameworks for Studying and Assessing the Development of Computational Thinking. *Proceedings of the 2012 annual meeting of the American Educational Research Association*, Vancouver, Canada.

Christensen, M. H. (2009). *Structural Synthesis using a Context Free Design Grammar Approach*. Generative Art Conf. Milan.

Collins, N. (2007). Live Coding Practice. *Proceedings of NIME*.

Dale, N., & Lewis, J. (2010). *Ciência da Computação*. Rio de Janeiro: LTC.

De Smedt, T., Lechat, L., Daelemans, W. (2011). Generative Art Inspired by Nature, Using NodeBox. In: *Lecture Notes in Computer Science*, 2011, Volume 6625, pp.264-272. Berlin / Heidelberg: Springer.

Denning, Peter J. (2003). Great principles of computing. *Communications of the ACM*, v.46 n.11, November 2003, pp.15-20.

Galanter, P. (2003). What is Generative Art? Complexity theory as a context for art theory. in *International Conference on Generative Art. 2003*. Milan: Generative Design Lab.

Gray, J., Abelson, H., Wolber, D., e Friend, M. (2012). Teaching CS Principles with App Inventor. *ACMSE'12, March 29–31*, 2012, Tuscaloosa, AL, USA.

Greenfield, A. (2006). *Everyware: the dawning age of ubiquitous computing*. Berkeley: New Riders.

Lentczner, M. (2008). *Community of Variation*. Moutain View: Ozone House.

Lieser, W. (2010). *Arte Digital: Novos caminhos na arte*. Potsdam: H.F. Ullman.

Papert, S. & Solomon, C. J. (1971). Twenty things to do with a computer. *MIT AI Lab. LOGO Memo 3*, July 1971.

Papert, S. (1996). An Exploration in the Space of Mathematics Educations. *International Journal of Computers for Mathematical Learning*, 1(1), pp.95-123.

Puckette, M. (1996). Pure Data: another integrated computer music environment. In: *Proceedings, Second Intercollege Computer Music Concerts*, Tachikawa, Japan, pp. 37-41.

_____ (2007). *The Theory and Technique of Electronic Music*. Disponível em http://tinyurl.com/pdmpbook. Acesso em 10 de agosto de 2012.

Resnick, M., Kafai, Y., Maeda, J. (2003). *A Networked, Media-Rich Programming Environment to Enhance Technological Fluency at After-School Centers in Economically-Disadvantaged Communities*. Proposal to National Science Foundation.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for All. *Communications of the ACM*, vol. 52, no. 11, pp. 60-67.

Sociedade Brasileira de Computação – SBC (2011). *Computação Brasil*, abril / maio / junho de 2011. Porto Alegre: Giornale Comunicação Empresarial.

Solomon, C. J. (1976). Leading a Child to a Computer Culture. *Proceedings of the ACM SIGCSE-SIGCUE technical symposium on Computer science and education*, pp.79-83, 1976.

Sorensen, A. (2005). Impromptu: An interactive programming environment for composition and performance. *In: Proceedings of the Australasian Computer Music Conference 2009*, 2-4 July 2009, Queensland University of Technology, Brisbane.

Stephenson, C. & Barr, V. (2011). Defining Computational Thinking for K-12. *CSTA Voice* 7 (2), pp.3-4.

Venturelli, S. e Burgos, F. (1999). *Arte Computacional*. Texto apresentado no evento DigitalArte, ICBA, Salvador, Bahia – 19 de março de 1999. Disponível em http://tinyurl.com/venturelli1. Acesso em 17 de agosto de 2012.

Wing, J. M. (2006) Computational Thinking, *Communications of the ACM*, 49(3), March 2006.

_____(2008). Computational Thinking and Thinking About Computing, *Philosophical Transactions of the Royal Society*, 366, 3717-3725.

Wolber, D., Abelson, H., Spertus, E., & Looney, L. (2011). *App Inventor: Create your own Android Apps.* Sebastopol: O'Reilly Media.