1
Introdução

Este capítulo introduz o contexto de pesquisa desta tese e uma visão detalhada da metodologia utilizada no desenvolvimento do trabalho. Ao final, descrevemos a organização do restante do texto.

1.1.
Contexto da pesquisa

Cada vez mais se ensina programação de computadores para crianças e adolescentes. Um dos primeiros trabalhos nesta linha surgiu na década de 80 com a linguagem Logo (Papert, 1980). Apesar de relativamente antiga, a iniciativa de ensinar programação vem ganhando força nos últimos anos por causa do papel importante do raciocínio computacional no ensino e aprendizado de conteúdos ligados a matemática, ciência e lógica, que são requisito fundamental para as carreiras como um todo e para profissionais na indústria de tecnologia (Mayer, 2013; Nickerson e Zodhiates, 2013). O termo 'raciocínio computacional' é comumente definido como um conjunto de habilidades que envolvem a resolução de problemas, a generalização e a transferência de conhecimentos adquiridos para outros contextos (Wing, 2006; ISTE, 2014; Google, 2014).

Em relatório de um *workshop* sobre raciocínio computacional promovido em 2010 pelo National Research Council (NRC, 2010), Resnick cria uma analogia dizendo que a programação é como a habilidade de escrita e o raciocínio computacional é como o letramento. Assim como é importante saber escrever, é também importante saber programar, mas não é o suficiente. O autor ressalta que a programação deve ser um meio de expressão e um ponto de partida para o desenvolvimento de novas formas de pensar. Complementarmente, Wolz argumenta que a programação é uma linguagem para expressar ideias e que, por isso, é preciso aprender a ler e a escrever em uma linguagem para conseguir pensar de acordo com ela. Em outro documento do mesmo conselho de pesquisa (NRC, 2011), Wilensky afirma que o raciocínio computacional pode aumentar a

autoexpressão e a colaboração entre as pessoas, apoiando diferentes formas de expressão e de compartilhamento fácil dessas expressões.

Nessa linha, Burke e Kafai (2010) investigaram como escrever programas pode ajudar a desenvolver a capacidade de contar histórias e de escrever de modo criativo. Neste estudo, os autores perceberam que contar histórias auxilia no aprendizado de programação e que a programação pode ser usada para refletir sobre a história que está sendo construída. Segundo eles, criar histórias usando programas é melhor porque a revisão do processo é imediata, diferentemente do que ocorre quando outros mecanismos são utilizados (por exemplo, câmera de vídeo, papel e caneta). Anteriormente, Kelleher *et al.* (2007) já haviam observado maior motivação dos alunos quando usaram o processo de contar histórias (*storytelling*) no aprendizado de programação em escolas de ensino médio. Isto foi especialmente importante para atrair a atenção de meninas para a área de computação.

No que diz respeito às ferramentas usadas no ensino de programação, podemos destacar três tipos: ambientes de programação, extensões desses ambientes na *Web* e mecanismos específicos de ajuda. Existem diversas plataformas direcionadas para o ensino de programação. Algumas delas propõem a criação de jogos e simulações através da construção de blocos de comandos prontos (Repenning e Ioannidou, 2004; Kelleher *et al.*, 2007; Maloney *et al.*, 2010; Fristoe *et al.*, 2011), enquanto outras introduzem o usuário a linguagens de programação e escrita de código fonte dos programas (Hambrusch *et al.*, 2009; Kölling, 2010). Com relação às extensões *Web* dos ambientes de programação, elas geralmente funcionam como um meio de publicação dos programas construídos (Bennett *et al.*, 2011) e como um espaço de compartilhamento, incentivo a *remixagem* e reuso, de comunicação entre os usuários ou de busca de material didático (Scratch, 2014; Greenroom, 2014). Por último, quanto aos mecanismos específicos de ajuda, destacam-se como exemplos os *plug-ins* de ambientes de programação que tem como objetivo explicar o funcionamento de funções, objetos, classes, como uma forma de documentação do ambiente de desenvolvimento e da linguagem (Myers *et al.*, 2004).

Entre as finalidades de uso dessas ferramentas de programação está a participação social através de *softwares* na Internet, que é um dos marcos mais importantes das sociedades contemporâneas (Shah *et al.*, 2001; Livingstone *et al.*,

2005). Um facilitador desta participação são os sistemas gerenciadores de conteúdos como Joomla[1] (Rahmel, 2009), Wordpress[2] (Brazell, 2010) e Drupal[3] (Byron, 2008), onde os usuários finais podem criar *sites* e interações para outros usuários. Vale ressaltar que os criadores de conteúdo *online* são uma minoria da população de usuários, e que suas contribuições são disponibilizadas segundo o que é mais interessante, de acordo com o que eles pensam (Brake, 2014). Uma das razões disso é o fato de que muitas destas ferramentas têm um alto grau de complexidade e, para disponibilizar *websites* na Internet, exigem de quem as utiliza um conhecimento avançado para melhor exploração dos recursos. Se as pessoas forem estimuladas a aprender e praticar a programação como forma de autoexpressão, poderemos contribuir para uma sociedade um pouco mais igualitária e consciente. Isto porque, ao perceberem os princípios básicos por trás de programas de computador e saberem como estes podem ser usados para comunicar ideias e exercer influência, possivelmente, mais usuários poderão decidir quais são as melhores maneiras de atuar na dimensão digital da vida em sociedade, seja no papel de programadores ou de usuários de tecnologias,

Note que, embora a programação possa ser estudada como um meio de expressão, ainda não existem recursos tecnológicos para destacar as características de comunicação social de programas. Por isso, é importante criar mecanismos que viabilizem o desenvolvimento desses recursos, de modo que seja possível oferecer uma plataforma tecnológica consistente para as práticas de ensino e aprendizado de programação como forma de autoexpressão e comunicação social em contextos educacionais. Caso este problema não seja endereçado, as tecnologias existentes podem continuar limitando de formas desconhecidas, primeiramente, a autoexpressão através de *software* e, futuramente, a atuação consciente das pessoas na sociedade.

Diante desse contexto, esta tese pretende contribuir para o *design* de tecnologias que auxiliem o processo de ensino e aprendizado de raciocínio computacional, considerando a programação como forma de autoexpressão. Nossa abordagem está alinhada com as anteriormente citadas, nas quais a programação é vista como um meio de comunicar uma ideia, está perto do uso de ensino de

[1] http://www.joomla.org/
[2] http://www.wordpress.com/
[3] http://www.drupal.org/

programação para contar histórias, e está além da habilidade de resolver problemas usando algoritmos. Com relação ao ferramental tecnológico, estamos interessados no *design* conceitual das ferramentas para esta perspectiva. Por isso, propomos uma forma para estruturar o *design* de extensões para ambientes de programação que também devem fornecer mecanismos específicos de ajuda. A originalidade da nossa abordagem é a exploração das dimensões da programação que buscam ressaltar as características de significação em programas, ou seja, que enfatizam diferentes formas de expressar conteúdos e intenções através de programas.

A pesquisa relatada nesta tese foi desenvolvida e aplicada no contexto do projeto *Scalable Game Design* Brasil (de Souza *et al.*, 2014). O SGD-Br é a versão brasileira do projeto *Scalable Game Design* (SGD), liderado pelo professor e pesquisador Alexander Repenning, da Universidade do Colorado em Boulder (EUA). Este projeto tem como objetivo inovar o ensino de Computação, Ciências e Matemática em escolas públicas dos Estados Unidos.

O projeto SGD-Br teve seu início em 2010, tendo uma escola pública como parceira. Em 2012, duas escolas particulares (uma nacional e uma internacional) integraram o grupo de escolas participantes. Ao final do projeto em 2015, espera-se ter alcançado uma sólida compreensão dos desafios, oportunidades e requisitos tecnológicos específicos para um programa mais amplo de educação para a Computação em escolas brasileiras. A maior motivação do SGD-Br é ajudar a promover a aquisição do raciocínio computacional para que as pessoas possam entender melhor o significado social da computação, dominar conceitos computacionais básicos e aprender a se expressar através de programas de computador.

Essa forma de ver programação como autoexpressão é a base da Semiótica Computacional (Andersen, 1990; Andersen, 1993) e de todas as abordagens que dela derivam. Esta tese foi desenvolvida no Semiotic Research Engineering Group (SERG), que é um centro internacional de referência para Semiótica Computacional e, também, o grupo onde teve origem a teoria da Engenharia Semiótica (de Souza, 2005). Trata-se de uma teoria semiótica proposta para a área de IHC (Interação Humano-Computador) cujo foco é o processo de metacomunicação (comunicação sobre comunicação). Tal como em todas as abordagens semióticas para IHC (Nadin, 1988; Andersen, 1993), na Engenharia

Semiótica a interação do usuário com as interfaces de tecnologias de base computacional é vista como uma comunicação mediada por computador que acontece entre o *designer* da tecnologia em questão e seu o usuário em tempo de interação. Esta teoria oferece o embasamento para a investigação realizada nesta tese e apoia a perspectiva adotada no projeto SGD-Br, em que o *software* é visto como uma forma de expressão e comunicação.

Assim como no SGD, estamos utilizando um *software* de programação visual chamado AgentSheets (AS), que permite o desenvolvimento de jogos e simulações através da manipulação direta de agentes. O AS foi concebido com a intenção de permitir a construção de simulações para explorar ideias complexas e comunicá-las para outras pessoas (Repenning e Ioannidou, 2004).

Após um ano de implementação do projeto SGD-Br, de Souza *et al.* (2011) realizaram uma pesquisa para verificar como significados expressos nas narrativas em linguagem natural de um grupo de alunos de uma escola parceira estavam relacionados aos significados que apareciam nos programas que eles construíram com o AS. Os resultados sugeriram uma grande riqueza nos significados das representações visuais do AS que, no entanto, precisava ser melhor explorada para que os estudantes desenvolvessem certos aspectos importantes ligados ao raciocínio computacional, principalmente com relação aos aspectos expressivos. Ainda nessa linha, Ferreira *et al.* (2012) tentaram descobrir o que poderia ser feito para explorar esses significados escondidos. Os autores investigaram, de maneira exploratória, o impacto da adição de novas representações na compreensão dos programas e de tarefas de modificação dos mesmos. Esta adição permitiu aos participantes corrigir e expandir o aprendizado prévio. Mais do que isso, essas representações mostraram potencial para apoiar novas estratégias de ensino de programação como forma de expressão.

Nesse contexto, a questão central de pesquisa desta tese é *como apoiar o design de múltiplas representações de programas para apoiar o processo de ensino e aprendizado de programação como forma de autoexpressão*.

Para iniciar o endereçamento desta questão, nós desenvolvemos o ambiente PoliFacets-AS que consiste em múltiplas representações dos programas construídos com o AS. O PoliFacets-AS é uma extensão *Web* do AS que tem como função principal permitir a exploração dos significados e explicitação de

conceitos importantes que fazem parte da programação como forma de autoexpressão.

Para facilitar o entendimento, vamos exemplificar como a programação realizada no AS pode ser explorada através do PoliFacets-AS. Considere o exemplo[4] presente na Figura 1.1. Esta é uma simulação simplificada do processo de produção do mel, onde duas abelhas que se movimentam por meio de comandos no teclado. Cada um das abelhas deve passar por cima das flores, coletar néctar e levá-lo até a colmeia. Este processo deve se repetir até que as colmeias fiquem cheias de mel. Entretanto, os sapos estão com fome e comem as abelhas sempre que elas se aproximam da boca deles.

A Figura 1.1 exibe a galeria de agentes, parte do comportamento do agente *abelha* e a planilha de trabalho onde a simulação é construída no AS. Os balões com o nome dos agentes na planilha de trabalho não aparecem no AS e foram introduzidos apenas na imagem para fins de explicação. Cada agente pode ter uma ou mais aparências. Na galeria, observe que a abelha tem três aparências: *Abelha*, *Abelha_Nectar*, *Abelha_Morta*.

[4]http://www.serg.inf.puc-rio.br/polifacets/facets/description.lua?id_project=915

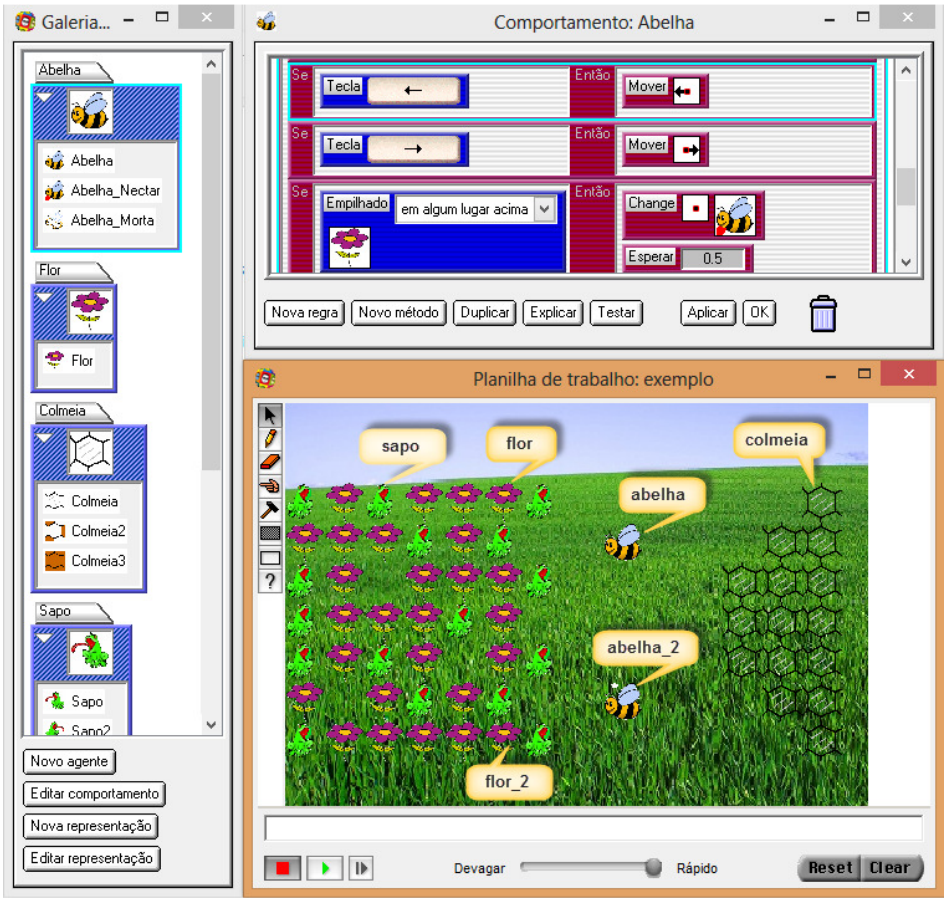

Figura 1.1 - Construção da simulação 'produção de mel' no AS

A programação do comportamento dos agentes adota o padrão de regras de produção, SE <condições> ENTÃO <ações>. As condições aparecem à esquerda e as ações à direita. Se todas as condições de uma regra são satisfeitas, as ações que estão na mesma regra (mesma linha) são executadas na ordem em que estão posicionadas. O conjunto de condições é visto como uma conjunção lógica, enquanto que o conjunto de ações é uma sequência de comandos.

Na planilha de trabalho destacada na Figura 1.1, note que os agentes *abelha* e *abelha_2* têm a mesma imagem, assim como os agentes *flor* e *flor_2*. Entretanto, se executarmos a simulação, observamos um comportamento diferenciado. Considere que os agentes *abelha* e *abelha_2* se movem e que pode ocorrer um efeito visual quando eles interagem com os outros agentes. A Tabela 1.1 detalha as interações entre os agentes, destacando quais delas têm efeitos iguais e quais agentes contêm as regras que produzem os efeitos visuais.

Tabela 1.1 - Esquema de interação entre os agentes e seus efeitos

Encontros entre agentes			
Agentes imóveis	**Agentes móveis**	**Efeito**	**Agente responsável**
Sapo	Abelha	X	sapo
	abelha_2	X	abelha_2
Flor	Abelha	Y	abelha
	abelha_2	-	-
flor_2	Abelha	-	-
	abelha_2	Y	flor_2
Colmeia	Abelha	Z	colmeia
	abelha_2	Z	colmeia

Vamos supor que dois programadores estivessem criando esta simulação. Um deles criou a *abelha* e a *flor* e o outro criou a *abelha_2* e a *flor_2*. Quem está apenas visualizando a simulação não consegue perceber que a programação é diferente em cada caso. Veja no Quadro 1.1 os efeitos visuais em diferentes instantes no tempo e a descrição deles a seguir:

- Efeito X - Quando o agente *abelha* encontra o agente *sapo* pelo lado direito do *sapo*, o *sapo* come a *abelha* e a regra deste comportamento está programada no *sapo*. Entretanto, o mesmo efeito visual pode ser programado em outro agente, como é o caso do encontro entre o agente *abelha_2* e o agente *sapo*, onde a *abelha* é responsável pelo efeito mudando sua própria imagem e depois desaparecendo ao chegar perto do *sapo*.
- Efeito Y - Os agentes *flor* e *flor_2* tem a mesma imagem, mas são agentes diferentes e têm comportamentos diferentes. O agente *flor* só produz efeito visual ao interagir com o agente *abelha*, enquanto que o agente *flor_2* produz o mesmo efeito visual apenas quando interage com o agente *abelha_2*. Ou seja, cada abelha consegue obter néctar de determinadas flores, apesar de abelhas e flores parecerem iguais, elas têm comportamentos diferentes. Nesse caso, novamente, um dos programadores posicionou a ação na abelha e o outro programou a ação na flor.
- Efeito Z - Suponha que os dois programadores construíram juntos as regras do agente *colmeia*. A colmeia recebe o néctar depositado pelos agentes *abelha* e a *abelha_2* de forma similar e o comportamento está programado apenas na *colmeia*. Note que não é a *abelha* que deposita o néctar na *colmeia*, mas a *colmeia* que absorve o néctar da *abelha*.

Quadro 1.1 - Efeitos visuais no tempo

Efeitos	Cenas no tempo
X	
Y	
Z	

Ao observar apenas a simulação no AS é difícil descobrir que agentes com imagens iguais são agentes diferentes sem ter que desfazer parte da construção na planilha de trabalho. Além disso, os comportamentos são programados em cada agente e não há uma visão geral da programação. Por essas e outras questões, nós desenvolvemos o PoliFacets-AS com a ideia de desconstruir os programas do AS e conseguir compreender melhor o que estava nos bastidores dos jogos e simulações. Denominamos as novas representações elaboradas no PoliFacets-AS de facetas.

Em uma das facetas, chamada ‘planilha’, por exemplo, é possível habilitar e desabilitar a visualização dos agentes, além de ver quais agentes e quantas instâncias estão na simulação. Observe que na Figura 1.2 desmarcamos os agentes *sapo* (16 instâncias) e *flor* (15 instâncias). Para descobrir o nome do agente presente na planilha, basta clicar nele, como foi feito com a *abelha_2*.

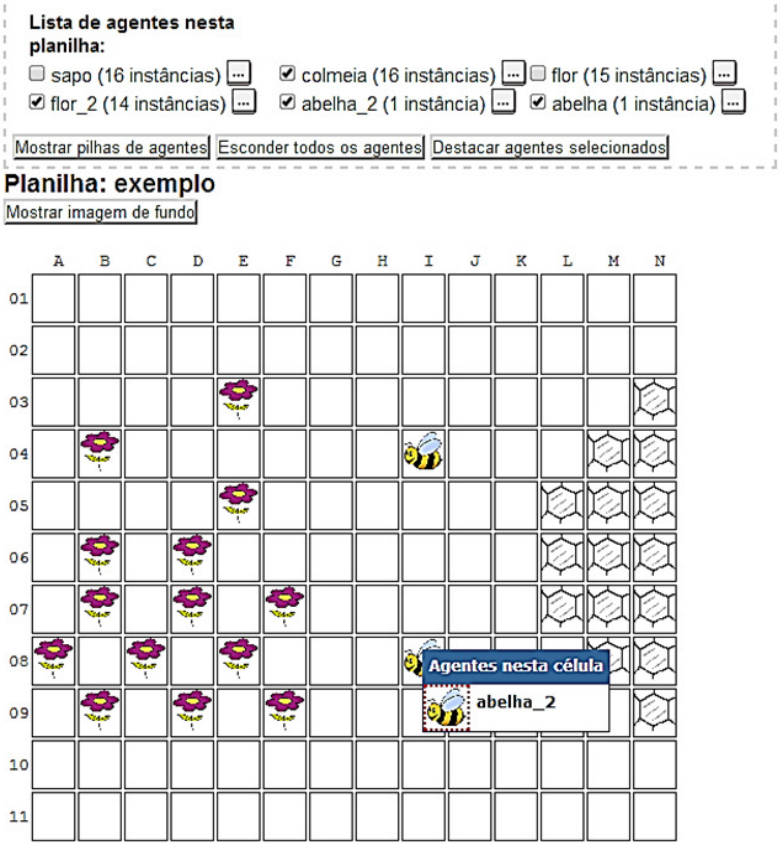

Figura 1.2 - Faceta 'planilha' da simulação 'produção de mel'

Outra representação do PoliFacets-AS é a faceta ‘conexões’ que mostra como os agentes estão relacionados e que regras formam as relações. Observe na Figura 1.3 as conexões do agente *flor_2*, que, como já explicamos, só interage com o agente *abelha_2*. É possível visualizar as diferentes imagens de cada agente e ler as regras que formam as conexões em uma linguagem pseudonatural.

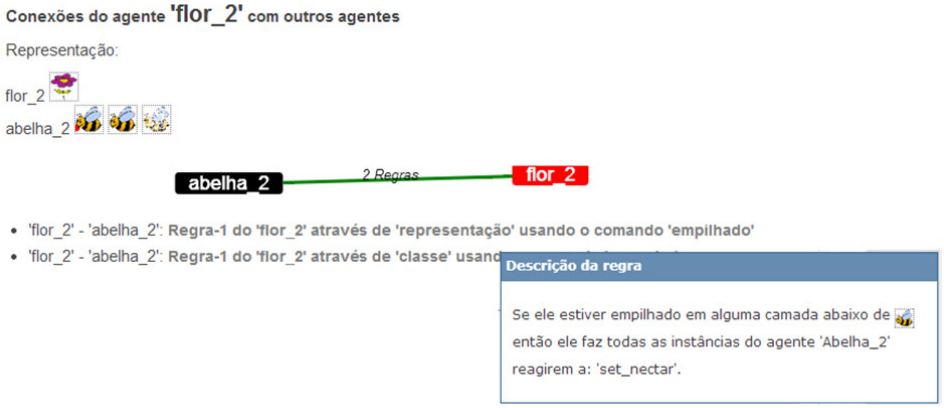

Figura 1.3 - Conexões do agente *flor_2*

Ainda na faceta ‘conexões’, veja na Figura 1.4 as ligações do agente *abelha_2*. A linha cinza ligando os agentes *colmeia* e *flor_2* ao agente *abelha_2* significa que os agentes estão conectados, mas que as regras não estão no agente *abelha_2* e sim no outro agente. Nesse caso, a *abelha_2* está conectada ao *sapo* através de quatro regras. A ausência de ligação entre *abelha_2* e *flor* mostra que não há nenhum efeito quando esses agentes interagem entre si.

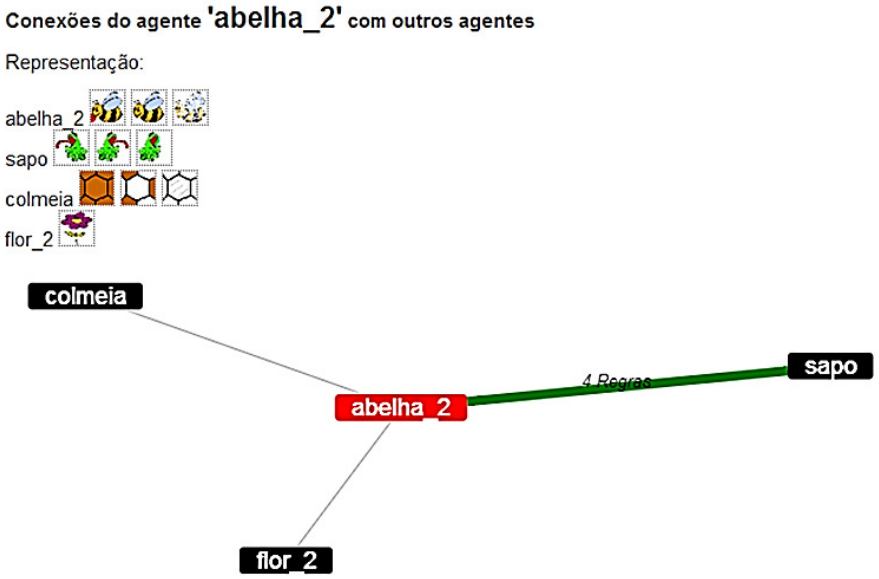

Figura 1.4 - Conexões do agente *abelha_2*

Mais informações sobre as facetas do PoliFacets-AS serão apresentadas nos próximos capítulos. Esse exemplo inicial já possibilita chamar a atenção para nossa perspectiva sobre a significação, ao tornar perceptível que a significação da interface pode ser diferente da significação do programa. O efeito que é comunicado através da interface pode ser programado de diversas formas. Essa diferença é resultado da autoexpressão, mas pode ser motivada pelo reuso, otimização de *software* e, até mesmo, por algum tipo de limitação do interpretador da linguagem entre outros fatores.

O PoliFacets-AS foi concebido como uma ferramenta que permite destacar a questão da significação para o usuário considerando aspectos relacionados à intenção (ação), ao conteúdo (informação) e à expressão (forma) nos jogos e simulações construídos no AS. É importante fazer os usuários do sistema (professores e alunos) perceberem que um mesmo conteúdo pode ser expresso de diferentes formas, uma mesma forma pode conter diferentes conteúdos e a

intenção também pode variar diante de modificações nas formas e conteúdos dos programas ou, até mesmo, na ausência de alterações. O programador deve estar consciente dessas possibilidades para compreender que tem escolhas sobre as formas que pode usar para se comunicar através dos programas.

Considerando a tendência atual de as ferramentas de programação disponibilizarem conteúdos adicionais em *sites* na *Web*, o AS oferece como recursos uma página Wiki[5] e um ambiente chamado SGD Arcade[6]. Na Wiki é possível encontrar descrições e tutoriais sobre os jogos e simulações que servem como exemplos. Já o SGD Arcade (Basawapatna *et al.*, 2010) permite o envio dos projetos para a *Web* a partir de usuários autorizados, mas qualquer usuário pode acessar os projetos, executá-los, fazer o *download* dos arquivos e visualizar um grafo de padrões de raciocínio computacional que ajuda a avaliar a evolução dos aprendizes. O PoliFacets-AS adotou a mesma linha, ser um ambiente *Web* complementar, mas com um diferencial importante de atuar como uma documentação ativa, não sendo apenas um ambiente para leitura de páginas e navegação através de hiperlinks.

Segundo Phelps (1997), a essência da documentação ativa é a noção de uma conversa entre o programa e seu usuário. O PoliFacets-AS habilita uma rede de 'conversas' que permite a reflexão sobre os jogos e simulações construídos pelos usuários do AS. Desse modo, o aprendiz deve conseguir 'conversar' com as facetas dos seus programas e refletir para compreender com mais clareza os conceitos e significados existentes.

Durante o desenvolvimento do PoliFacets-AS, realizamos diversos estudos empíricos que resultaram em informações sobre as facetas e sobre como elas podiam ser exploradas. Essas informações, somadas a fundamentação teórica na Engenharia Semiótica, deram origem ao modelo conceitual de *design* chamado PoliFacets. O principal resultado desta tese é o modelo para estruturar o conteúdo de documentação ativa destinada a auxiliar no processo de ensino e aprendizado de raciocínio computacional, quando esta atividade visa ressaltar a autoexpressão através de programas.

Não está no escopo desta tese apresentar um ferramental prático comprovado sobre o efeito do modelo conceitual proposto no aprendizado de

[5] http://sgd.cs.colorado.edu/wiki/Scalable_Game_Design_wiki
[6] http://scalablegamedesign.cs.colorado.edu/arcade/

raciocínio computacional. Portanto, não é nosso objetivo apresentar e validar uma tecnologia de documentação ativa que ensina as pessoas a usar programação como forma de autoexpressão e participação social. Para obter uma evidência empírica sobre a implicação do ensino de programação como forma de autoexpressão, precisaríamos de estudos posteriores com um escopo muito mais abrangente do que esta tese. Ainda assim, estes estudos dependem da disponibilidade de ferramentas tecnológicas que, justamente, esta tese pretende facilitar através do modelo conceitual proposto.

Desse modo, o modelo PoliFacets é importante como uma ferramenta epistêmica, pois ajuda a explorar a natureza e o espaço de interpretação do problema, que é o primeiro passo para potencializar o sucesso de futuras ferramentas comprometidas com a questão da programação como forma de expressão. Os resultados da aplicação do modelo indicam como ele pode contribuir para o *design* de documentos ativos no contexto do processo de ensino e aprendizado de programação.

O diferencial do modelo PoliFacets é oferecer ferramental epistêmico para construção de ferramentas que permitam a exploração de significação em programas. Ele é uma ferramenta para ajudar o *designer* a pensar nas possibilidades da criação de múltiplas representações de programas; não é um guia prático com regras específicas.

1.2. Metodologia

Para responder a questão de pesquisa desta tese foi utilizada a metodologia qualitativa (Denzin e Lincoln, 2006). Realizamos estudos em profundidade com pequenos grupos de participantes, coletando evidências empíricas a partir de observações, questionários e entrevistas. Dados de áudio e vídeo foram capturados de acordo com as necessidades de cada estudo, mediante a concordância explícita dos participantes, por meio de termos de consentimento que fazem parte do protocolo de ética da pesquisa. Cada estudo empírico realizado teve um procedimento próprio e será relatado em seção específica no terceiro capítulo.

Como dito anteriormente, esta tese é parte de um grande projeto de pesquisa. Diversos estudos empíricos foram realizados durante aproximadamente três anos. Entretanto, vamos tratar apenas do subconjunto que nos levou à

construção do modelo de *design* da tecnologia para auxiliar o processo de ensino e aprendizado de programação.

As grandes etapas envolvidas nesta construção estão na Figura 1.5 . Os elementos na cor azul reportam a parte da pesquisa realizada antes desta tese. Os elementos na cor laranja relatam a parte da pesquisa que foram importantes para a tese e para o projeto SGD-Br de modo geral. Por último, os elementos na cor verde representam a parte da pesquisa especificamente elaborada e conduzida para esta tese.

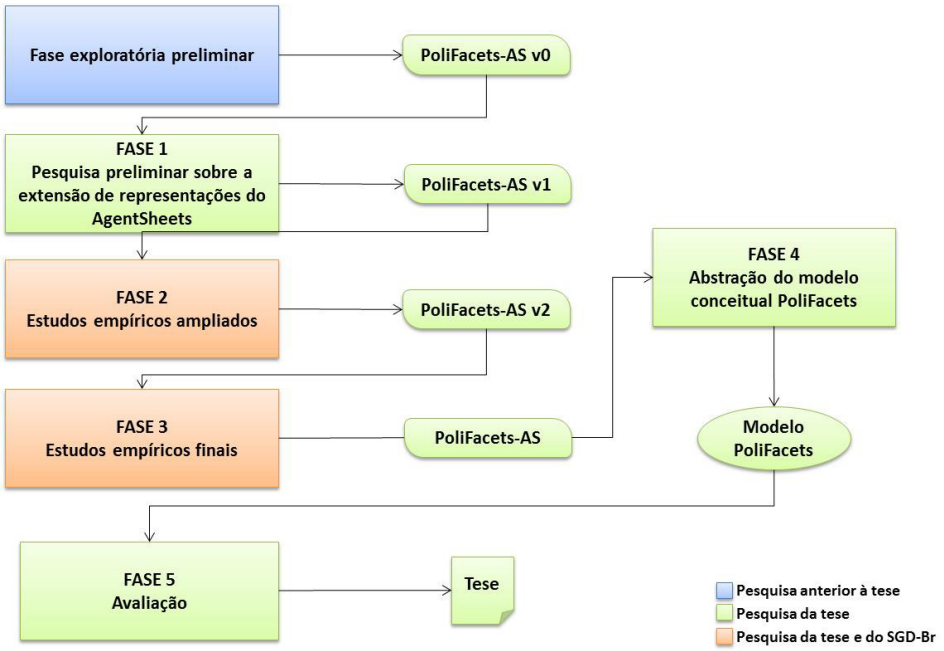

Figura 1.5 - Etapas para a construção do Modelo PoliFacets

Primeiramente, houve uma fase exploratória preliminar realizada por pesquisadores do SGD-Br (de Souza *et al.*, 2011; Ferreira *et al.*, 2012) que motivou a pesquisa desta tese, dando origem a um protótipo do PoliFacets-AS (PoliFacets-AS v0). A FASE 1 da pesquisa foi um estudo sobre a extensão das representações do AS. Neste estudo inicial, confirmamos a necessidade de auxiliar os professores no processo de ensino e aprendizado de programação e começamos a perceber o valor das representações neste processo. Os resultados deste estudo serão descritos no capítulo três (subseção 3.2.1). Depois desta fase, o PoliFacets-AS foi melhorado e relançado em nova versão (PoliFacets-AS v1).

Os objetivos dos estudos nas FASES 2 e 3 estavam relacionados à pesquisa do projeto SGD-Br e desta tese. A FASE 2 relata estudos empíricos ampliados. A

primeira etapa desta fase foi a realização do primeiro *workshop* para professores (subseção 3.2.2). Depois do *workshop*, duas novas escolas iniciaram a parceria de pesquisa. Assim, introduzimos a utilização do PoliFacets-AS nos cursos com o AS. Então, foi possível realizar um estudo específico sobre uma faceta que mostra as regras e narrativas de programas (subseção 3.2.3) e outro estudo sobre uma forma de expressar as conexões entre os agentes dos programas (subseção 3.2.4). Ao final, novamente, o PoliFacets-AS foi avaliado e melhorado resultando na disponibilização de uma nova versão (PoliFacets-AS v2).

Na FASE 3, foram realizados dois estudos: o segundo *workshop* para professores (subseção 3.2.5); e um estudo sobre uma nova estratégia de comunicação no PoliFacets-AS (subseção 3.2.6). Estes estudos empíricos finais resultaram na versão atual do PoliFacets-AS.

As FASES 1, 2 e 3 relatam os estudos empíricos (seção 3.2) que serviram de insumo para a criação do modelo PoliFacets. Cada estudo foi responsável por agregar determinadas características ao modelo. Com os estudos, descobrimos requisitos para possibilidades de facetas que, quando abstraídas, possibilitam uma nova oportunidade de *design* a ser explorada. Além disso, os resultados mostram a relevância das facetas no processo de ensino e aprendizado de programação.

As versões sucessivas do PoliFacets-AS e os estudos foram importantes porque permitiram, na FASE 4, a abstração de um modelo PoliFacets (descrito no capítulo 3). Ao contrário das fases restantes, a FASE 4 foi essencialmente teórica e analítica, tendo em vista que e a construção do modelo foi baseada nos resultados dos estudos empíricos realizados e na teoria da Engenharia Semiótica.

Na FASE 5, o modelo PoliFacets passou por um processo de avaliação. Primeiramente, foi realizado um estudo sobre a recepção das facetas por parte de alunos participantes do projeto SGD-Br (subseção 4.3.1). Depois disso, o modelo foi validado na qualidade de prova de conceito (em contraposição ao procedimento de validação empírica) através da sua aplicação em outra ferramenta de programação (subseção 4.3.2). Realizamos a validação aplicando o modelo PoliFacets no NCL Composer, que é uma ferramenta destinada ao desenvolvimento de programas para TV Digital. A parte final da pesquisa é a síntese do conhecimento adquirido que resulta no texto desta tese.

1.3. Organização do texto

Esta tese está dividida em cinco capítulos. O segundo capítulo reporta os trabalhos mais importantes relacionados a esta pesquisa.

O terceiro capítulo apresenta o modelo PoliFacets incluindo a fundamentação teórica, os estudos empíricos que serviram insumo para sua construção e sua elaboração, componentes estruturais e visão completa.

O quarto capítulo detalha a instância PoliFacets-AS no contexto do projeto SGD-Br explicando como cada faceta funciona e como as ligações entre elas acontecem. Além disso, o quarto capítulo reporta a avaliação do modelo com estudos empíricos envolvendo estudantes e uma aplicação do modelo em outra ferramenta de programação.

No quinto capítulo, estão as considerações finais; ele trata das lições aprendidas e das limitações da pesquisa, sintetiza as contribuições e trabalhos futuros, apresentando novas oportunidades para descobertas.

Após as referências, há apêndices[7] e anexos com materiais adicionais utilizados e obtidos na coleta de dados para os estudos empíricos. Os textos referentes aos termos de consentimento utilizados estão ao final de cada seção de apêndices e anexos.

[7] Segundo a NBR 14724 da ABNT, apêndice é um documento ou texto elaborado pelo autor do trabalho. O anexo é um documento elaborado por terceiros. Os anexos e apêndices são utilizados para complementar informação de um trabalho principal.