PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

João Antonio Dutra Marcondes Bastos

Apoio à transferência de conhecimento de raciocínio computacional de linguagens de programação visuais para linguagens de programação textuais

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio.

Orientador: Prof. Clarisse Sieckenius de Souza

Rio de Janeiro
Abril de 2015

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

João Antonio Dutra Marcondes Bastos

Apoio à transferência de conhecimento de raciocínio computacional de linguagens de programação visuais para linguagens de programação textuais

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Clarisse Sieckenius de Souza
Orientador
Departamento de Informática – PUC-Rio

Prof. Alberto Barbosa Raposo
Departamento de Informática – PUC-Rio

Prof. Bruno Feijó
Departamento de Informática – PUC-Rio

Prof. José Eugênio Leal
Coordenador (a) Setorial do Centro
Técnico Científico - PUC-Rio

Rio de Janeiro, 13 de abril de 2015

João Antonio Dutra Marcondes Bastos

Graduou-se em Ciência da Computação pela Universidade Federal de Viçosa (2009). Sua experiência profissional inclui três anos atuando como desenvolvedor e analista de sistemas.

Ficha Catalográfica

Bastos, João Antonio Dutra Marcondes

Apoio à transferência de conhecimento de raciocínio computacional de linguagens de programação visuais para linguagens de programação textuais / João Antonio Dutra Marcondes Bastos ; orientador: Clarisse Sieckenius de Souza. – 2015.
76 f. ; 30 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2015.
Inclui bibliografia

1. Informática – Teses. 2. Linguagens de programação. 3. Raciocínio computacional. 4. AgentSheets. 5. GreenFoot. 6. Ensino de programação para jovens e crianças. I. Souza, Clarisse Sieckenius de. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Aos que sempre me apoiaram...

Agradecimentos

Agradeço, em primeiro lugar, a minha mãe, Míriam, pelo apoio incondicional ao longo de toda minha trajetória. Obrigado por todas as vezes que colocou minhas necessidades na frente até mesmo das suas. Sem você eu não teria chegado até aqui.

Agradeço a minha querida professora e orientadora Clarisse Sieckenius de Souza pela confiança em mim depositada. Muito obrigado por compartilhar comigo parte de seu amplo conhecimento, e por sua orientação sempre objetiva e de excelência. Apesar das dificuldades em me adaptar, estou certo que tive o maior crescimento pessoal e profissional possível nesses dois anos de mestrado.

Agradeço aos amigos do SERG e do projeto SGD-Br por todas as conversas e conselhos ao longo desses dois anos. Cleyton, Marcelle, Ingrid, Eduardo, Hugo, Rosana, Martha, Carla, Luciana, Juliana, Luiz, Rafael, Priscila e Bruno, meu muito obrigado.

Agradeço a minha amada namorada, Érica, que sempre me apoiou e entendeu quando eu precisava me dedicar aos estudos. Estar com você esse ano fez com que a caminhada ficasse mais simples e feliz. Aproveito para agradecer também ao nosso filhote de quatro patas Sheldon, meu maior companheiro nas madrugadas de estudo.

Agradeço aos meus familiares, em especial ao meu irmão e minha avó que sempre me apoiaram.

Agradeço ao departamento de informática da PUC-Rio, a CAPES e a AMD pelo financiamento da minha pesquisa ao longo do mestrado. Por falar em financiamento, obrigado mais uma vez a minha mãe, que além de todo apoio ainda deu suporte financeiro.

Agradeço às escolas parceiras e aos professores e alunos que participaram do projeto SGD-Br que serviu como base para execução da minha pesquisa.

Por último, agradeço a todos que de alguma forma contribuíram para a execução dessa pesquisa de mestrado

Resumo

Bastos, João Antonio Dutra Marcondes; de Souza, Clarisse Sieckenius. **Apoio à transferência de conhecimento de raciocínio computacional de linguagens de programação visuais para linguagens de programação textuais.** Rio de Janeiro, 2015. 76p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Produzir tecnologia tem se mostrado uma habilidade cada vez mais indispensável na sociedade moderna. Os usuários estão deixando de ser simples consumidores e passando a ser produtores, usando a tecnologia para expressarem suas ideias. Nesse contexto, o aprendizado do chamado "raciocínio computacional" deve ser tão importante quanto o de disciplinas básicas, como a leitura, a escrita e a aritmética. Ao desenvolver tal habilidade o aluno vai conseguir se expressar através do software. Diversos projetos ao redor do mundo têm suas tecnologias e didáticas próprias a fim de auxiliar o aluno a desenvolver tal capacidade. Porém, sabemos que em um contexto que está em constante evolução como é o caso da informática, não podemos deixar que o aluno fique preso a uma única ferramenta ou meio de se expressar. Ferramentas podem ficar obsoletas e ele perderia seu poder de produtor de tecnologia. Pensando nisso, foi elaborado um modelo de transferência do aprendizado do raciocínio computacional a ser incorporado a sistemas de documentação ativa que apoiam o ensino-aprendizado desta habilidade. O modelo auxiliará o designer na criação de um artefato tecnológico que seja capaz de ajudar alunos e professores a aprenderem uma nova linguagem de programação. O modelo, que é baseado na Engenharia Semiótica, é a principal contribuição científica dessa dissertação de mestrado.

Palavras-chave

Linguagens de Programação; Raciocínio Computacional; AgentSheets; GreenFoot; Ensino de Programação para Jovens e Crianças

Abstract

Bastos, João Antonio Dutra Marcondes; de Souza, Clarisse (Advisor). **Support for computational thinking knowledge transfer from visual programming languages to textual programming languages.** Rio de Janeiro, 2015. 76p. MSc. Dissertation - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Producing technology has been an increasingly essential ability in modern society. The users are no longer simple consumers but actually, also, technology producers, using technology to express their ideas. In this context, the learning of the so-called "computational thinking" should be as important as learning basic disciplines such as reading, writing and arithmetic. As long as the student can develop this ability, he will be able to express himself or herself through the software. Many projects around the world have their own technologies and pedagogy to help the student develop such capacity. However, we know that in a context that is constantly evolving as is the case of informatics, we cannot allow the student to be attached to a single tool or means. Tools may become obsolete and students would lose their technology producer status. With this in mind, we designed a learning transfer model of "computational thinking", which will assist the designer in the creation of a technological artifact to help students and teachers learn a new programming language. The model, which is based on the Semiotic Engineering, is the main scientific contribution of this master's dissertation.

Keywords

Programming Languages; Computational Thinking; AgentSheets; Greenfoot; Programming Education for Youth and Children

Sumário

Lista de figuras

Lista de Tabelas

1
Introdução

Um dos principais desafios da atualidade para a sociedade é a introdução da informática como disciplina básica nos ciclos iniciais da educação. Aprender computação significa muito mais que a simples habilidade de navegar na internet, participar de redes sociais, ou usar editores de textos, planilhas, vídeos e apresentações para fazer seus trabalhos escolares. E esse aprendizado não deve acontecer visando apenas as possibilidades de mercado em TIC (área de Tecnologia de Informação e Comunicação). Assim como aprendemos biologia no ensino médio tendo o auxílio de um microscópio para aulas práticas, deveríamos também aprender computação, tendo o computador como um objeto de prática. Não é por aprendermos um pouco de biologia que nos tornamos biólogos aptos ao mercado de trabalho; mas tomamos conhecimento de aspectos do universo vivo que têm, por exemplo, grande impacto ambiental e consequências sociais evidentes. E é nesse meio, tão bem definido em outras áreas do conhecimento, mas ainda nebuloso para computação, que inserimos esse trabalho.

Existem várias frentes de pesquisa em busca de como e por onde começar o desenvolvimento do aprendizado desses conceitos. Uma corrente muito forte é a "*CS Unplugged*" (Tomohiro Nishida, 2009), onde professores ensinam computação através de atividades sem o uso do computador, explicitando assim, que a computação enquanto ciência vai muito além do uso da máquina, que não passa de uma ferramenta de trabalho.

Outra frente que tem grande representatividade é o desenvolvimento desses conceitos através do aprendizado da habilidade que recebeu o nome de "raciocínio computacional". O termo 'raciocínio computacional' (*computational thinking*) foi primeiramente definido por Wing em 2006. A autora defende que o raciocínio computacional é uma habilidade fundamental para todos, não apenas para graduados na área de computação, e envolve a formulação, a compreensão e a solução de problemas (WING, 2006). A estratégia mais aceita tem sido o desenvolvimento dessa habilidade através do uso de ferramentas que permitem aos alunos programar seus próprios jogos e simulações. Dentre vários ambientes

podemos citar: *Scratch* (MALONEY, RESNICK, RUSK, SILVERMAN, & EASTMOND, 2010), *Alice* (Pierce, 1998), *GreenFoot* (Kölling, 2010) e *AgentSheets* (Repenning & Ioannidou, 2004).

Durante o período em que fazia meu mestrado, o grupo de pesquisa do qual faço parte estava trabalhando com uma versão brasileira de um projeto chamado Scalable Game Design (SGD) (BASAWAPATNA, KOH, & REPENNING, 2010), originado na Universidade do Colorado em Boulder, EUA. Este projeto tem como objetivo a introdução do raciocínio computacional principalmente para alunos de ensino médio e fundamental através da criação de jogos e simulações. É utilizada uma ferramenta de programação visual especificamente desenvolvida para esta finalidade, o AgentSheets.

O AgentSheets é uma ferramenta onde o aprendiz pode construir jogos e simulações em duas dimensões usando uma linguagem de programação visual, denominada "Visual AgenTalk". Nessa construção, o aluno deve "Arrastar e Soltar" elementos visuais que correspondem a comandos da linguagem de programação . Um programa é, neste caso, uma sequência de regras no padrão "SE" e "ENTÃO", ou seja, um conjunto de regras de produção: "SE condição ENTÃO ação". Devido a essas duas características, o AgentSheets tem se mostrado uma ferramenta poderosa para o primeiro contato de alunos do ensino fundamental e médio com o raciocínio computacional.

No AgentSheets cada jogo é composto por uma ou mais planilhas de trabalho, onde os agentes são posicionados para executar seus comportamentos. Na Figura 1 abaixo, podemos ver essa planilha de trabalho (Número 2), em conjunto com o que o AgentSheets chama de Galeria de Agentes (Número 1). Nessa galeria podemos visualizar todos os agentes que estão prontos para serem colocados em alguma posição da planilha. Além disso, temos na Figura 1 a área Comportamento (Número 3), onde é feita a programação dos agentes. Nessa área é que descrevemos como os agentes irão atuar no jogo ou simulação, programando este comportamento através de manipulação direta de componentes e comandos.

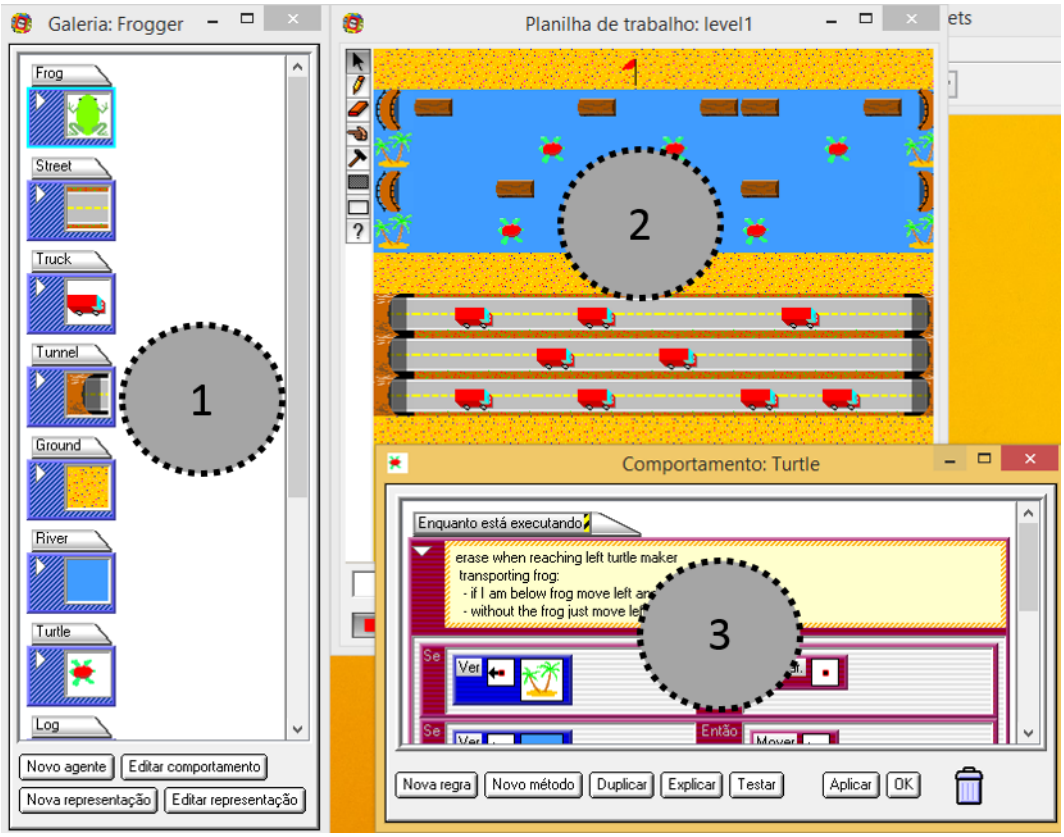

Figura 1 - Visão Geral do AgentSheets

Na versão brasileira do SGD, o SGD-Br, exploramos o raciocínio computacional de um outro ponto de vista. Acreditamos que a capacidade de entender e programar computadores não se resume a resolução de problemas complexos. Um usuário de computador, pode por exemplo, criar um artefato computacional não necessariamente complexo com o simples intuito de entreter algum amigo, ou de passar alguma mensagem para a sociedade. A esse fenômeno, damos o nome de "autoexpressão através de software" (Mota, 2014) (de Souza, Salgado, Leitão, & Serra, 2014) (Monteiro, 2015).

A autoexpressão através de software é um conceito que está dentro da teoria da Engenharia Semiótica, mas que já despontava nos primeiros estudos pioneiros de Papert (1991) e Turkle (2005) no MIT. O estudo aqui descrito também estará o tempo todo inserido no domínio da Engenharia Semiótica (de Souza C. S., 2005b), uma teoria semiótica da Interação Humano-Computador. O foco da teoria é o processo de metacomunicação. A interação humano computador é vista como uma comunicação mediada por computador que acontece entre o designer e o usuário do sistema em tempo de interação. O teor da mensagem do

designer para os usuários é como eles podem ou devem se comunicar com o software para fazer vários tipos de coisas (donde a noção de "metacomunicação", uma comunicação sobre como e por que comunicar). Esta inserção explica a perspectiva adotada no projeto SGD-Br em que o software é visto como uma forma de expressão e comunicação. Além disso, a Engenharia Semiótica oferece o embasamento teórico para os estudos aqui descritos.

Neste contexto, foi desenvolvido um software de apoio ao ensino do raciocínio computacional, o PoliFacets (Mota, 2014). Este software gera uma documentação ativa dos jogos criados pelos alunos (i. e. uma documentação que "reage" a interações que o usuário tem com o material documentado e, por isto, não tem uma forma única e definitiva de se apresentar para o usuário). O PoliFacets, como sugere o nome, oferece diferentes facetas para melhor exploração dos conceitos, significados e representações envolvidos na criação dos jogos. Através do PoliFacets, o aluno pode explorar as múltiplas relações possíveis entre o que deseja expressar e a forma como programar aquilo. As facetas presentes no sistema foram projetadas com a intenção de ajudar o aluno a entender melhor aquilo que ele mesmo criou, além de dar a oportunidade de ele incluir uma descrição própria na documentação gerada, adicionando mais significado ao seu jogo.

No contexto de ensino de raciocínio computacional através de jogos e simulações, como já assinalado acima, podemos encontrar diferentes ambientes de programação. Cada um evidencia ou esconde certos conceitos, de acordo com o entendimento e a intenção dos seus criadores. Só como um exemplo rápido, enquanto o AgentSheets enfatiza o uso de regras de produção na atividade de programação (v. Figura 1), seguindo uma linha muito usada em aplicações de Inteligência Artificial, o NetLogo (Wilensky, 1999) (que como o nome sugere é herdeiro do primeiro ambiente usado em ampla escala para ensinar programação para crianças, o Logo (Harvey, 1997)) salienta aspectos procedimentais que simplesmente não aparecem no AgentSheets (v. Figura 2).

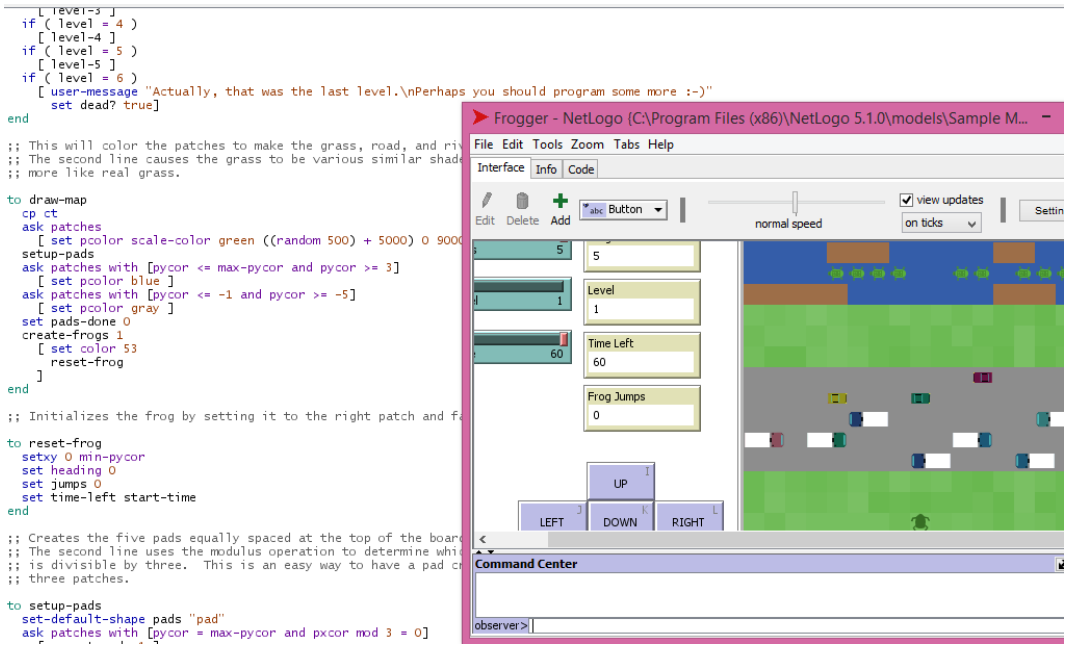

Figura 2 - Visão geral do NetLogo

Dentro da ciência da computação, encontramos diferentes formas de programar computadores. Essas diversas formas surgiram para o programador poder se adaptar a diferentes contextos, e também para otimizar a forma como elas são executadas pelo computador. Visto essa diversidade de opções, é natural que existam diversas formas de se ensinar a programação de computadores. A essas diferentes formas, damos o nome de paradigmas de programação (Floyd, 1979). Uma linguagem de programação pode evidenciar conceitos de um ou mais paradigmas e esconder conceitos de outros.

É verdade que não existe uma linguagem de programação "vencedora", o que sugere que várias delas (não sendo prático pensar que todas elas) devem ser ensinadas. Isto capacita o aprendiz a fazer escolhas ou identificar-se mais com uma ou com outra. No SGD-Br, isso já está sendo abordado. No projeto, apesar de ainda se limitar ao uso do AgentSheets, já abordamos com os alunos que existem várias linguagens de programação e que é importante capacitá-los a fazer a opção certa de qual usar dependendo da necessidade. Profissionalmente, essas diversas linguagens oferecem diferenças que devem ser entendidas e bem exploradas na hora de um profissional tomar a decisão de qual a melhor a ser usada. É importante que essas diferenças já sejam notadas pelo aluno desde o ensino básico para que ele não se perca no futuro.

O objetivo deste trabalho não foi buscar qual a melhor linguagem para ensinar programação de computadores. Tomamos como verdade que em alguma linguagem o aprendiz terá que iniciar. Logo, o compromisso deste trabalho foi buscar entender e oferecer tecnologia para auxiliar e apoiar a transição do aprendizado dos conceitos de uma linguagem para outra.

No decorrer da pesquisa, apoiados nos trabalhos de Kölling (2010), Denny (2011) e Dann (2012) percebemos, que podemos dividir as LP's de aprendizado em duas categorias: as que têm uma sintaxe definida, em geral LP's textuais; e as que têm uma sintaxe simples e apoiada pelo ambiente, em geral LP's visuais. Baseados na literatura, acreditamos que é mais eficiente que o aprendiz comece por uma LP visual, eliminando assim sua barreira sintática no aprendizado. Em cima disso, propomos um modelo de transferência do raciocínio computacional de uma LP visual para uma LP textual.

O modelo de transferência é uma expansão do modelo PoliFacets proposto por Mota (2014). Presumimos que o aprendiz tenha passado pelo PoliFacets usando uma linguagem de programação visual para então seguir em direção a uma nova linguagem textual apoiado pelo modelo de transferência de raciocínio computacional.

Após a definição do modelo de transferência foi realizado um estudo qualitativo com alguns usuários que já estavam familiarizados com o modelo PoliFacets. Esse estudo teve a intenção de realizar uma prova de conceito do modelo proposto e buscar entender quais pontos podem ser melhorados em trabalhos futuros. O modelo em si é abstrato e teoricamente adaptável a quaisquer duas linguagens de programação, desde que respeitado o fato que a LP de origem é visual e a de destino é textual.

Para fins do estudo, realizamos uma implementação em mockups do modelo em cima de duas linguagens de programação. A LP visual AgentSheets foi escolhida como a LP de origem devido ao amplo conhecimento que já temos dessa linguagem após cinco anos de andamento do projeto SGD-Br. Além disso, teríamos disponibilidade de usuários já treinados em AgentSheets o que poderia não existir em outras linguagens.

Para LP textual de destino, a escolha foi o GreenFoot. O GreenFoot é ambiente de programação textual para auxiliar na aquisição de raciocínio computacional através da criação de jogos e simulações. Esta ferramenta permite

um fácil desenvolvimento de aplicações gráficas de duas dimensões utilizando a linguagem de programação Java como sua fonte de codificação. A escolha de GreenFoot se deu por algumas razões. O contexto de jogos e simulações é o mesmo que o AgentSheets. Além disso, a estrutura de definição dos projetos é bem parecida. Enquanto em AgentSheets programamos separadamente cada Agente e Planilha, em GreenFoot programamos separadamente cada Ator e Mundo. Outra razão fundamental para escolha do GreenFoot foi a sua proximidade com a LP profissional Java e a existência de um projeto de transferência do conhecimento de GreenFoot para Java através da IDE BlueJ.

Nas Figura 3 e Figura 4 abaixo, temos uma comparação entre a interface do AgentSheets e do GreenFoot.

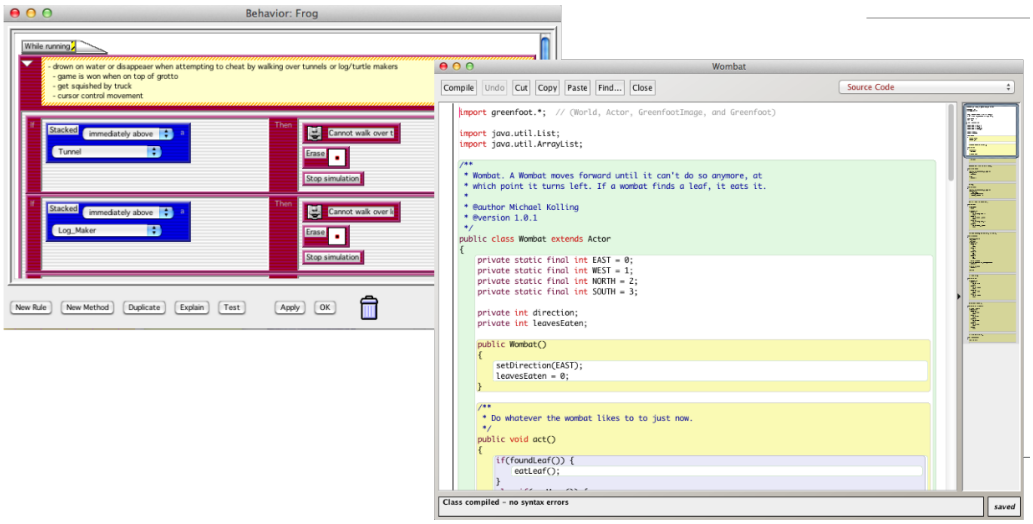

Figura 3 - Agentes e Atores

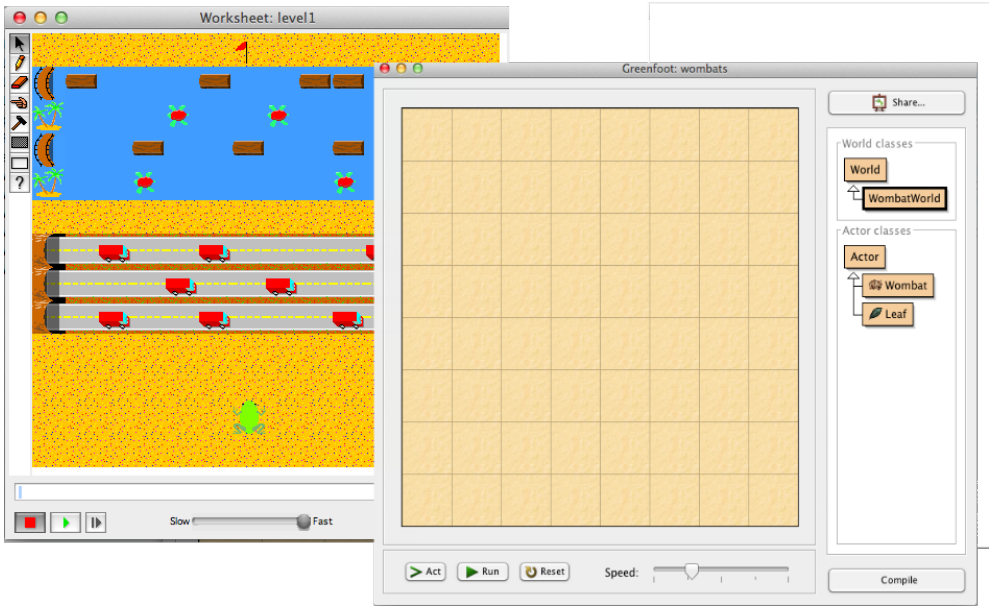

Figura 4 - Planilha e Mundo

Os resultados desse estudo nos serviram para prova de conceito do modelo proposto. Conseguimos identificar falhas e sucessos ao observar os usuários interagindo com o sistema. No capítulo 6 discutiremos com mais detalhes esses resultados, assim como proporemos trabalhos futuros para melhorias e correções no modelo proposto.

O texto dessa dissertação está organizado em 6 capítulos. Após este primeiro capítulo de introdução temos o capítulo 2 descrevendo os principais trabalhos relacionados a essa pesquisa. Em seguida, no capítulo 3, temos uma descrição das ferramentas que foram utilizadas no apoio ao desenvolvimento da pesquisa. No capítulo 4, temos uma descrição da estratégia de pesquisa e os principais resultados encontrados. No capítulo 5 faremos uma discussão preliminar do modelo proposto e por último no capítulo 6 temos a conclusão e os trabalhos futuros que podem surgir dessa pesquisa.

2
Trabalhos fundamentais para a pesquisa

Os trabalhos fundamentais a esta pesquisa podem ser classificados de acordo com três categorias a serem discutidas a seguir. Primeiro vamos discutir questões de ensino de programação e do raciocínio computacional, seja no ensino fundamental e médio, seja no ensino profissionalizante. Uma segunda categoria é a dos estudos que levam em conta a presença ou não de sintaxe em uma linguagem de programação. Por último, vamos falar de trabalhos que envolvem a utilização de dois ou mais ambientes de programação.

2.1
Questões de ensino

O ensino de computação é um campo de pesquisa com vasta literatura. Como o foco deste trabalho é dar suporte tecnológico ao ensino de raciocínio computacional através de ambientes próprios para o aprendizado, vamos nos concentrar somente em trabalhos com essa característica. Começaremos por definir o termo raciocínio computacional. Como já dissemos, ele foi definido pela primeira vez por Wing (2006). A autora defende que o raciocínio computacional é uma habilidade fundamental para todos, não apenas para graduados na área de computação, e envolve a formulação, a compreensão e a solução de problemas. O raciocínio computacional é a capacidade de usar a abstração e a decomposição para resolver tarefas complexas ou projetar sistemas complexos.

Atualmente temos diversos ambientes de aprendizado que evocam essa definição de Wing para raciocínio computacional. O ambiente *Scratch*, tido como o mais usado atualmente, é definido por seu criador como um ambiente de programação simples, voltado para crianças a partir dos 8 anos de idade (MALONEY, RESNICK, RUSK, SILVERMAN, & EASTMOND, 2010). Em trabalho de 2010, Maloney *et al* descrevem o funcionamento e o design da ferramenta Scratch (MALONEY, RESNICK, RUSK, SILVERMAN, & EASTMOND, 2010). Eles defendem que uma transferência de Scratch para uma linguagem de programação textual deve ser incentivada, citando inclusive o

programa GreenFoot como uma opção para essa transferência. Os autores porém não entram a fundo nesse assunto, deixando como trabalho futuro tal abordagem.

Outro ambiente de programação amplamente utilizado tanto no ensino fundamental quanto no superior é o Alice 3D. Em seu trabalho de 2010, *The Design of Alice* (COOPER, 2010), Stephen Cooper descreve o processo de design de sua ferramenta, bem como cita quais conceitos ela explora durante o processo de aprendizado. Fica claro a intenção de Alice 3D ser uma linguagem que busca enfatizar conceitos de orientação objeto. Além disso, o autor afirma que o ambiente faz uma comparação um-para-um entre o que o aprendiz está programando na linguagem visual e o que seria a programação na linguagem textual Java, que é orientada a objetos. Esse trabalho claramente tangencia minha questão de pesquisa quanto a transferência de aprendizado, porém o autor não se aprofunda no tema.

Em se tratando de ambientes de programação que utilizam de linguagens textuais para o aprendizado temos como destaque o ambiente GreenFoot. No artigo *The Greenfoot Programming Environment* (Kölling, 2010), o autor define o GreenFoot como um ambiente de aprendizado voltado para alunos desde 14 anos de idade até o ensino superior. O autor diz que não é obrigatório, porém é desejável que o aprendiz tenha passado por um ambiente de programação visual antes de começar seu aprendizado em GreenFoot. Mais uma vez a questão de transferência do conhecimento fica latente porém inexplorada pelo autor que cita o Scratch e o Alice como dois bons ambientes para iniciar o aprendizado de programação.

Nesses três trabalhos citados acima, vemos que a transferência de conhecimento é vista como desejável. É certo que todos incentivam que o aprendiz passe por mais de uma linguagem de programação durante o processo de aprendizado porém nenhum desses três trabalhos se posiciona em relação a *como fazer essa transferência*.

2.2
Questões sintáticas

A sintaxe em uma linguagem de programação textual muitas vezes é o principal desafio para o aprendizado do aluno. Em seu trabalho publicado em

2011 (Understanding the Syntax Barrier for Novices), Paul Denny diagnosticou que mesmo aprendizes mais avançados tiveram problemas em mais de 50% dos casos estudados. Para Denny, sintaxe é um grande problema nos estágios iniciais do aprendizado. Os alunos ficam presos em erros de sintaxe e não dedicam seu tempo para o que realmente importa, a lógica de programação. Esse trabalho nos serviu de base para propormos que o modelo deve ser sempre direcionado de uma LP visual para uma LP textual. Assim, o desafio sintático para o aluno só viria após ele ter algum conhecimento de lógica de programação.

Kölling, em seu trabalho de 2010 também toca na questão da sintaxe. Para ele, aprendizes com menos de 14 anos ainda não têm maturidade suficiente para lidar com a sintaxe em uma LP (Kölling, 2010). Assim, eles devem começar por um ambiente de programação que elimine o problema sintático, deixando que o aprendiz foque apenas na resolução do problema proposto. Nesse trabalho, Kölling sugere ambientes como *Scratch* e *Alice 3D* como bons ambientes para o aluno começar antes de avançar para o GreenFoot.

Para Tebring Daly, a ausência de sintaxe em uma linguagem de programação voltada para o ensino é fundamental. No trabalho *Minimizing To Maximize: An Initial Attempt At Teaching Introductory Programming Using Alice* (Daly, 2011), Daly mostra através de um estudo quantitativo que a ausência de sintaxe na linguagem de programação Alice 3D ajudou os aprendizes a fixarem melhor os conceitos de programação. Para ele, ao começarem pelo Alice, sem as barreiras sintáticas, os alunos puderam aprender melhor os conceitos sem se preocupar com erros de sintaxe, construção de códigos desnecessários e etc.

Esses trabalhos citados foram fundamentais para definição do modelo proposto. Diagnosticamos que sintaxe é de fato uma grande barreira no processo de aprendizado e passamos a dar um tratamento especial a esse problema.

2.3
Trabalhos relacionando duas ou mais linguagens de programação

Em geral, os autores concordam que há a necessidade de se ensinar mais de uma forma de programar ao aluno. Em especial, os trabalhos confrontam os paradigmas de programação utilizados por cada linguagem. Apesar de um aprofundamento nas questões teóricas de paradigmas de programação não ser o

foco desse estudo, vale ficar atento ao que está se falando nessa área. No artigo "A three paradigm first course for CS majors" (Reinfelds, 1995), Juris Reinfelds apresenta um currículo de dois semestres para introdução à ciência da computação. Ele descreve três diferentes paradigmas de programação (Imperativo, Funcional e Lógico) e como o ensino de cada um deles contribuiu para o desenvolvimento de determinados conhecimentos na área da informática.

O autor conclui que o resultado desta abordagem é ter alunos com maior capacidade de resolver problemas diversos. Ele cita como exemplo a comparação entre dois grupos de alunos (participantes e não participantes do curso). Ao passar uma atividade onde o uso de uma linguagem funcional deixaria a resolução do problema mais fácil, os participantes do curso usaram *Prolog* e conseguiram resolver o problema de forma mais simples e rápida do que os não participantes, que por sua vez acabaram usando alguma linguagem imperativa. A partir desse resultado, podemos crer que quanto maior o conhecimento do aluno, maior sua capacidade de resolver problemas e de se autoexpressar através de *software*.

Olhando mais para a transferência entre linguagens de programação específicas, temos o trabalho "Mediated transfer: Alice 3 to Java" (Dann, Cosgrove, Slater, Culyba, & Cooper, 2012). Nesse trabalho, Wanda Dann *et al.* descrevem uma pedagogia para início do curso de ciência da computação, onde os alunos começam a trabalhar no ambiente "Alice" e somente depois, com algum conhecimento adquirido nessa fase, eles fazem uma transferência de conhecimento para linguagem de programação Java. Essa transferência é baseada em uma técnica chamada "*mediating transfer*" (Forgarty, Perkins, & & Barell, 1991) ou mediação por transferência, numa tradução livre, onde o professor procura criar pontes entre conceitos aprendidos no contexto anterior para o novo contexto, através de metáforas e comparações.

Essa técnica de mediação por transferência se dá através da criação de pontes entre os conceitos explorados nos diferentes ambientes. Os autores descrevem situações em que o Alice 3 explora conceitos de forma mais simples, mas já criando a ponte para o mesmo conceito, mais complexo em Java.

Por último, temos um trabalho que se apresenta como algo intermediário entre o extremo das linguagens de programação visuais e o das linguagens de programação textuais. No artigo *"Heterogeneous Visual Languages - Integrating Visual and Textual Programming"* (Erwig & Meyer, 1995), os autores apresentam

um esquema que integra linguagens visuais com linguagens textuais em um mesmo ambiente de programação. O objetivo desse esquema, segundo os autores, é aproveitar o melhor de cada forma de programar. Para eles, linguagens visuais e textuais têm seu potencial máximo explorado dependendo de cada situação.

Com esse esquema, espera-se que o programador tenha mais facilidade em resolver os problemas, usando o visual e o textual quando lhe for mais conveniente. Certamente isso vai ao encontro dos resultados que esperamos encontrar nesta dissertação. Ao trabalharmos com uma transferência de conhecimento entre uma linguagem visual para uma linguagem textual, desejamos fortemente que o aluno aprendiz seja capaz de identificar em quais aspectos uma linguagem pode ser melhor explorada do que a outra.

Assim, imaginamos que ao final da transferência entre o visual e o textual, o aprendiz esteja apto a embarcar em um esquema como o proposto pelos autores, aproveitando o que cada ambiente pode oferecer de melhor durante o processo de desenvolvimento do software.

2.4 Outros Trabalhos relacionados

Além dos trabalhos descritos anteriormente, também foram fundamentais para essa pesquisa uma série de outros trabalhos relacionados que de alguma forma influenciaram nossa linha de pesquisa. No artigo "*Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers*" (Kelleher & Pausch, 2005) os autores fizeram uma classificação de diversos ambientes de programação, separando quais conceitos cada ambiente explora. Essa classificação serviu como inspiração para aquilo que veio a ser a base dessa dissertação e será apresentada mais a frente. Os autores fizeram uma classificação da existência do conceito ou não em cada ambiente. Nessa dissertação, optamos por uma abordagem um pouco diferente, onde introduzimos a ideia de que um conceito teórico pode existir por si só, como pode também aparecer apenas como um efeito. Tal definição ficará mais clara no capítulo em que o modelo aqui proposto é defendido.

Por último, temos o artigo "*Alice, Greenfoot, and Scratch – A Discussion*" (UTTING *et al.*) onde os autores de cada ambiente fazem uma discussão a

respeito do que cada ambiente tem de melhor a ser explorado. Baseado principalmente nesse trabalho que decidimos utilizar do *GreenFoot* em nossos estudos nessa dissertação. Durante a discussão, os autores não falam diretamente a respeito de uma transferência de conhecimento, porém deixam claro que existe uma sequencia natural de exploração entre esse três ambientes, partindo do *Scratch* (mais simples) até o *GreenFoot* (mais complexo).

A conclusão a que chegamos na revisão de trabalhos relacionados ao nosso tópico de pesquisa é que, embora haja vários trabalhos afirmando que o tópico é relevante, ou demonstrando por que os alunos devem criar uma cultura de uso de múltiplas linguagens de programação, ou apontando para o fato de que um aprendizado inicial com linguagens visuais tem de ser seguido por um aprendizado de linguagens de programação textuais, nenhum deles propõe um modelo de ferramenta de apoio para a transferência do aprendido com uma LP visual para o próximo passo em uma LP textual. Nos próximos capítulos mostraremos a nossa proposta para isto.

3
AgentSheets, GreenFoot e PoliFacets

Para facilitar a compreensão e contextualização nos capítulos seguintes, serão apresentadas agora as três ferramentas que foram usadas durante a nossa pesquisa. Fundamentais do início ao fim dos estudos, essas ferramentas colaboraram tanto na busca pelas evidências em que nos baseamos na proposta de um modelo de ferramenta de suporte para a transferência de conceitos de programação adquiridos com LPs visuais para LPs textuais, quanto para os estudos de prova de conceito do que foi proposto.

3.1
AgentSheets

Como descrito na introdução, o AgentSheets é uma ferramenta de programação visual por manipulação direta. Ela é indicada para criação de jogos e simulações em 2D por aprendizes com pouco ou nenhum conhecimento de programação (Repenning & Ioannidou, 2004). Desde sua origem tem se mostrado uma ótima ferramenta para pesquisas sobre o desenvolvimento do raciocínio computacional, em especial, em estudos com alunos dos ensinos fundamental e médio.

A programação é feita através do "arrastar e soltar" de condições e ações que formam regras de comportamento para os personagens do jogo, os Agentes. Os aprendizes criam seus jogos e simulações sem ter que digitar nenhum código textual ficando livres de preocupações com possíveis erros sintáticos. Abaixo, mostramos uma figura com interface geral do AgentSheets (Figura 5).

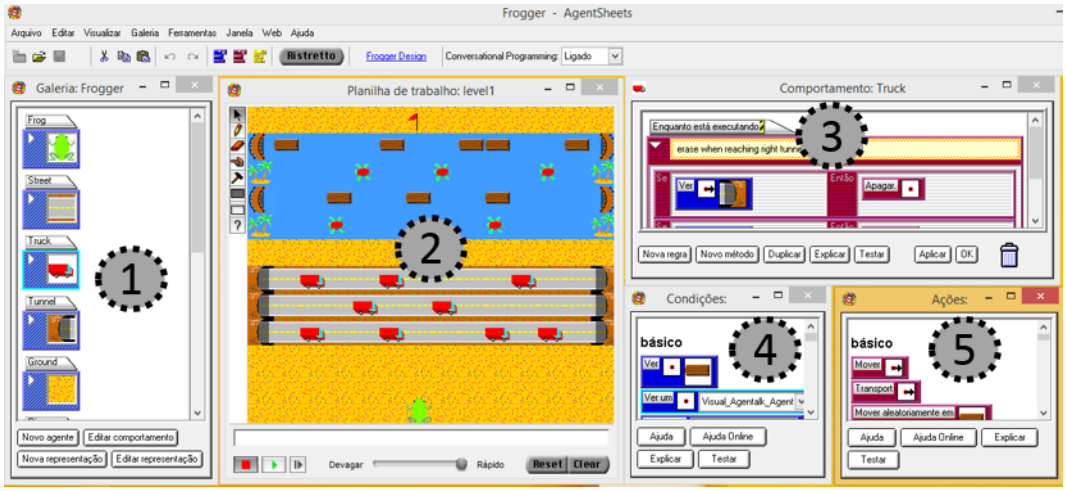

Figura 5 - Interface geral do AgentSheets

Todos os projetos criados no AgentSheets possuem os seguintes elementos:

Planilhas: Uma planilha (Número 2) é uma grade bidimensional onde estão posicionados (estaticamente) e atuam (dinamicamente) todos os objetos do jogo. A criação de uma planilha no AgentSheets é feita posicionando cada agente na grade utilizando-se as ferramentas de desenho dispostas do lado esquerdo da janela em que está a planilha. Além dos agentes, uma planilha pode ter uma imagem de fundo para auxiliar na aparência do projeto. (Veja na Figura 6 um exemplo de projeto que utiliza imagem de fundo).

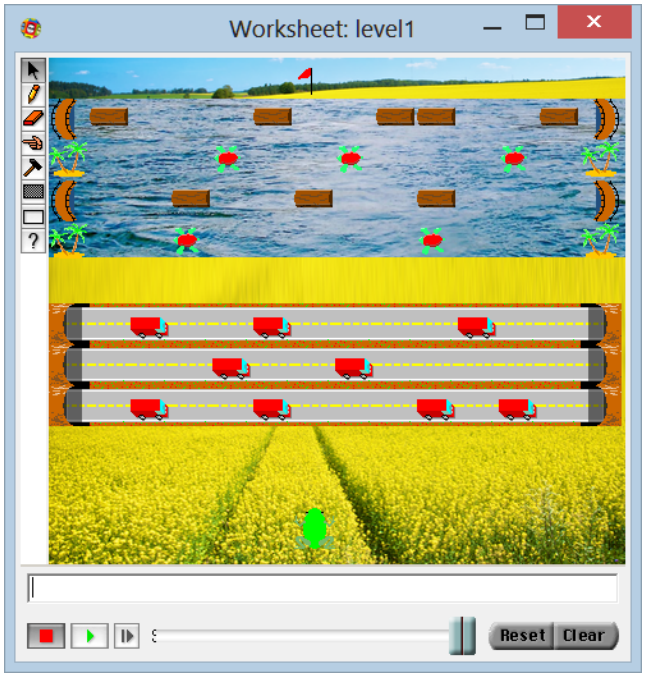

Figura 6 - Projeto utilizando imagem de fundo

Agentes: São os personagens do projeto que podem ter uma função específica na interação ou podem ser meramente decorativos. Cada agente é programado separadamente através das regras, que são descritas abaixo.

Regras: As regras (Número 3) no AgentSheets têm formato “Se-Então”, conhecido como padrão de regras de produção. O usuário programa o agente através do arrastar e soltar de condições (Número 4) (associadas à parte "Se") e ações (Número 5) (associadas à parte "Então). Um conjunto (potencialmente vazio) dessas regras “Se-Então” definem o comportamento de cada agente do jogo. Agentes cujo comportamento é definido por um conjunto vazio de regras não fazem nada durante a execução do jogo (mas podem servir de contexto para a ação de outros agentes, por exemplo).

3.2 GreenFoot

O GreenFoot é uma ferramenta de programação textual baseada na linguagem de programação Java e no framework GreenFoot. Ela é indicada para criação de jogos e simulações em 2D. Pode ser usada desde o ensino fundamental até as disciplinas iniciais de curso de computação no ensino superior (Kölling, 2010). Kölling, o criador da ferramenta, indica que o GreenFoot não deve ser usado por crianças com menos de 14 anos de idade devido à presença da sintaxe da sua LP textual. Assim sendo, em um currículo que envolva o ensino de programação desde o início do ciclo da educação básica, o GreenFoot teria que ser precedido por alguma LP visual.

A programação em GreenFoot é feita através da codificação dos Atores e Mundos usando a linguagem de programação Java. Como apoio ao aprendiz, para que o mesmo não tenha que lidar com temas mais complexos de computação gráfica, a ferramenta disponibiliza um framework com algumas funções definidas. Tais funções ajudam o aprendiz a definir a movimentação dos atores e o mapeamento dos mundos. Abaixo, mostramos uma figura da interface geral do GreenFoot (Figura 7).

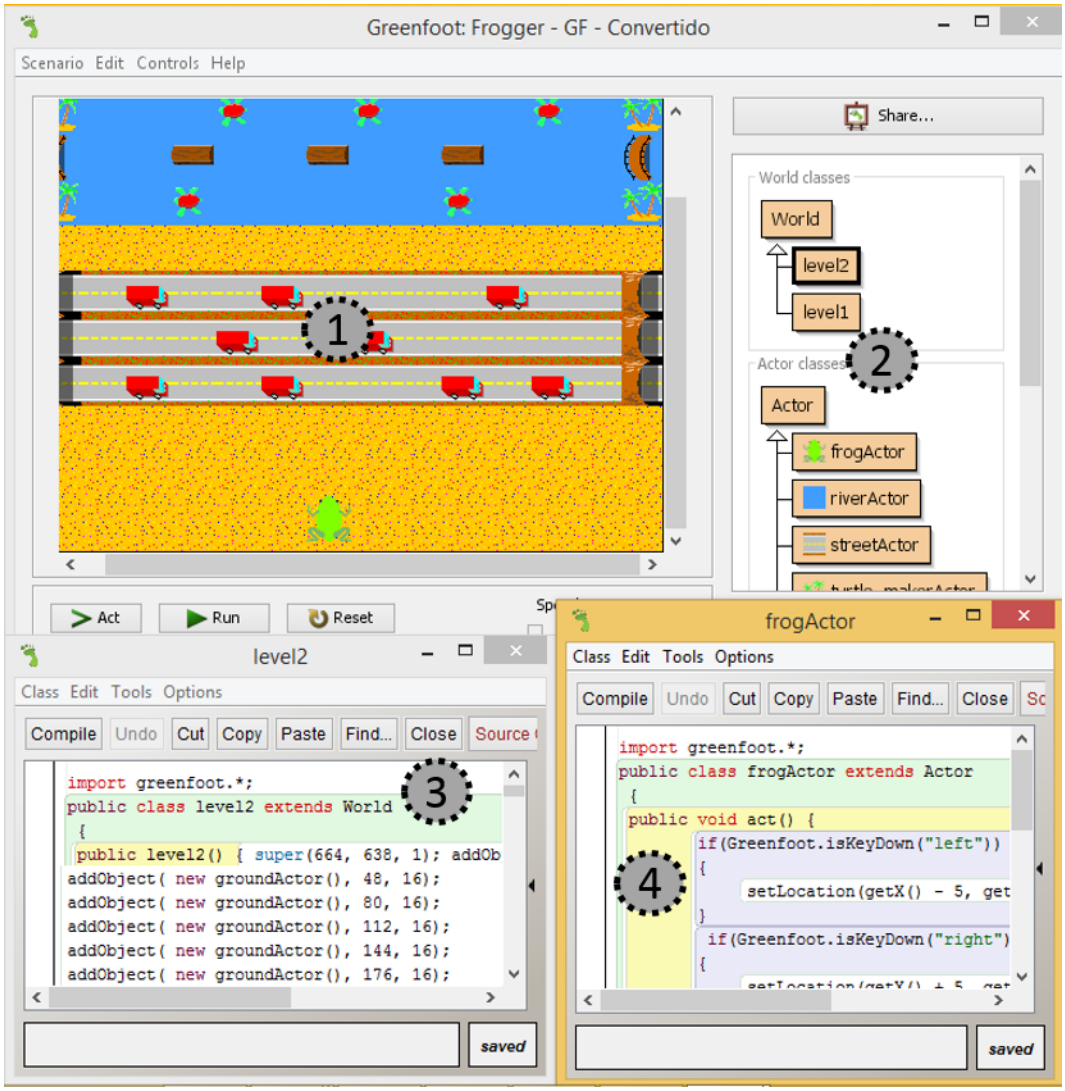

Figura 7 - Interface geral do GreenFoot

Todos os projetos em GreenFoot possuem os seguintes elementos.

Mundos: Um mundo (Número 1) é uma grade bidimensional onde os personagens do cenário interagem. Todo objeto que se destina a ser um mundo tem que herdar da classe World e programar individualmente como deve ser preenchida a grade (Número 3). Essa programação é feita com a linguagem de programação Java utilizando funções específicas do *framework greenfoot* para adicionar elementos a grade.

Atores: Todo objeto que se destina a aparecer em um mundo é uma classe do tipo Actor. Atores são os personagens de um Cenário no GreenFoot. Têm sua própria aparência e comportamento programados individualmente (Número 2).

Código Fonte: Ambos, Atores e Mundos devem ser programados utilizando-se a linguagem de programação Java e o *framework greenfoot.* O

conjunto de todos Atores e mundos constituem o código fonte do projeto (Número 4).

3.3 PoliFacets

O PoliFacets é uma ferramenta de apoio a alunos e professores durante o processo de aprendizado do raciocínio computacional. Concebido durante a pesquisa do SERG (Semiotic Engineering Research Group) no projeto SGD-Br, ele foi implementado para dar suporte ao ensino-aprendizado de programação de jogos e simulações com a ferramenta AgentSheets. Posteriormente foi estendido e abstraído como um modelo de documentação ativa por Mota em sua tese de doutorado (Mota, 2014). Não entraremos em detalhes no modelo proposto por Mota aqui, nosso foco é apenas a ferramenta.

Seu principal objetivo é a desconstrução, e através dela a documentação detalhada, de um projeto feito em AgentSheets em diversas facetas, onde o aprendiz pode explorar o seu projeto de diferentes visões. Cada faceta do PoliFacets fornece ao usuário uma nova perspectiva de seu projeto, facilitando, entre outras coisas, que ele entenda melhor o que foi programado, perceba alternativas de programação e, em processo de análise e destaque de elementos que constituem o programa, encontre erros que podem estar escondidos quando se examina a programação no ambiente do AgentSheets. Abaixo a Figura 8 mostra a interface geral do sistema PoliFacets. Na parte superior direita da figura (Número 1) podemos ver links para as seis facetas. Entraremos em detalhes mais adiante no texto sobre a faceta “Regras” que será aproveitada na instância do modelo aqui proposto. A Figura 8 também exibe parte do conteúdo da faceta Descrição (Número 2) para o jogo Frogger (Número 3). No menu à esquerda (número 4) aparecem diversas opções de acesso a outras partes do sistema PoliFacets e/ou a websites e recursos relacionados ao SGD-BR na Web.

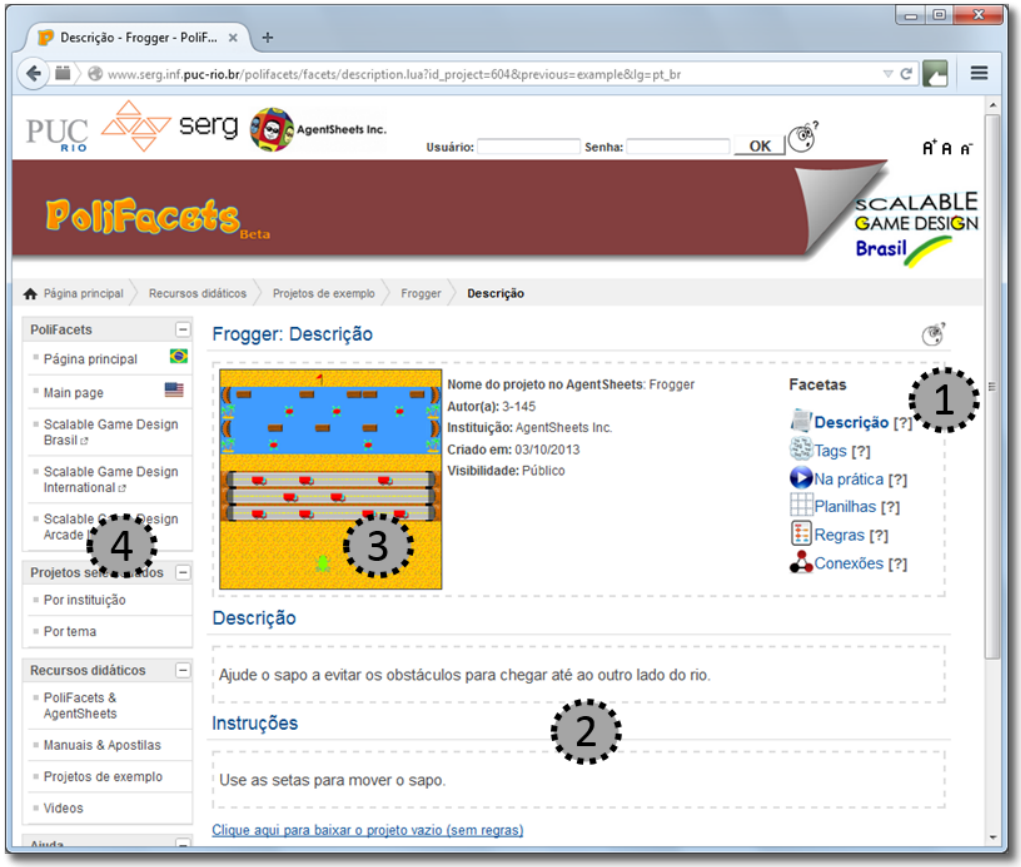

Figura 8 - Interface Geral do PoliFacets

O PoliFacets é um sistema que pode ser visitado na Web no endereço: www.serg.inf.puc-rio.br/polifacets (vários recursos do sistema podem ser explorados sem a necessidade de usuário e senha). Nas atividades do SGD-Br, professores e alunos das escolas parceiras tinham acesso autenticado à área reservada do sistema, com acesso a suas turmas, jogos e simulações de outras turmas de sua escola.

4
Uma ferramenta de apoio à transferência de conhecimento sobre programação no ambiente PoliFacets

Nesse capítulo serão descritas as estratégias de pesquisa e os resultados encontrados a partir dos estudos realizados. No primeiro tópico falaremos da estratégia utilizada na pesquisa e quais etapas foram executadas. Depois falaremos dos resultados do estudo prévio feito com a ferramenta AgentSheets. Tais resultados foram fundamentais para a elaboração do modelo proposto na pesquisa. Em seguida, como resultado principal da dissertação apresentaremos o modelo de transferência de raciocínio computacional de uma LP visual para uma LP textual. No capítulo seguinte discutiremos o modelo em um novo estudo empírico que foi realizado para prova de conceito.

4.1
Estratégia de pesquisa

Toda a pesquisa desenvolvida até aqui no projeto SGD-Br utilizou métodos qualitativos, isto é, realizou estudos aprofundados tomando por evidência empírica pequenas quantidades de dados coletados. Durante os dois anos de mestrado fiz parte deste projeto que me colocou em contato com alunos, professores e seus respectivos projetos criados com o AgentSheets. A primeira etapa da minha pesquisa foi a análise dos tipos de dados coletados em uma das turmas que acompanhei.

Durante o segundo semestre do ano de 2013, participei de todas as aulas de uma turma em uma escola parceira do projeto. Por se tratar de uma escola internacional, ela segue o período letivo e a estrutura didática das escolas do país de origem. A turma acompanhada era uma turma no primeiro semestre do oitavo ano. As aulas em que o projeto foi aplicado eram obrigatórias e faziam parte da grade curricular da escola.

Para a realização dessa primeira etapa da pesquisa, foram analisados todos projetos submetidos ao sistema PoliFacets durante o período do projeto nessa

turma. No total, foram submetidos 36 projetos de 20 alunos. Desses, temos projetos em resposta a um pedido do professor para os alunos replicarem jogos de exemplo do AgentSheets (30 projetos), bem como projetos autorais (6 projetos), onde o aluno criou sua própria narrativa e descrição do jogo.

Nessa análise dos projetos buscávamos responder quais conceitos de programação os alunos estavam implementando em seus projetos. Na seção seguinte veremos como cada comando do AgentSheets pode evocar um conceito de programação. Através dessa análise buscamos realizar um corte para focar nos elementos e conceitos mais usados na programação com o AgentSheets. Assim levantamos dados numéricos sobre quais são os comandos mais usados na base da turma, e de que forma os alunos interagiram com (ou exploraram) o ambiente.

A partir desse estudo, cujos resultados estão no capítulo seguinte, partimos para a segunda etapa da pesquisa. Nessa etapa, nos baseamos nos resultados do estudo anterior e nos trabalhos encontrados na literatura para propormos um modelo de transferência do raciocínio computacional de uma linguagem de programação visual para uma linguagem de programação textual.

O modelo final proposto é resultado de sucessivos ciclos de aperfeiçoamento. Em cada ciclo, apresentávamos uma implementação concreta do modelo corrente a especialistas em IHC para fazermos o equivalente a uma crítica por pares. Recebíamos feedbacks com opiniões e sugestões de melhorias e aperfeiçoávamos baseado nesses feedbacks e de novas revisões da literatura da área.

Após chegarmos a uma versão satisfatória do modelo, partimos para a terceira etapa do estudo. Nessa etapa, foram implementados *mockups* para uma versão concreta do modelo. Essa instância do modelo foi baseada nas linguagens de programação AgentSheets (visual) e GreenFoot (textual). De posse desse sistema de mockups partimos para realização de um estudo final qualitativo envolvendo três usuários.

O estudo realizado bem como seus resultados estão detalhados em uma seção mais à frente. Esse estudo nos possibilitou realizar uma prova de conceito do modelo proposto. Tal prova de conceito não pode ser vista como uma validação pois não é automaticamente aplicável para uma generalização do modelo. Porém, conseguimos mostrar que o sistema baseado no modelo ajudou os

aprendizes consultados na avaliação a transferirem conhecimentos da linguagem AgentSheets para a linguagem GreenFoot.

4.2
Estudo com Projetos de Programação com AgentSheets

O estudo feito com AgentSheets nos possibilitou enxergar quais conceitos são explorados por esse ambiente de programação. Foram analisados 36 jogos de 20 alunos e o resultado foi um levantamento numérico de quais conceitos os alunos exploram mais.

Tabela 1 - Quantas vezes ocorre cada comando

Quantas vezes ocorre cada comando				
Condição			**Ação**	
SEE	599		MOVE	415
KEY	185		MESSAGE	275
ONCE-EVERY	176		ERASE	182
IS	100		CHANGE	115
STACKED	85		WAIT	112
%-CHANCE	48		SET	90
STACKED-A	7		RESET-SIMULATION	77
SEE-A	5		BROADCAST	67
EMPTY	2		NEW	59
			SHOW_MESSAGE	50
			SWITCH-TO-WORKSHEET	40
			SAY	31
			STOP-SIMULATION	30
			TRANSPORT	22
			PLAY-SOUND	15
			MOVE-RANDOM-ON	6
TOTAL	1207		TOTAL	1586

Tabela 2 - Em quantos projetos aparece cada comando

Em Quantos projetos ocorre cada comando				
Condição			**Ação**	
SEE	36		MOVE	36
KEY	36		CHANGE	28
ONCE-EVERY	27		RESET-SIMULATION	28
STACKED	21		SWITCH-TO-WORKSHEET	25
%-CHANCE	21		ERASE	24
IS	11		NEW	24
STACKED-A	4		MESSAGE	23
SEE-A	3		WAIT	20
EMPTY	2		SHOW_MESSAGE	19
			BROADCAST	15
			SAY	12
			STOP-SIMULATION	12
			TRANSPORT	12
			SET	11
			PLAY-SOUND	9
			MOVE-RANDOM-ON	3

Na Tabela 1 acima temos um quadro de quais são os comandos mais usados e na Tabela 2 temos com qual frequência eles aparecem no total de jogos examinados. Na Tabela 1, temos uma totalização de comandos recorrentes nos 36 projetos analisados. Já na Tabela 2, temos a totalização do número de programas em que cada comando aparece.

A partir destas tabelas, conseguimos fazer várias observações importantes. Por exemplo, os comandos “IS” e “SET” são usados para manipulações de variáveis no código AgentSheets e ocorreram em 11 programas. Ou seja, não só o AgentSheets aborda, de certa forma, o conceito de “Manipulação de Variáveis”, mas a turma observada utilizou este mecanismo (quer o professor tenha chamado a atenção para o conceito, quer não). Veja na Figura 9 abaixo um exemplo do uso desses comandos retirados da amostra analisada. Para melhor compreensão dos

resultados expostos, veja no Apêndice 1 a lista de comandos do AgentSheets e suas respectivas explicações.

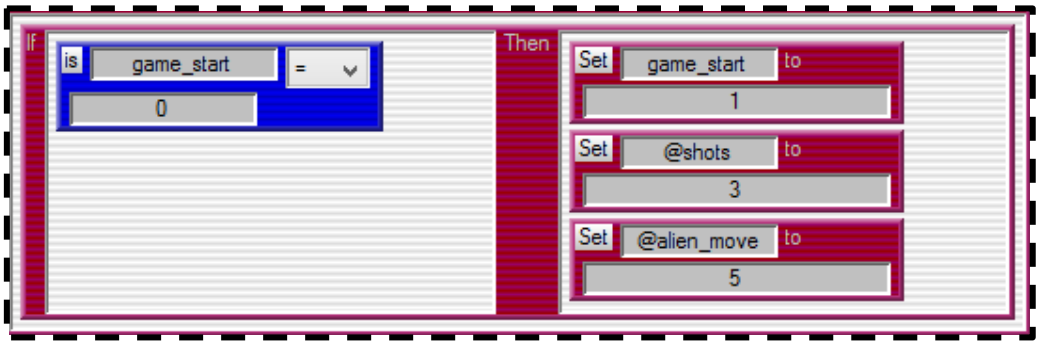

Figura 9 - Comandos IS e SET no AgentSheets

Na Tabela 3 temos a listagem das principais combinações (CONDIÇÃO -> AÇÃO) que aparecem nos dados empíricos coletados junto à turma observada.

Tabela 3 - Combinação de condições / ações

	Condição	Ação
65	SEE	ERASE
55	KEY	MOVE
54	SEE	MESSAGE; MOVE
38	SEE	MOVE
33	SEE	MOVE; MESSAGE
33	IS	BROADCAST
32	SEE	MESSAGE
29	SEE	CHANGE
25	SEE; KEY	MOVE
23	SEE; ONCE-EVERY	MOVE
23	KEY; SEE	MOVE
19	%-CHANCE; SEE; ONCE-EVERY	NEW
16	IS	SET; SET
15	ONCE-EVERY; SEE	MOVE
13	ONCE-EVERY; %-CHANCE; SEE	NEW
13	KEY; SEE	MESSAGE
13	ONCE-EVERY	SET; MESSAGE

13	SEE; IS	MOVE
11	SEE; KEY	MESSAGE
9	SEE	RESET-SIMULATION
9	ONCE-EVERY	MESSAGE; SET

Na Tabela 3, podemos constatar que os alunos fizeram bastante uso da combinação de mais de um comando, tanto na parte de Condição (à esquerda), quanto na parte de Ação (à direita). Do lado esquerdo da tabela, percebemos a combinação de condições, separadas por ";". No AgentSheets, quando duas condições estão sendo combinadas, isto significa uma conjunção ('e' lógico). Já do lado direito da tabela, o separador ";" usados em combinação de ações não significa uma conjunção lógica, mas sim uma execução sequencial de comandos. Veja na Figura 10 um exemplo de combinação de condições (Número 1) e de ações (Número 2).

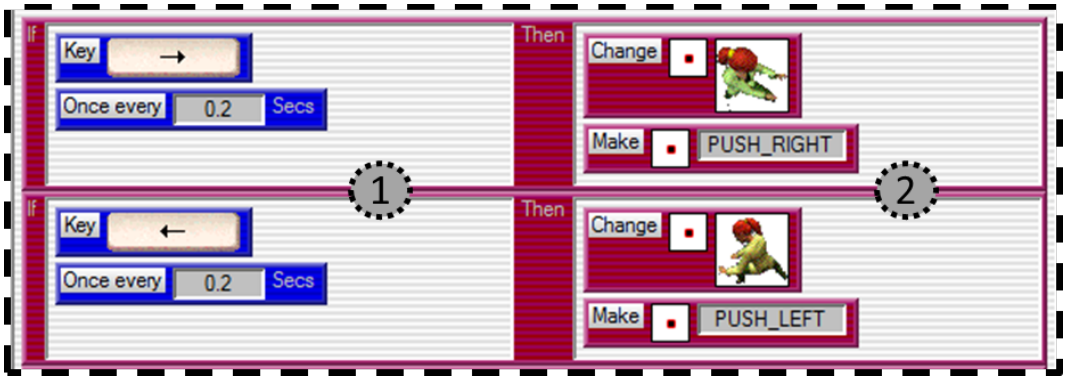

Figura 10 - Exemplo de Conjunções e Sequências de Ações

Além da análise dos comandos mais usados, foi feito também um levantamento de quais "*triggers*" estão sendo usadas no AgentSheets. Uma *trigger* no *AgentSheets*, é uma sequência de regras (condições / ações) que executam de acordo com o gatilho nela determinado. O *AgentSheets* conta com 5 tipos de *triggers*. São elas: a *trigger WHILE-RUNNINNG*, que é executada em um laço infinito enquanto o projeto está em execução; a *trigger ON*, que é nomeada e pode ser chamada através de um comando específico por qualquer agente; a trigger *WHEN-CREATING-NEW-AGENT,* que é executada toda vez que um novo agente é criado; a *trigger TOOL*, que é executada quando ferramentas da planilha é usada, ou seja, um agente pode ganhar comportamentos específicos em tempo

de execução; e a *trigger MOUSE*, que capta eventos do mouse do computador e reage a eles de acordo com as configurações.

A trigger *WHILE-RUNNING* expõe o conceito de "repetições", já que ela fica executando o tempo todo, enquanto o programa está ativo. Apesar de não ser obrigatória, esta trigger está presente em quase todos os agentes nos 36 jogos analisados. Como o *AgentSheets* sempre adiciona automaticamente essa trigger ao criar um novo agente, ele pode estar levando os aprendizes a crer que ela é obrigatória. Veja na Figura 11 um exemplo do uso dessa *trigger* retirado da amostra.

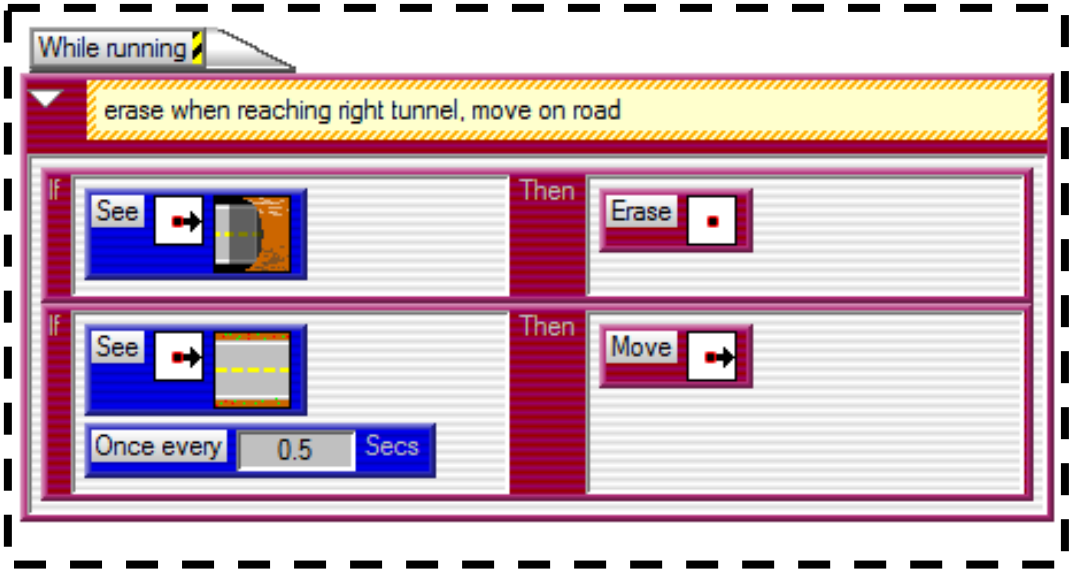

Figura 11 - Exemplo de uso da Trigger *WHILE-RUNNING*

A trigger ON simula o uso de funções dentro do ambiente AgentSheets; ela aprece em 27 dos 36 jogos analisados. Podemos ver então que o conceito de chamada de funções está presente no AgentSheets. Veja na Figura 12 um exemplo de uso da *trigger* retirado da amostra. À esquerda, um exemplo de chamada de função através do comando "*make*", à direita um exemplo de definição da função através da *trigger ON*.

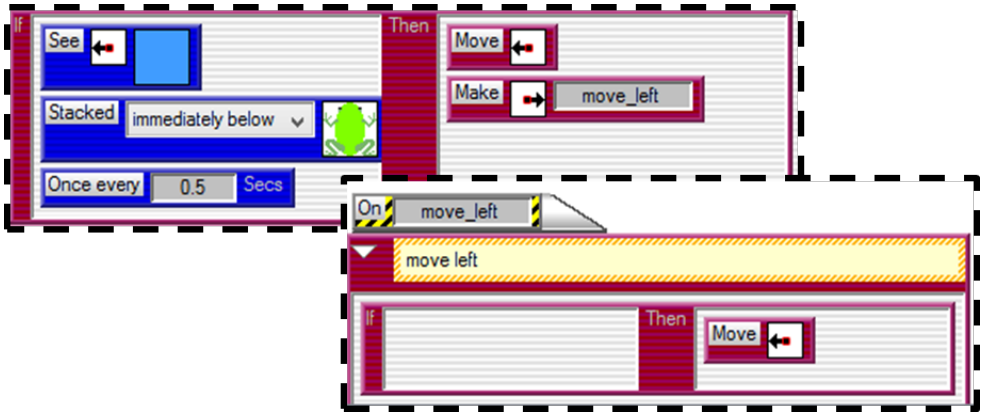

Figura 12 - Exemplo de uso da *trigger ON* e do comando *make*

As outras triggers não aparecem em nenhum projeto analisado. Fica o questionamento de por que isso acontece. Um fato interessante é que a *trigger WHEN-CREATING-NEW-AGENT* funciona como um "construtor" do Agente, ou seja, sempre que um novo "objeto" do tipo do agente é criado, ele roda as regras que estão dentro dessa *trigger*. Daqui podemos tirar um claro exemplo de um conceito embutido na mensagem do AgentSheets que pode não estar sendo percebido pelo aprendiz. Veja um exemplo criado para ilustrar uma possibilidade de uso dessa *trigger* na Figura 13 abaixo.

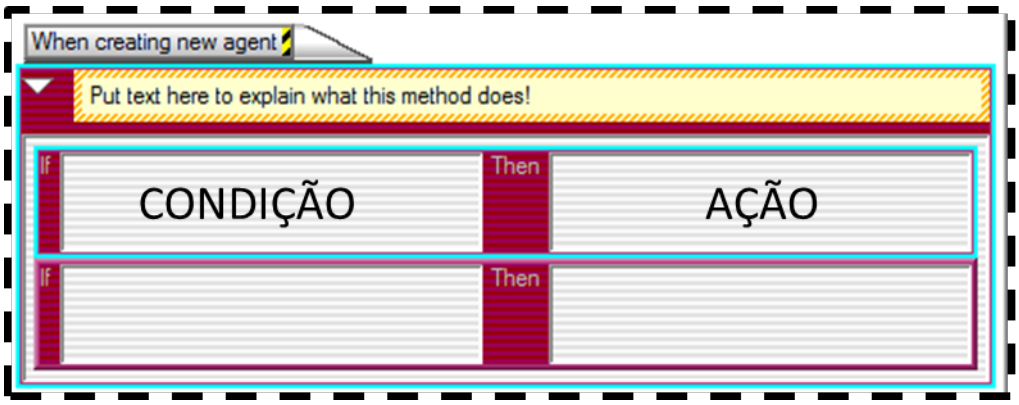

Figura 13 - Exemplo de uso da trigger *WHEN-CREATING-NEW-AGENT*

Como resultado desse estudo, percebemos que vários conceitos teóricos de computação podem ser explorados pela ferramenta *AgentSheets*. Dentre eles, podemos destacar alguns:

Estruturas de Controle (laços e seleções): A própria estrutura de regras "condição/ação" evidencia o conceito de seleção, enquanto a *trigger* "*WHILE-RUNNING*" evidencia os laços, mesmo que implicitamente.

Chamada de função: A *trigger ON* é claramente a forma como o AgentSheets passa o conceito de chamada de função. Porém, é uma chamada de função mais simples, onde o usuário não pode optar pela passagem de parâmetros.

Construtores: A *trigger WHEN-CREATE-NEW-AGENT* é equivalente a um construtor como em linguagens orientadas a objetos. Apesar de não ter ocorrência em nossa base de dados empírica, o conceito está sendo passado pelo AgentSheets, porém aparentemente não recebido pelos aprendizes.

Operadores Lógicos: Conjunção implícita quando combinados os comandos de condição. Apesar dos outros operadores estarem ausentes, o aprendiz já tem uma clara ideia do que é uma operação lógica.

Uso de variáveis: A presença e o grande uso dos comandos "*SET*" e "*IS*" evidenciam que o conceito de variáveis está sendo passado pelo *AgentSheets*.

Este estudo nos mostrou que os alunos da turma examinada utilizavam estruturas de programação que são usadas em LPs profissionais, mesmo que em seu aprendizado este fato não tenha sido (necessariamente) explorado ou sequer mencionado pelo professor. Também ficou claro que mesmo não tendo sido explorados nos projetos dos alunos, a LP visual do AgentSheets tem outras estruturas e elementos em comum com LPs textuais usadas por programadores profissionais ou leigos. Com estes dois resultados em mãos conseguimos então uma base empírica sólida para propor um modelo de ferramenta para a transferência de conceitos de programação entre uma LP visual e uma LP textual.

4.3
Um modelo de ferramenta de apoio à transferência de conhecimentos de programação

Baseado nos resultados apresentados nos tópicos anteriores, será proposto nesse tópico o modelo de transferência do raciocínio computacional entre duas linguagens de programação (LP). Por definição o modelo é unidirecional, onde a origem é uma linguagem visual e o destino uma linguagem textual. O modelo foi assim definido, pois com base nos trabalhos de Kölling (2010), Denny (2011) e Dann (2012), chega-se à conclusão que é mais eficiente o aluno, principalmente no ensino fundamental, aprender primeiro uma linguagem visual. A passagem de

uma LP visual para uma LP textual adiciona mais um desafio ao aprendizado: a sintaxe da linguagem.

Para Kölling (2010), aprendizes com menos de 14 anos ainda não têm maturidade suficiente para lidar com a sintaxe em uma LP. Assim, eles devem começar por um ambiente de programação que elimine o problema sintático, deixando que o aprendiz foque apenas na resolução do problema proposto.

Além disso, uma vantagem de apresentar o aluno a uma linguagem visual baseada na manipulação direta de objetos da interface faz com que ele foque sua atenção no aprendizado da construção do programa, eliminando as distrações e frustrações que os eventuais erros sintáticos podem causar nessa etapa do processo (Dann, Cosgrove, Slater, Culyba, & Cooper, 2012).

Também por definição, temos que o modelo é aplicável a um conjunto limitado de projetos, ou seja, não se aplica a qualquer projeto que tenha sido desenvolvido em uma LP visual. Tal definição é importante pois durante as pesquisas realizadas não chegamos a nenhuma conclusão que nos permitisse afirmar que qualquer projeto pode ser traduzido entre duas LP's arbitrárias. Então, para a definição do modelo, tomaremos um conjunto arbitrário, porém limitado de projetos que sabemos serem possíveis de implementar em ambas as LP's desejadas. Esse conjunto de projetos a serem escolhidos não é definido pelo modelo, pois envolve outras questões que não tecnológicas no processo de transferência do raciocínio computacional.

Como base teórica para o desenvolvimento do modelo, foi usada a Engenharia Semiótica (de Souza C. S., 2005b), uma teoria semiótica de Interação Humano Computador. Usaremos em especial um princípio definido como contínuo semiótico (de Souza, Barbosa, & Silva, 2001). O contínuo semiótico procura avaliar o quanto dois códigos artificiais apresentam obstáculos à antecipação de como deve ser a tradução de um para o outro. Nesse caso o modelo tem a proposta de fazer a (re)construção do contínuo semiótico entre duas LP's, eliminando os obstáculos existentes para que o aprendiz consiga fazer a tradução da LP de origem para a de destino.

A engenharia semiótica também serviu de inspiração na construção da mensagem que o modelo deverá passar ao aprendiz. Ao interagir com um sistema baseado no modelo, o usuário irá se deparar com três diferentes tipos de signos: os estáticos, os dinâmicos e os metalinguísticos (de Souza e Leitão, 2009). Os signos

estáticos têm o seu significado interpretado independentemente do tempo e de relações causais. Os signos dinâmicos surgem da interação e devem ser interpretados em relação ao desdobramento no tempo das atividades do usuário e da interface. Os signos metalinguísticos fazem referência aos signos estáticos, dinâmicos, e até mesmo metalinguísticos, tipicamente com o objetivo de explicar os outros signos. Os signos metalinguísticos podem ser, eles mesmos, estáticos ou dinâmicos.

O modelo aqui defendido foi desenvolvido visando sua incorporação ao modelo PoliFacets (Mota, 2014). Em sua tese, Mota (2014) define o PoliFacets como um modelo de documentação ativa baseado na criação de facetas. Em especial, a implementação desse modelo, o PoliFacets-AS, é uma instância que tem como objetivo desconstruir e documentar projetos feitos pelo *AgentSheets*. A desconstrução dos projetos é um processo onde o modelo PoliFacets desmonta o projeto e o remonta de diferentes maneiras criando facetas para exploração dos mesmos. Através dessa desconstrução, o aprendiz pode ver de forma separada diferentes elementos do seu projeto.

Em especial, uma categoria de facetas proposta por Mota será amplamente utilizada no modelo de transferência: as facetas que fazem operações de paráfrase e reformulação. A paráfrase é uma operação semiótica que atua no ponto de vista do usuário. Ela traz uma maneira diferente de representar o mesmo conteúdo. Dentro dessa categoria, usaremos em específico as facetas que fazem a reformulação, ou seja, mudam a representação do mesmo conteúdo sem modificar a semântica. Em sua tese, Mota implementou como caso de estudo uma faceta dessa categoria: a faceta Regras. Assim, quando exemplificarmos possíveis implementações para o modelo aqui proposto, também utilizaremos da faceta Regras já implementada. Ela traz uma maneira diferente de representar o mesmo conteúdo, porém em uma linguagem mais próxima da linguagem natural. A esse processo, damos o nome de *verbalização*, onde o modelo transforma o conteúdo programado em certa LP em uma linguagem natural, que se espera ser mais fácil de compreender para o aprendiz.

Mota defendeu em sua tese que o modelo PoliFacets poderia ser estendido de forma a promover a transferência de conhecimento entre duas LP's. A essa extensão, Mota deu o nome de "facetas transversais" que seriam responsáveis por levar o aprendiz de uma LP de origem a outra LP de destino. No entanto, Mota

não aprofundou nesse conceito, deixando o caminho aberto para uma nova pesquisa que preenchesse essa parte do modelo PoliFacets.

As facetas transversais também vão contar com a desconstrução do projeto feito na LP de origem. Por isso, o modelo de transferência defendido nesse texto terá então como ponto de partida o modelo PoliFacets. Ou seja, presumimos a existência de uma desconstrução instanciada de um programa feito com uma linguagem visual de programação que o usuário supostamente conhece. Consideramos, portanto, que o usuário já está familiarizado com os termos e as definições desse modelo.

As facetas transversais têm o objetivo de aproveitar a desconstrução do projeto feita pelo PoliFacets, fazendo a reconstrução do mesmo em uma nova LP. Esse processo está dividido em três facetas transversais, são elas: “Sintaxe”, “Conceitos” e “Reconstrução”.

As formas como essas facetas transversais se relacionam está representada na Figura 14 abaixo.

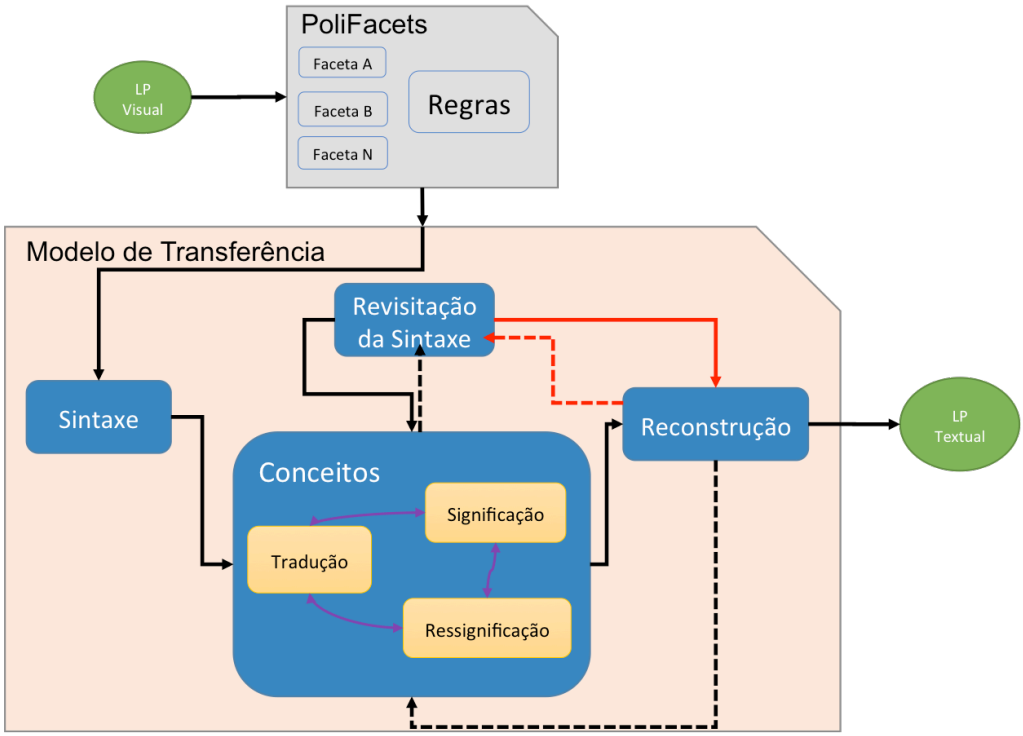

Figura 14 - Visão Geral do Modelo de Transferência

Como podemos ver na Figura 14, existe uma ordem para visitação das facetas transversais. As setas pretas contínuas indicam uma ordem natural que deve ser respeitada na interação de um sistema baseado no modelo. As setas

pretas pontilhadas indicam a possibilidade de voltar para a faceta anterior. As setas vermelhas pontilhadas indicam a possibilidade de voltar para qualquer outra faceta já visitada. Esse esquema de interação sugere uma sequência de visitação e ao mesmo tempo deixa o aprendiz livre para revisitar as facetas a qualquer momento.

A partir das definições das setas, algumas restrições podem ser notadas na interação. As três facetas têm que ter sua visitação obrigatoriamente na sequência sugerida. No momento estamos apenas definindo a estrutura do modelo. A motivação para essa definição ficará mais clara nos itens a seguir. A Figura 14 mostra também que o aprendiz deve passar antes pelo modelo PoliFacets, e obrigatoriamente ele deve conhecer a faceta "regras" que será muito utilizada na faceta transversal "conceitos".

A faceta "Conceitos" é subdividida em três subfacetas. Subfacetas, nesse modelo, são formas diferentes de visualizar a mesma faceta. Assim, poderemos explorar essa faceta de "Conceitos" de três diferentes formas: Tradução, Significação e Ressignificação. Essas formas serão explicadas detalhadamente no tópico sobre a faceta "Conceitos".

A seguir, teremos uma explicação detalhada de cada faceta transversal. Na descrição será explicado o motivo dessa sequência ser obrigatória. Começaremos pela faceta que serve como porta de entrada para o modelo: "Sintaxe".

4.3.1 Sintaxe

Em um ambiente de programação visual, o aprendiz em geral não precisa se preocupar com a possibilidade de cometer erros sintáticos. Com isso, ao ter contato somente com LP's desse tipo, ele não toma conhecimento da importância que tem a sintaxe em LP's de uso geral, como as usadas profissionalmente. É correto então afirmar que se precisamos ensinar aos aprendizes pensando que entre eles poderá haver futuros profissionais de informática e desenvolvedores de sistemas, eles devem, sim, ter contato com elementos sintáticos.

Para Denny (2011), se um aluno não é capaz de produzir um código que compila ao responder um determinado exercício, isto indica que ele está lutando com a sintaxe a tal ponto que não consegue receber nenhum feedback sobre a

lógica do seu código. Ou seja, a sintaxe passa a ser uma dificuldade primária no processo de aprendizado.

Com base nisso, nosso modelo tem como porta de entrada a introdução da sintaxe de uma linguagem de programação. Essa introdução será a partir dos conhecimentos já adquiridos pelo aprendiz em um primeiro ambiente que deve obrigatoriamente ser visual e isento de sintaxe.

Toda LP visual está obrigatoriamente abstraindo alguma linguagem ou *framework* para o usuário final. Tome por exemplo a LP visual *AgentSheets*. Ao programar usando o ambiente *AgentSheets*, o aprendiz não sabe, mas por baixo da camada abstrata está sendo gerado um código na linguagem de programação *Java* com o *framework* específico chamado *uAgentSheets*. Assim, podemos concluir que há uma relação de equivalência entre o que foi programado visualmente e o que foi gerado automaticamente em código *java*.

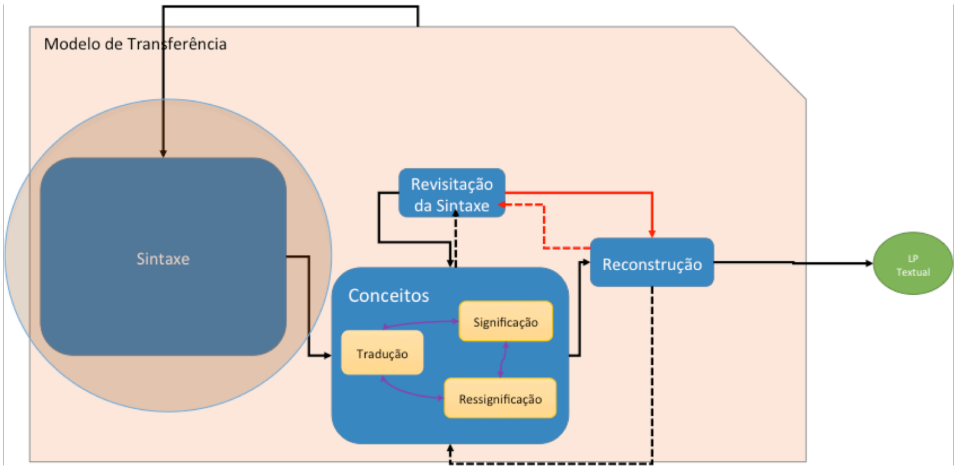

Figura 15 - Modelo de Transferência - Destaque para a faceta Sintaxe

Nessa faceta transversal, o modelo propõe a reconstrução do contínuo semiótico entre o que o usuário vê em sua interface de programação visual e um possível código em linguagem textual que represente a mesma programação. Para isso, devemos utilizar um componente comum a ambas as linguagens (visual e textual): a execução do projeto.

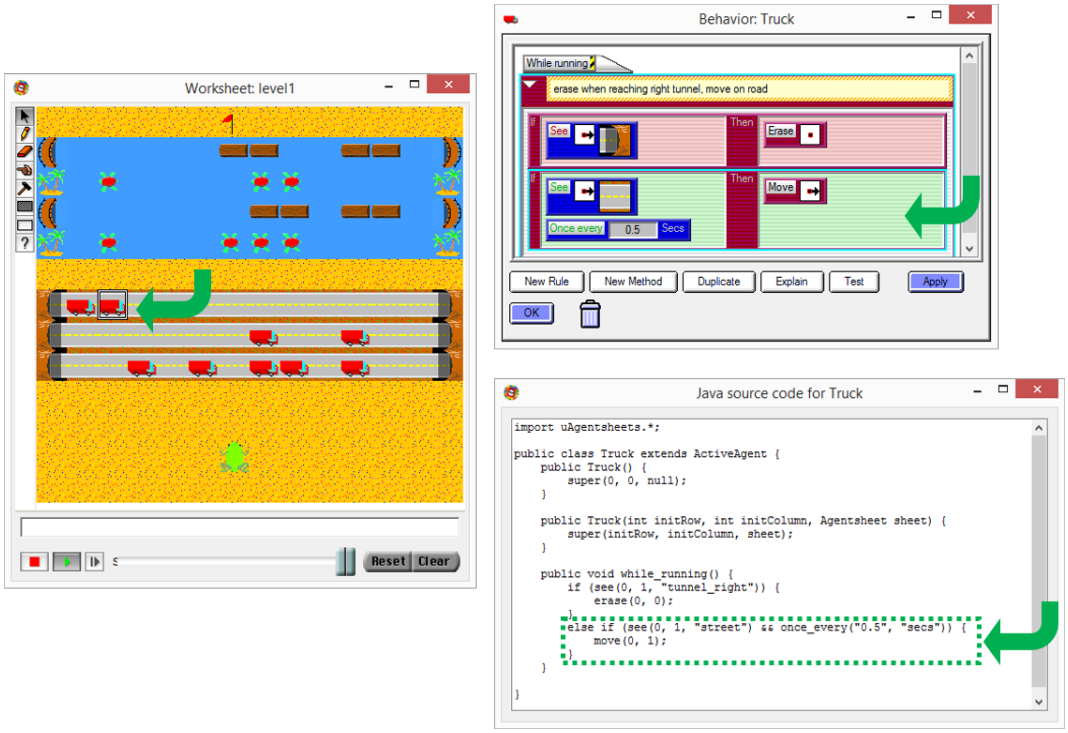

Figura 16 - 3 elementos (Execução, LP visual, LP Textual)

Veja na Figura 16 acima como se relacionam esses três elementos para um exemplo feito com o AgentSheets: à esquerda a execução do projeto; à direita em cima a LP visual oferecida pelo ambiente *AgentSheets;* e à direita abaixo a LP textual gerada com o *uAgentSheets*. Marcado com a seta verde, podemos notar as três diferentes formas de representar o mesmo objeto.

A LP visual representada por um signo estático e a LP textual representada por um signo metalinguístico ficam facilmente representadas na imagem. Porém, para representarmos a execução precisaríamos de signos dinâmicos. Como não é possível representar esses signos em uma única imagem, basta imaginarmos que no momento seguinte na execução, o caminhão estará deslocado a sua direita no espaço demarcado pela seta verde.

O modelo faz a reconstrução do contínuo semiótico entre representação textual e visual. Para que essa reconstrução aconteça, o sistema baseado no modelo de transferência deverá dar a possibilidade de o usuário interagir pedindo explicações sobre um trecho ou uma linha do código.

Ao pedir essa explicação, ela poderá vir de duas formas diferentes. A primeira forma ocorrerá caso o trecho de código tenha um equivalente direto com o que foi programado na LP visual. Como vimos na Figura 16 o trecho de código

que está demarcado pelo pontilhado verde tem um trecho equivalente bem definido na linguagem visual do *AgentSheets*. A segunda forma ocorrerá caso não exista um código equivalente na LP visual. Tal situação pode ocorrer, por exemplo, em trechos de código de importação de bibliotecas, tal como demarcado pela linha pontilhada azul na Figura 16.

Veja na Figura 17 um *mockup* de uma possível implementação dessa interação para o caso *AgentSheets.* No primeiro trecho hachurado temos o caso em que o código não tem um equivalente em *AgentSheets*, enquanto no segundo trecho hachurado o sistema permite que o aprendiz visualize o código *AgentSheets* equivalente.

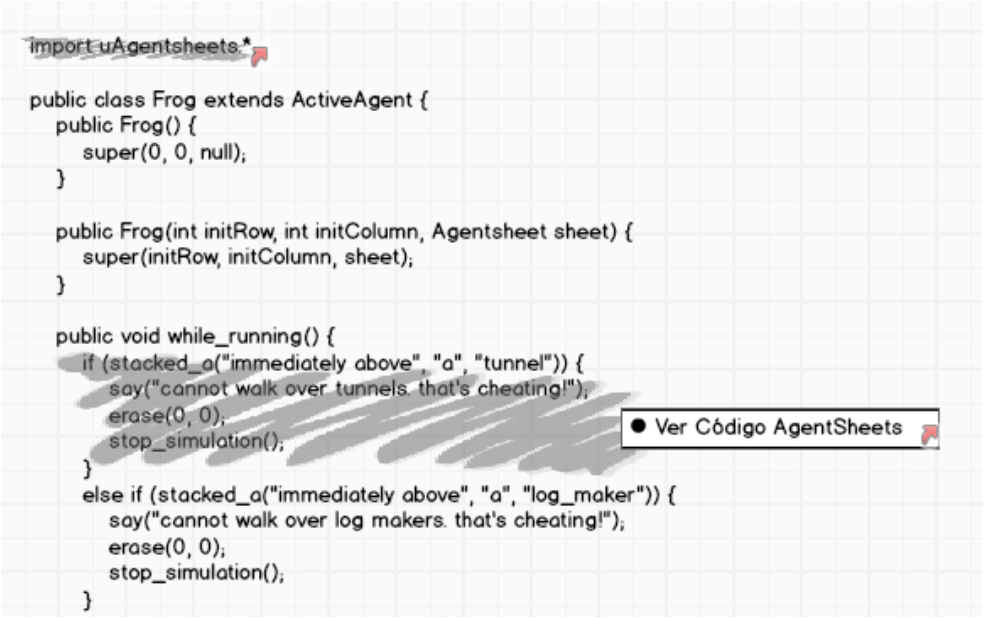

Figura 17 - Mockup da faceta Sintaxe

Caso a linha ou trecho de código selecionado tenha um equivalente na LP visual, então o sistema deve exibir esse trecho ao aprendiz utilizando signos estáticos. Para facilitar a recuperação do contínuo semiótico, o sistema deve exibir também em signos dinâmicos, uma fração da execução que aquele código representa.

A outra forma ocorrerá caso o trecho de código não tenha um equivalente direto com a LP visual. Como já foi dito, trechos de código como inclusão de bibliotecas, por exemplo, raramente têm seu equivalente na linguagem visual. Nessa situação, o modelo deve então oferecer uma explicação ao usuário através

de signos metalinguísticos, de forma que ele possa compreender a função sintática e semântica daquele trecho no código do projeto.

Usando essa faceta, o usuário passará a ter seu primeiro contato com a sintaxe em uma LP. Os problemas sintáticos não vão aparecer para atrapalhar e desmotivar, pois nesse ponto o usuário não está digitando nenhum código, ele estará apenas visualizando o código que foi criado a partir da desconstrução de um projeto permitindo que ele faça um paralelo com o que foi produzido na LP visual.

4.3.2 Conceitos

Após o usuário se familiarizar com o conceito de sintaxe, ele está apto para passar para a próxima faceta transversal. Esta se constitui como o núcleo do modelo e está voltada para o auxílio na transferência dos conceitos aprendidos na linguagem já conhecida. A faceta transversal *conceitos* vai usar como base a premissa de que o usuário conhece e está familiarizado com a faceta regras do modelo PoliFacets (Figura 18).

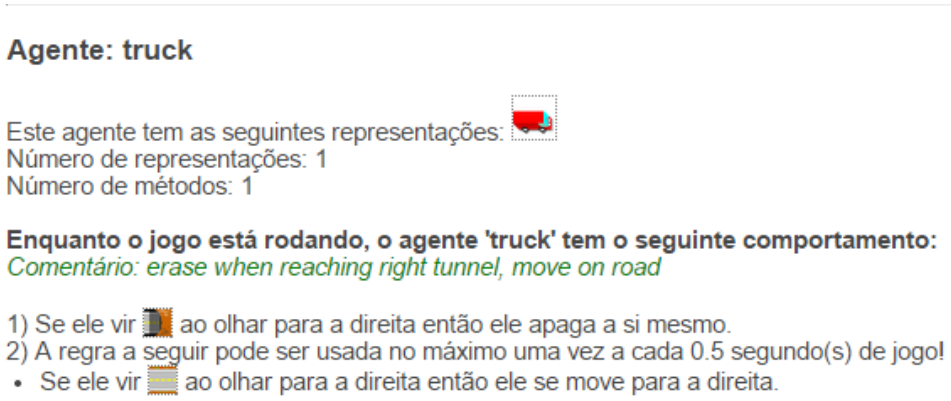

Figura 18 - Faceta Regras do PoliFacets-AS

Esta faceta é a responsável por transferir os conceitos aprendidos pelo usuário em uma LP visual para uma LP textual. Essa transferência pode se dar de três maneiras diferentes: Tradução, Significação e Ressignificação. Antes de entrarmos a fundo nessas subfacetas, vamos primeiro descrever o que são os conceitos e como eles aparecem (ou não) representados em uma linguagem de programação.

Um conceito é aqui definido como qualquer representação de elementos abstratos em um programa de computador. Por Exemplo: repetições, condicionais,

funções, classes e etc. Determinadas linguagens de programação podem, por opção de design, omitir ou expor determinados conceitos. Nessa faceta, o modelo fará a ligação entre um conceito (existente ou não) em uma LP Visual para um conceito (existente ou não) em uma LP Textual.

A representação de um conceito em uma LP pode se dar em quatro diferentes níveis: O nível mais completo é quando temos uma representação formal para o conceito, ou seja, quando temos uma estrutura sintática que representa aquele determinado conceito dentro de um programa de computador. Por exemplo, o conceito "condicional" é representado em muitas LP's pelo elemento sintático *IF / THEN*.

O segundo nível é quando temos o Conceito sem a presença da representação formal, ou seja, sem a presença de um elemento sintático específico que denote aquele conceito. Nesse nível porém, o usuário de uma linguagem consegue perceber de alguma forma (por exemplo, por inferência) a existência do conceito mesmo sem ver sua representação formal. Para exemplo desse caso, tome a linguagem Java. Toda passagem de parâmetro em uma função Java é feita por referência, porém, não existe a representação formal para indicar tal comportamento. Nesse caso, temos o conceito de "ponteiros" para indicar posição de memória sem a existência de sua representação formal presente em muitas linguagens, como C++ por exemplo.

O terceiro nível é quando temos um "efeito" de determinado conceito sem que o usuário perceba ou entenda o motivo de tal comportamento. Nesse caso, não temos nem o conceito, nem a representação formal explícita. Para exemplo, tome a linguagem AgentSheets. Ao invocar o comando "mover aleatório", a linguagem AgentSheets está por baixo executando o conceito de "números randômicos (ou aleatórios)". Tal conceito não é exibido em momento algum, porém o efeito do mesmo é visível para o usuário.

O último nível é quando não temos nem mesmo o efeito de um conceito na LP. Ou seja, por questões de design, a LP não trabalha com determinado elemento. Um exemplo simples desse nível é a ausência completa de "classes" em linguagens não orientadas a objetos. Tome por exemplo a LP 'C', nem mesmo o efeito do conceito "classe" aparece em momento algum da definição da linguagem.

Pensando agora nas duas LP's que são os casos de estudo do modelo aqui proposto, veja na Tabela 4 abaixo uma lista de conceitos e suas marcações nas existências de "efeito" (E), "conceito" (C) e "representação formal" (RF) para as linguagens AgentSheets e GreenFoot.

Tabela 4 - Efeito, Conceito e Representação Formal

	AgentSheets			GreenFoot		
	E	**C**	**RF**	**E**	**C**	**RF**
Classes	X	X*		X	X	X
Objetos	X	X*	X*	X	X	X
IF_THEN	X	X	X	X	X	X
ELSE	X			X	X	X
Método	X	X	X	X	X	X
Variáveis	X	X	X	X	X	X
Tipo de Dado	X			X	X	X
Loops	X			X	X	X
Concorrência	X*			X*		
Hierarquia				X	X	X
Encapsulamento				X	X	X
Números Randômicos	X			X	X	X
Eventos de teclado / mouse	X	X	X	X	X	X
Operações Lógicas	X			X	X	X
Recursão	X	X	X	X	X	X
Arquivos	X*	X*		X	X	X
Controle do Tempo	X	X	X			
Controle do Espaço de Jogo	X	X	X	X	X*	X*

Em cada linha da tabela temos um conceito. A presença de um 'X' significa a existência desse conceito em um dos níveis apresentados acima. Um 'X*'

significa que o conceito está presente naquele nível porém com algumas restrições. Para exemplificar, tome o conceito de “concorrência”. Em ambas LP’s, o usuário percebe a execução como sendo concorrente em seus diferentes Agentes / Atores, porém, na máquina a execução é sempre sequencial.

Agora que definimos “efeito”, “conceito” e “representação formal”, voltamos a descrever o funcionamento da faceta transversal conceitos. Essa faceta será dividida em 3 subfacetas: Tradução, significação e ressignificação. Ao acessar essa faceta, o aprendiz vai ter opção de destacar qual subfaceta ele deseja trabalhar no momento.

O aprendiz irá utilizar-se da faceta Regras proposta pelo modelo PoliFacets de maneira semelhante como proposto na faceta anterior. A interação porém vai ser diferente em cada subfaceta a ser descrita a seguir. Ao selecionar uma subfaceta, o modelo deve destacar quais pedaços da verbalização oferecida pela faceta Regras pertencem àquela subfaceta deixando o usuário livre para interagir com esses pedaços. Nos próximos tópicos discutiremos como será essa interação em cada subfaceta.

4.3.2.1 Tradução

No modelo definimos que a tradução de um conceito é possível entre duas diferentes linguagens de programação se e somente se em ambas as LP’s existirem representações formais bem definidas para esses conceitos. Quando um conceito está presente e bem representado em ambas linguagens, basta indicarmos ao aprendiz como codificar aquele conceito na nova linguagem de programação que ele está buscando aprender. Nesse caso não é preciso mais explicações pois o usuário já aprendeu e entendeu o conceito na LP original.

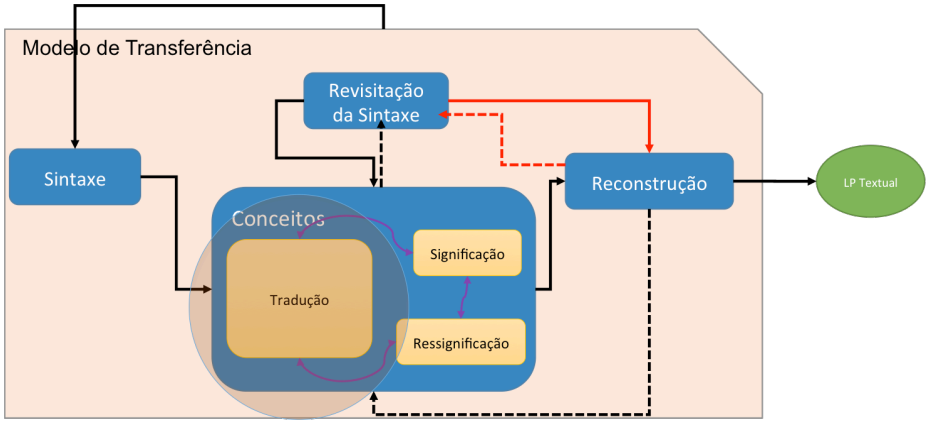

Figura 19 - Modelo de Transferência - Destaque para a subfaceta Tradução

Tome como exemplo no nosso caso de estudo o conceito de "métodos". Em AgentSheets existe uma representação formal através de sua linguagem visual de como definir e executar um método. Para que o aprendiz passe a usar esse conceito na nova LP GreenFoot, basta que o modelo forneça a ele uma tradução da codificação necessária para criar e executar um método. Veja que nesse caso o conceito é o mesmo e está presente em ambas LP's. A única diferença entre elas é em como é implementado a representação formal do conceito.

Dessa forma, na subfaceta Tradução, ao interagir com as partes destacadas da verbalização, o aprendiz poderá ter 3 opções: A primeira opção é a exibição do código visual da LP de origem para aquele pedaço da verbalização. Essa exibição deve acontecer usando signos estáticos, mostrando o código da LP de origem.

A segunda opção é a exibição, através de signos metalinguísticos, do código textual da nova LP que o aprendiz está trabalhando. Por último, usando de signos dinâmicos, o modelo deve exibir a execução daquele pequeno pedaço de programa. Essa execução, que é a única parte em comum entre as diferentes LP's funcionará como uma ponte, recuperando o contínuo semiótico entre o que está verbalizado (e programado nas duas LP's) e o que está sendo executado na interface final do programa.

4.3.2.2 Significação

Uma significação é definida no modelo quando um determinado conceito está presente na LP de destino, porém está ausente ou apenas representado pelo

efeito na LP de origem. Nesses casos o aprendiz nunca teve contato direto com o conceito que ele está para aprender. O modelo então terá que explicar ao aprendiz o que significa aquele conceito. Aqui, uma simples tradução como na faceta anterior não seria possível pois o aprendiz nunca se deparou com uma representação formal deste conceito.

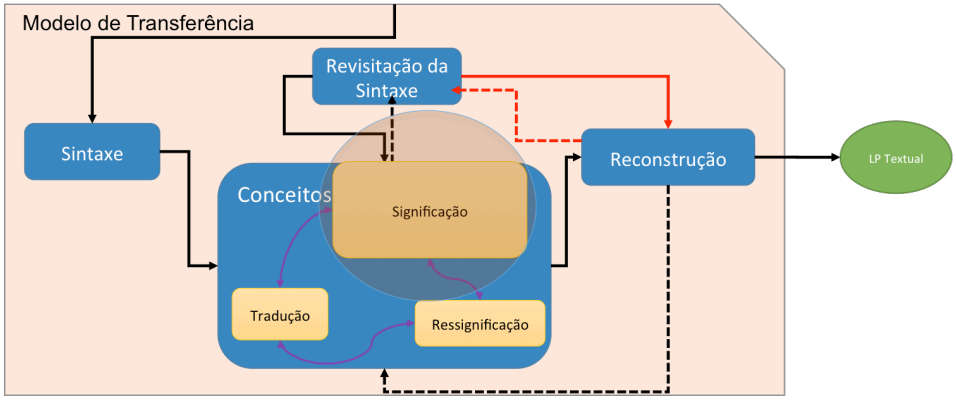

Figura 20 – Modelo de Transferência - Destaque para a subfaceta Significação

Tome como exemplo no nosso caso de estudo o conceito de "operações lógicas". Em AgentSheets existe o efeito causado pelas operações lógicas, porém, o conceito não aparece explícito para o usuário em momento algum. Ao juntar duas condições em uma só regra no AgentSheets, automaticamente o sistema faz uma operação de conjunção lógica entre as duas condições. O efeito é claro para o aprendiz, porém o conceito não é exposto para ele em momento algum. Já na LP GreenFoot, o aprendiz terá que lidar diretamente com operações lógicas caso ele deseje combinar duas condições. Para isso o modelo deve antes de exibir a representação formal, explicar ao aprendiz o que é esse novo conceito.

Nessa subfaceta teremos então uma nova opção de interação além das três já existentes na faceta anterior. Esta é a opção de exibir uma explicação ao aprendiz daquele conceito usando signos metalinguísticos. Essa explicação deve vir de maneira que aproveite o conhecimento prévio do aprendiz e não cabe ao modelo definir como deve ser, deixando que a implementação do modelo decida pela melhor didática em passar o conteúdo.

4.3.2.3
Ressignificação

Uma ressignificação é definida no modelo quando um determinado conceito está presente na LP de origem, porém está ausente ou apenas representado pelo efeito na LP de destino. O modelo terá então que ressignificar o conceito da LP de origem em um novo conceito na LP de destino. Nesse caso em particular, vale lembrar que essa ressignificação tem sempre que ser possível no conjunto de jogos arbitrários que definem a instância do modelo. Caso não seja possível uma ressignificação do conceito, então, o projeto não é mutuamente traduzível entre as duas LP's com a qual deseja-se trabalhar, fazendo assim com que o ele não possa passar pelo modelo aqui proposto.

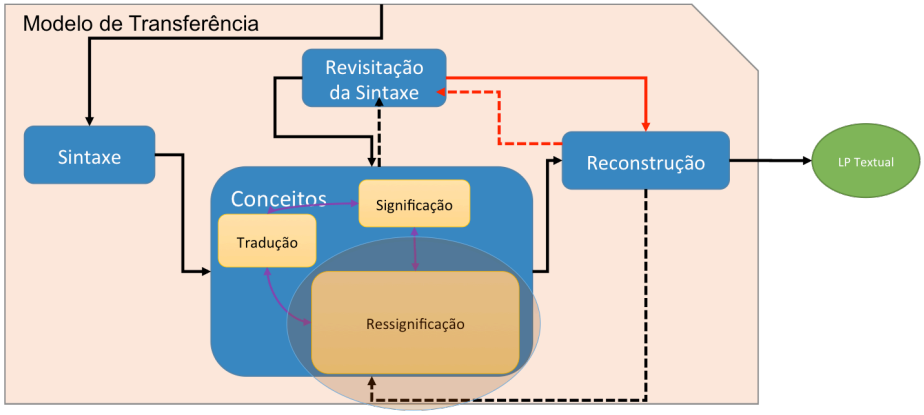

Figura 21 – Modelo de Transferência - Destaque para a subfaceta Ressignificação

Tome como exemplo de ressignificação no nosso caso de estudo o conceito de "controle do tempo". Esse conceito está amplamente difundido na LP AgentSheets, através de vários comandos que fazem a manipulação do tempo, como por exemplo o comando "uma vez a cada x segundos". Tal manipulação do tempo não existe na LP GreenFoot, ou, seja, é necessário que o modelo ressignifique esse conceito em outro(s) conceito(s) que existem no GreenFoot.

Nessa subfaceta, teremos então somado mais uma possibilidade, além das três anteriores da faceta tradução. A possibilidade de exibir ao aprendiz qual ou quais conceitos ele deveria aplicar para conseguir o mesmo efeito na nova LP. Essa possibilidade é diferente da exibição do código final na nova LP, pois aqui, o modelo apenas apresenta quais conceitos o aprendiz deve explorar, deixando a critério dele criar a sua representação formal para esse comportamento.

4.3.3
Reconstrução

Para finalizar, o modelo é capaz de apoiar o aprendiz na reconstrução do projeto na linguagem textual desejada. Esta última faceta transversal explora dois conceitos já consolidados na engenharia de software: reuso de código e modularização.

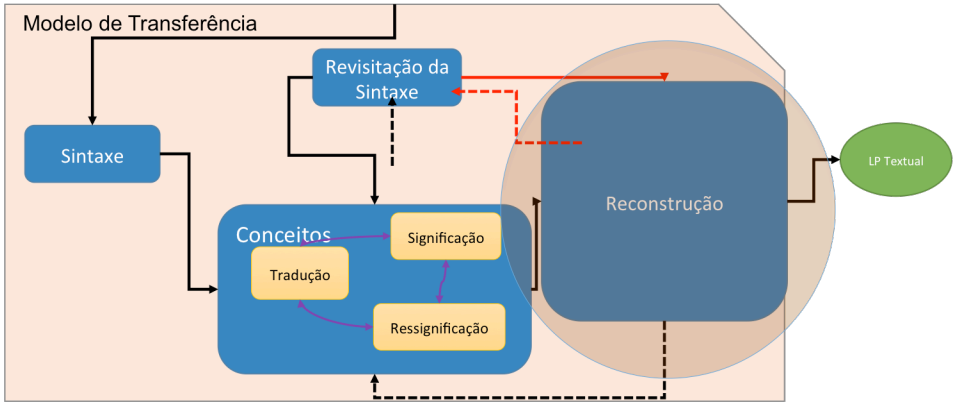

Figura 22 - Modelo de Transferência - Destaque para a faceta Reconstrução

O sistema baseado no modelo de transferência deve oferecer ao aprendiz a possibilidade de fazer o download de um projeto base. Este projeto, que vai funcionar como um template, faz com que o aprendiz passe a ter contato com o reuso de código, pois ele já teria um código base de onde começar seu novo projeto. O template deve vir com todos arquivos necessários para criação do projeto na LP textual. Dentro desses arquivos devem vir as instruções mínimas como chamadas de bibliotecas, assinaturas de funções e relacionamentos com outros arquivos do projeto.

Um projeto base instanciado para o GreenFoot por exemplo, deve conter arquivos como o mostrado na Figura 23. Veja que o arquivo tem apenas as inclusões de bibliotecas, assinaturas da classe e das funções. Assim, o aluno se preocupa apenas em preencher a lógica interna de cada procedimento já assinado.

```
import greenfoot.*;

public class Frog  extends Actor
{
    public Frog (){
    }

    public void act(){
    }

    public void move_right(){
    }

    public void move_left(){
    }
}
```

Figura 23 - Exemplo de arquivo template para um projeto GreenFoot

Nessa faceta transversal, o aprendiz pode desejar voltar a faceta anterior de conceitos para consulta. Com a ajuda da faceta "conceitos" e o template fornecido pela faceta "reconstrução", o aprendiz é capaz de reconstruir todo seu projeto na nova LP textual.

Uma outra função nessa faceta transversal é o sistema oferecer ao aprendiz a possibilidade de fazer o download de cada arquivo separado do seu projeto, porém já com a programação realizada, ou seja, com o código preenchido. Com essa funcionalidade, o aprendiz passa a ter contato com a noção de modularização. Ao ter acesso a arquivos isolados do projeto o aprendiz poderá importar esses novos arquivos no seu projeto, como se fossem módulo independentes. Cabe aqui ao aprendiz, saber fazer a integração desse novo arquivo com o resto do projeto, desafio muito comum em sistemas modularizados.

Um exemplo de arquivo para a instância GreenFoot segue na Figura 24 abaixo. Nesse caso, além da assinatura da classe e das funções, o arquivo vem também com toda lógica implementada. Essa funcionalidade completa o modelo como a etapa final de uma caminhada que partiu da visualização do projeto na LP visual, terminando com a visualização do projeto em uma LP textual.

```
import greenfoot.*;

public class truck  extends Actor
{
    public void act()
    {
        if(atWorldEdge()){
            getWorld().removeObject(this);
        }
        else if(getOneObjectAtOffset(0, 0, Frog.class); != null) {
            getWorld().removeObject(actor);
             Greenfoot.playSound("crash2.wav");
        }
        else if(){
            setLocation(x, y + 32);
        }
    }
}
```

Figura 24 - Exemplo de arquivo GreenFoot preenchido

Nesse capítulo foi apresentado o modelo de transferência de raciocínio computacional defendido nessa dissertação de mestrado. No capítulo seguinte iremos descrever uma avaliação preliminar feita sobre o modelo. Em seguida faremos a conclusão e posicionaremos o modelo e suas contribuições para a comunidade científica.

5
Avaliação Preliminar do Modelo

Nosso trabalho se posiciona como um modelo para tecnologia de apoio a transferência de raciocínio computacional. Acreditamos que professores e alunos podem tirar proveito de uma implementação deste modelo durante o processo de aprendizado de uma nova linguagem de programação textual. Como última etapa da nossa estratégia de pesquisa, realizamos um estudo com o objetivo de realizar uma prova de conceito do modelo proposto. A seguir uma descrição do estudo e dos resultados encontrados.

5.1
Perfil do Estudo com Usuários

Com o objetivo de realizar uma prova de conceito do modelo proposto foi realizado um estudo qualitativo com a participação de três usuários com diferentes perfis. O estudo foi conduzido individualmente com cada usuário que teve que consentir a sua participação assinando o termo de consentimento em pesquisa que está no Apêndice 2 desse texto. Durante o estudo percebemos como usuários navegam pelo modelo de transferência proposto, colhendo informações sobre a interação e suas opiniões sobre a proposta. Através desse estudo poderemos ver quais partes do modelo têm alguma deficiência para que ele possa ser melhorado em trabalhos futuros.

Os usuários que participaram do estudo eram todos parceiros de pesquisa e instrutores do projeto SGD-Br, ou seja, com ampla formação em AgentSheets e grande conhecimento do PoliFacets. Todos participantes tinham curso superior e dois deles eram pós graduados. Vale ressaltar aqui que esse estudo não serve como base para a avaliação do modelo voltada para usuários reais. Aqui, todos eram experientes usuários que participaram de forma colaborativa durante a pesquisa. Infelizmente, dentro do tempo de construção dessa dissertação, não foi possível a realização de estudo com usuários reais. Tal estudo ficará como trabalho futuro de curto prazo.

No estudo, precisaríamos de um projeto no AgentSheets que estivesse dentro das definições propostas pelo modelo, ou seja, que fosse traduzível de AgentSheets para GreenFoot. Este foi criado antecipadamente pelo pesquisador para garantir essa característica. O projeto explora as três diferentes formas de se transferir um conceito aprendido: a tradução, significação e ressignificação. O projeto corresponde a uma simulação onde temos um agente "bolinha" (●) que se move para direita enquanto ele vir acima dele o agente "quadrado" (■) e abaixo o agente "triângulo" (▲). Veja na Figura 25 a posição inicial da simulação, na Figura 26 a representação do comportamento do agente "bolinha" no AgentSheets e na Figura 27 o comportamento do agente "bolinha" verbalizado na faceta regras o PoliFacets.

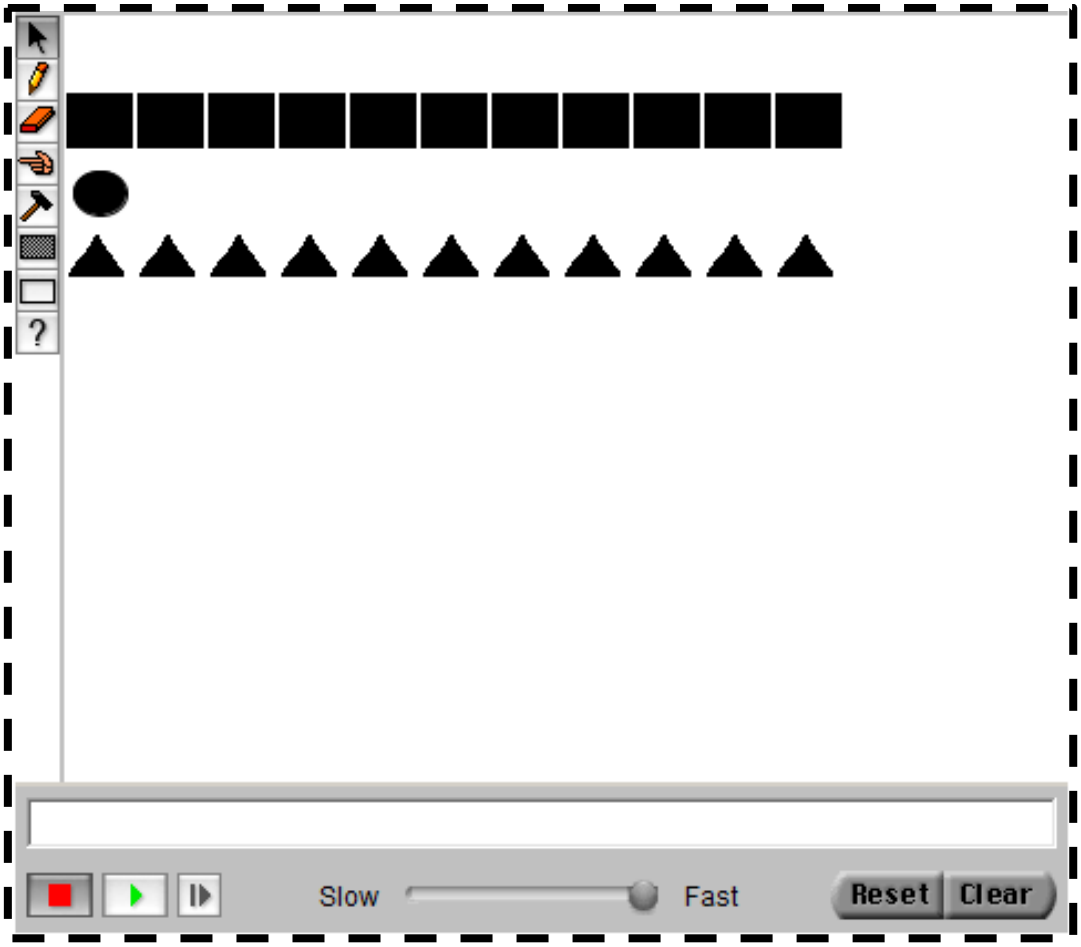

Figura 25 - Planilha do projeto usado para o estudo

Figura 26 - Comportamento do Agente "bolinha" representado no AS

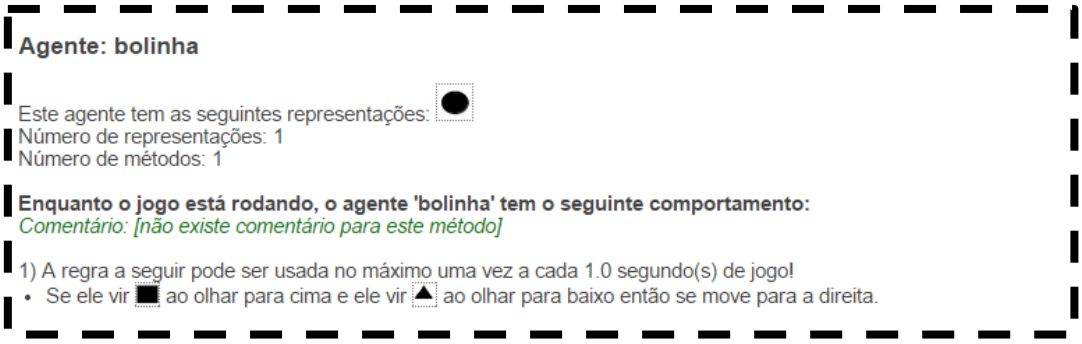

Figura 27 - Verbalização do Comportamento no PF

Ao entrar nesse projeto no PoliFacets o usuário tinha a opção de clicar no link "Go To GreenFoot" que o levava a uma instância das novas facetas de transição propostas pelo modelo. Tal instância deve levar o usuário do AgentSheets para o GreenFoot. Veja na Figura 28 abaixo como esse link se posiciona no sistema PoliFacets.

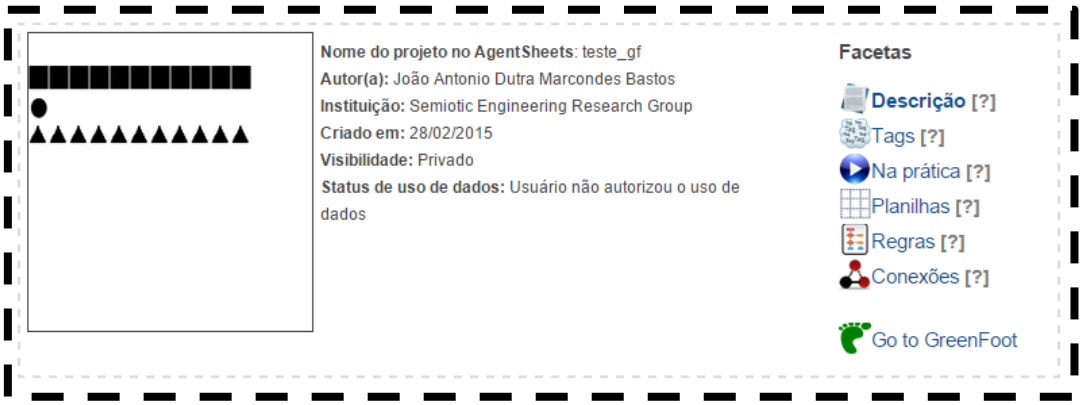

Figura 28 - Link "Go To GreenFoot"

A atividade, que teve duração média de 60 minutos, foi dividida em três partes. Na primeira parte o usuário ficou livre para interagir com o sistema PoliFacets que já conhecia explorando todas as facetas do jogo "*Corredor de Bolinha*".

Na segunda parte, o usuário migrou para o sistema de transferência do raciocínio computacional clicando no link "Go To GreenFoot" onde fizemos uma navegação guiada pelas facetas de transição que estão sendo propostas. Esta navegação guiada é uma estratégia de pesquisa onde o participante pode interagir com o sistema e ao mesmo tempo conversar com o pesquisador para tirar dúvidas e pedir instruções sobre como realizar a navegação no sistema. Essa estratégia foi adotada pois não era o objetivo do estudo avaliar a comunicabilidade do sistema, então, sempre que a mensagem não era passada somente pelos signos do sistema, o usuário podia pedir uma "explicação" para o pesquisador sobre o que não tinha ficado claro.

A navegação pelas facetas de transição foi realizada em um *mockup* de baixa fidelidade de uma instancia do modelo para o caso de AgentSheets para GreenFoot. Esses mockups podem ser vistos abaixo. Na Figura 29 temos o mockup dos links das facetas de transição. Na Figura 30 temos o mockup da faceta de transição "sintaxe". Na Figura 31 temos o mockup da faceta de transição "conceitos". E por último, na Figura 32 temos o mockup da faceta de transição "reconstrução".

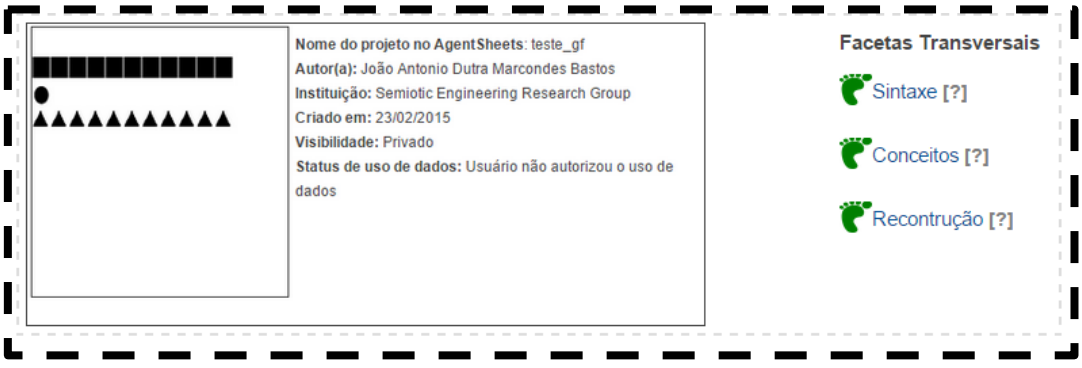

Figura 29 - Mockup dos Links das Facetas de Transição

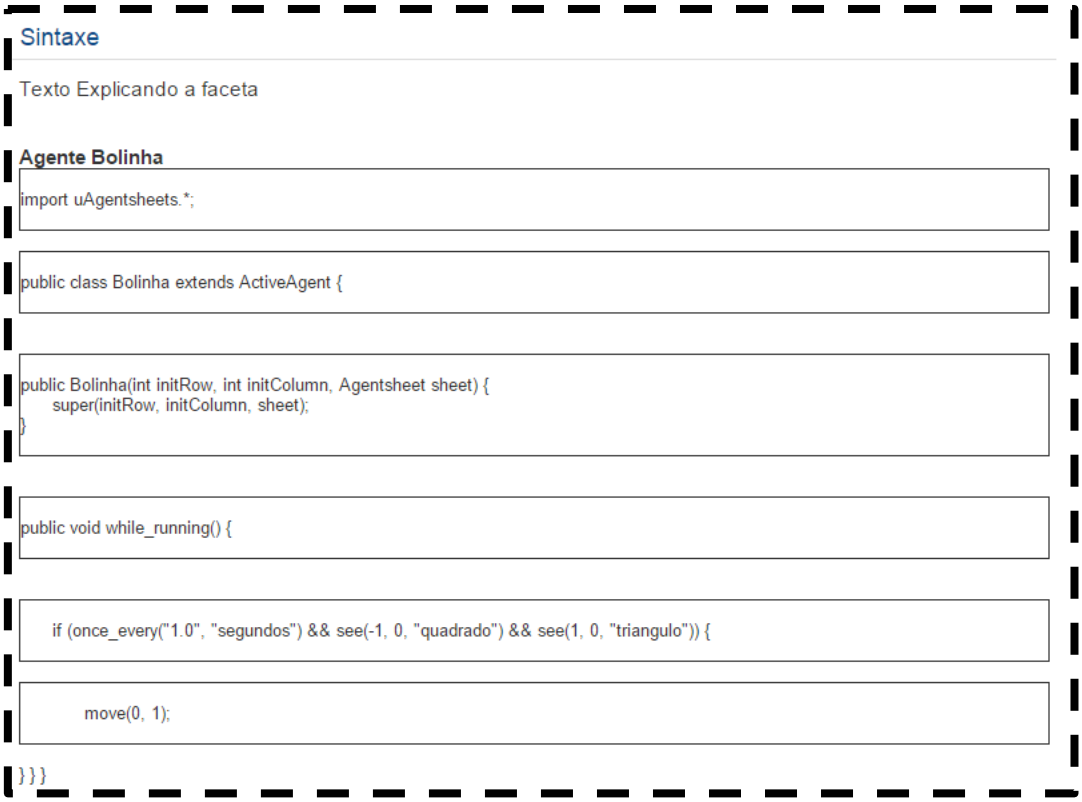

Figura 30 - Mockup da faceta "sintaxe"

Figura 31 - Mockup da faceta "Conceitos"

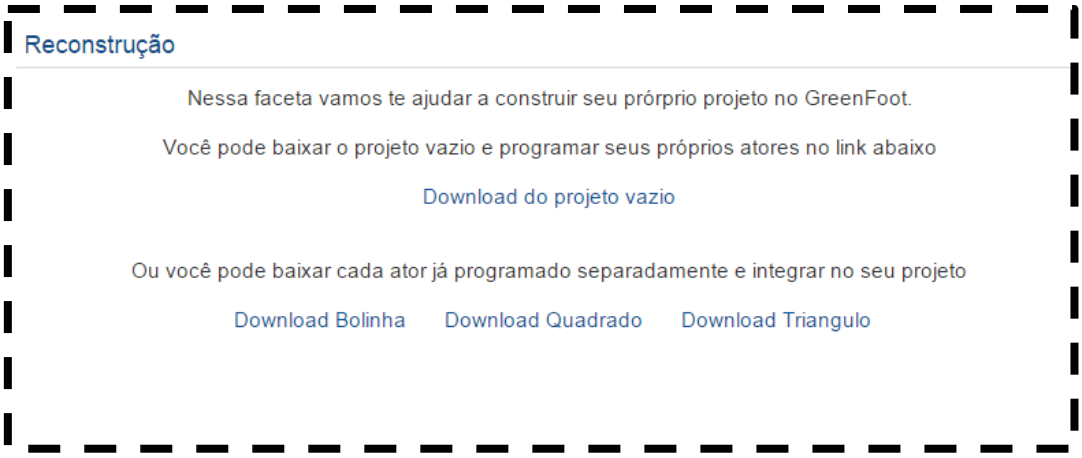

Figura 32 - Mockup da faceta "reconstrução"

Na terceira parte foi pedido que o usuário recriasse o projeto "*Corredor de Bolinhas*" na nova linguagem de programação chamada GreenFoot, tendo como fonte de pesquisa apenas o sistema apresentado no estudo. Ao final, os participantes passaram por uma entrevista final para coletar suas críticas a respeito do modelo.

Os três usuários participantes tinham em comum, de acordo com eles próprios, conhecimento alto na linguagem AgentSheets. Além disso, todos já conheciam a ferramenta PoliFacets-AS. Os usuários também tinham em comum a experiência profissional como professor, seja de programação ou não.

Passaremos agora a adotar nomes fictícios para os usuários para podermos identificá-los preservando seu anonimato. O usuário 1, que chamaremos de Ricardo, não é uma pessoa da área de computação. É um professor de matemática que nunca teve contato com nenhuma linguagem de programação textual. Seu conhecimento de programação se restringe ao que aprendeu em AgentSheets. O usuário 2, que chamaremos de Igor, é formado em curso da área de informática e tem conhecimento de várias linguagens de programação profissionais. O usuário 3, que chamaremos de Mário também é formado em curso da área de informática com amplo conhecimento em várias linguagens de programação.

Os três executaram o mesmo teste e todos conseguiram cumprir como seria de se esperar o objetivo final que era reconstruir o projeto "*Corredor de Bolinhas*" na linguagem de programação GreenFoot. A coleta dos dados foi realizada através de gravação da tela e do áudio do usuário nas conversas com o pesquisador. Esses registros foram posteriormente analisados e separados em categorias de análise de resultados que serão apresentadas no tópico a seguir.

5.2
Resultado da avaliação

Os resultados encontrados se referem a uma apreciação preliminar dos participantes sobre o quanto e como uma ferramenta que instancie o modelo proposto pode apoiar o processo de transferência de conhecimentos de programação do AgentSheets para o GreenFoot. Os usuários selecionados são experientes e têm a capacidade de realizar uma crítica de vários aspectos da complexidade da transferência de conceitos de programação de uma linguagem para outra em relação à proposta conceitual do modelo. Capturar essas críticas era o principal objetivo do estudo. Nesse tópico serão apresentadas essas críticas categorizadas.

Os participantes perceberam a familiaridade entre o modelo proposto e o PoliFacets. Em várias oportunidades temos relatos dos participantes dizendo que se sentiram confortáveis com o sistema pois este parecia muito com o sistema que eles já conheciam, o PoliFacets. Podemos destacar aqui uma fala do participante Ricardo em que ele diz: "*De ler a verbalização, no vicio já vem as regras do AS e associo automaticamente uma coisa com a outra*". Essa familiaridade era de fato um dos objetivos da proposta do modelo. Aproveitar ao máximo o conhecimento prévio dos usuários.

Essa associação com o PoliFacets foi percebida muito devido a faceta conceitos. Todos usuários fizeram a associação rápida desta com a faceta regras do PoliFacets. Uma evidência que podemos ressaltar é uma fala de Igor dizendo: "*Isso aqui (faceta conceitos) é o mesmo texto da faceta regras*". Essa evidência nos mostra que o usuário faz uma associação mais rápida entre o que tinha sido programado com o que ele estava aprendendo.

Ainda sobre a faceta conceitos, temos evidências que ela ajudou os participantes na transição entre as duas linguagens. Para Mário, esta foi a faceta que mais lhe ajudou no processo, perguntado o motivo ele respondeu: "*quando eu cheguei na faceta conceitos ficou mais fácil eu ver a transição pois ai eu consigo ver tanto o código AgentSheets quanto o código GreenFoot*".

Outra categoria que conseguimos destacar foi o papel das explicações parceladas e sua a aproximação a nova linguagem a medida que o usuário navegava pelas facetas na ordem proposta pelo modelo. Dois participantes

conseguiram perceber essa aproximação. Podemos destacar uma fala de Mário onde ele diz: "*A faceta 'reconstrução' já está mais perto do GreenFoot enquanto a faceta 'sintaxe' está mais perto do AgentSheets*". Essa aproximação é desejada pelo modelo e foi percebida pelos participantes.

Do estudo, emergiram também algumas críticas ao modelo. No nível mais abstrato, temos evidência que a faceta sintaxe não foi bem explorada. O participante Ricardo não conseguiu entender qual o propósito dela como fica evidenciado em sua fala: "*não entendi o que significa essa faceta, este já é o código GreenFoot que devo usar?*". Os outros participantes relataram que a presença da faceta sintaxe não ajudou muito no processo de transferência.

Essa crítica percebida nos três participantes com relação à faceta sintaxe nos leva à questão de se ela realmente é útil. Podemos nesse ponto fazer um confronto com o que foi encontrado na literatura. Seria a sintaxe de uma linguagem de programação realmente um grande problema no processo de aprendizado? Infelizmente não podemos tirar nenhuma conclusão a respeito em cima de nossos estudos, porém cabe aqui uma breve discussão sobre o assunto.

Como vimos em muitos estudos relacionados, a sintaxe se mostrou um grande problema no processo de aprendizado. Isso ficou claro nos estudos de Denny (2011), Kölling(2010) e Daly(2010) já citados anteriormente nos trabalhos relacionados. Porém, o que percebemos durante nossos estudos é que a sintaxe não foi a principal barreira de aprendizado de uma nova linguagem. Na maior parte dos casos, o problema que se mostrava era mais semântico do que sintático. De fato, uma sintaxe de uma linguagem de programação é apenas a forma que tal linguagem escolheu para expressar a semântica que está atrelada ao código.

Para considerarmos apenas em estudos futuros, levanto aqui uma reflexão de que a semântica de uma linguagem de programação talvez seja mais difícil de compreender do que a sintaxe da mesma. A sintaxe de fato não passa de um conjunto de regras a ser decorado pelo aprendiz, enquanto a semântica envolve vários outros conceitos computacionais e matemáticos.

Tal observação fica ainda mais evidente se levarmos em conta que com o apoio de uma linguagem de programação visual e de uma ferramenta que facilite o aprendiz a lembrar as regras sintáticas da linguagem, percebemos que a sintaxe deixa de ser de fato um problema e o aprendiz passa a ter grandes dificuldades na hora de formar as regras semânticas de seu programa.

Outra crítica que os participantes citaram foi a de que faltou adicionarmos ao modelo e ao sistema uma fase de explicações preliminares dos elementos da interface do GreenFoot. Para eles, apesar de que entenderam em parte como era a programação em GreenFoot, eles não conseguiram saber por onde começar ao abrir o sistema. O participante Igor, por exemplo, disse que se perdeu ao abrir o GreenFoot pela primeira vez. Para ele faltou *"por exemplo explicar conceitos da interface do greenfoot como ator, mundo e outros.*"

De posse desses resultados, apresentaremos agora no capítulo de conclusão uma breve discussão sobre o posicionamento do modelo no conhecimento científico e quais as possibilidades de estudos futuros abrimos com essa pesquisa. Muito dos resultados descritos aqui podem e devem ser reaproveitados em trabalhos futuros para aprimoramento e redesign do modelo de transferência proposto.

6
Conclusão

A pesquisa realizada se alinha com aspectos da pesquisa-ação. O seu principal resultado é uma proposta de modelo de tecnologia que por sua vez se insere em outro modelo de tecnologia já implementado e sendo utilizado no contexto a que se destina. Toda tecnologia carrega consigo uma proposta de intervenção no estado das coisas, com vistas a causar uma transformação benéfica para as pessoas que estão sofrendo a intervenção. Portanto, esta pesquisa se alinha ao estilo de pesquisa-ação cujo o ciclo é: (I) Análise de estado das coisas e oportunidade de intervenção; (II) elaboração de modelo de intervenção; (III) intervenção; (IV) avaliação da situação dos artefatos e do valor da intervenção. Durante a pesquisa, as duas primeiras etapas foram cumpridas com o levantamento da literatura e identificação de oportunidade; e com a proposta do modelo de transferência para preencher tal lacuna. As duas últimas etapas não foram completamente cumpridas, porém cuidamos de antecipar o que poderá ocorrer nelas através do estudo de prova de conceito que foi descrito no Capítulo anterior.

Dado o contraste entre o modelo proposto e avaliação preliminar realizada, concluímos que a pesquisa contribuiu em duas frentes principais. A primeira é que o modelo de transferência do raciocínio computacional proposto preenche uma lacuna na área de ensino do raciocínio computacional. Sabemos que muito ainda pode ser feito e estudado para aprimorarmos o modelo, mas este foi o ponto inicial para uma pesquisa que pode e deve ser estendida. Durante levantamento da literatura, como foi descrito nos trabalhos relacionados, muitos trabalhos tangenciaram a ideia de transferência de conhecimento entre dois ambientes de programação. Ficou claro porém que esse espaço estava em aberto na comunidade e não tinha sido explorado a fundo.

O modelo proposto tem o objetivo de preencher a lacuna tecnológica desse processo de transferência de raciocínio computacional. Não fizemos nenhum estudo, no entanto não podemos afirmar nada sobre métodos didáticos e pedagogias que envolvam esse processo. Não podemos dizer nesse trabalho quais

LP’s são mais eficientes no ensino de programação. Essa questão fica em aberto para uma pesquisa na área de educação, onde poderão ser abordados aspectos como a ordem que as LP’s e os conceitos devem ser explorados.

Por se posicionar em uma lacuna ainda não explorada no ensino / aprendizado de programação, concluímos que o modelo é algo inovador. Por isso mesmo deve ser ainda mais aprofundado em trabalhos futuros e posto a mais provas de conceito para que sua implementação possa chegar ao usuário final. E é nesse ponto que temos a segunda frente de contribuição da pesquisa. Com ela temos uma coleção de trabalhos futuros que foram indicados a partir do estudo de avaliação realizado, são eles:

Como trabalhos futuros a essa pesquisa temos três frentes distintas: Curto, Médio e Longo prazo. No curto prazo, um trabalho futuro é o redesign do modelo em cima dos resultados obtidos no estudo de prova de conceito realizado. Esse trabalho não será muito longo e pode ser realizado em um curto prazo com uma nova rodada de estudos para nova prova de conceito. Após mais uma rodada de prova de conceito com participante do projeto, o próximo passo será levar o modelo a ser testado por usuários reais, ou seja, alunos e professores que precisam do apoio do modelo para realização da transferência do conhecimento entre duas linguagens de programação.

No médio prazo, um trabalho futuro desejável é a implementação desse modelo em mais de uma instância a fim de fazer a validação do mesmo. Todo o modelo foi criado baseado em estudos realizados com duas ferramentas, AgentSheets e GreenFoot. Seria interessante no médio prazo que o modelo fosse implantado para outras LP’s, assim poderíamos medir se o modelo é de fato válido da forma genérica como ele se propões.

No longo prazo, um trabalho futuro pode explorar a expansão desse modelo de forma que ele fique ainda mais genérico. Por exemplo, o modelo como definido hoje é para ser aplicado a duas linguagens de programação voltadas para o ensino, onde a primeira é visual e a segunda textual. Seria um trabalho muito importante a expansão desse modelo para que ele seja aplicado a quaisquer duas linguagens de programação, sejam elas de aprendizado ou não e visuais ou textuais.

7

Referências bibliográficas

BASAWAPATNA, A. R., KOH, K. H., & REPENNING, A. (2010). **Using scalable game design to teach computer science from middle school to graduate school**. Proceed-ings of the fifteenth annual conference on Innovation and technology in computer science education (ITiCSE '10). New York.

Caitlin Kelleher and Randy Pausch. 2005. **Lowering the barriers to programming**: A taxonomy of programming environments and languages for novice programmers. ACM Comput. Surv. 37, 2 (June 2005), 83-137. DOI=10.1145/1089733.1089734 http://doi.acm.org/10.1145/1089733.1089734

COOPER, S. (2010). **The Design of Alice**. ACM Trans. Comput. Educ. 10.

DALY, T. Minimizing to maximize: an initial attempt at teaching introductory programming using Alice. **Journal of Computing Sciences in Colleges**, 2011. 23-30.

Dann, W., Cosgrove, D., Slater, D., Culyba, D., & Cooper, S. (2012). Mediated Transfer: Alice 3 to Java. SIGCSE. North Carolina.

de Souza, C. S. (2005b). **The semiotic engineering of human-computer interaction**. Cambridge, Mass.: The MIT Press.

de Souza, C. S., & Leitão, C. F. (2009). **Semiotic Engineering Methods for Scientific Research in HCI**. Princeton: NJ. Morgan & Claypool.

de Souza, C. S., Garcia, A. C., Slavieiro, C., Pinto, H., & Repenning, A. (2011). **Semiotic traces of computational thinking acquisition**. Third International Symposium on End-User Development, IS-EUD (pp. 155-170). Torre Canne (BR), Italy: Springer Berlin Heidelberg.

de Souza, C. S., Salgado, L. C., Leitão, C. F., & Serra, M. M. (2014). **Cultural appropriation of computational thinking acquisition research**: seeding fields of diversity. ITiCSE '14 Proceedings of the 2014 conference on Innovation & technology in computer science education (pp. 117-122). New York, NY, USA: ACM.

de Souza, C., Barbosa, S., & Silva, S. R. (2001). **Semiotic Engineering Principles for Evaluating End-User Programming Environments**. Interacting with Computers, (pp. 467-495). Amsterdam.

Erwig, M & Meyer, B (1995). **Heterogeneous Visual Languages - Integrating Visual and Textual Programming**. VL ’95, pp. 318–325

Floyd, R. W. (1979). **The paradigms of programming**. Communications of the ACM 22 .

Forgarty, R., Perkins, D., & & Barell, J. (1991). **The Mindful School**: How to Teach for Transfer. IRI/Skylight Publishing.

Haberman, N. R., & Bruria, H. (2010**). Linking different programming paradigms**: thoughts about instructional design. Innovation and technology in computer science education. New York.

Harvey, B. (1997). **Computer Science Logo Style** (Vol. III). MIT Press.

HAREL, I. a. (1991). **Situating Constructionism. In Constructionism**. Norwood.

Kölling, M. (November de 2010). **The Greenfoot Programming Environment.** ACM Transactions on Computing Education , 10 (4).

MALONEY, J., RESNICK, M., RUSK, N., SILVERMAN, B., & EASTMOND. (2010). **The Scratch Programming Language and Environment**. Trans. Comput. Educ. 10, (p. 15).

Monteiro, I. T. ; de SOUZA, C. S. ; TOLMASQUIM, E. T. . **My Program, My World**: Insights from 1st -Person Reflective Programming in EUD Education.. In: 5th International Symposium, IS-EUD 2015, 2015, Madri - Espanha. Proceedings of the Fifth International Symposium on End-User Development (IS-EUD 2015). Suíça: Springer International Publishing, 2015. p. 76-91. Ian Utting,

Mota, M. P. (2014). **PoliFacets**: um modelo de design da metacomunicação de documentos ativos para apoiar o ensino e aprendizado de programação. Rio de Janeiro: Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Paul Denny, A. L.-R. (2011). **Understanding the Syntax Barrier for Novices**. ITiCSE. Darmstadt.

Pierce, J. S. (1998). **Alice**: Easy to learn interactive 3d graphics. CHI 98 conference summary on Human factors in computing systems. Los Angeles.

Reinfelds, J. (1995). **A three paradigm first course for CS majors**. SIGCSE technical symposium on Computer science education. New York.

Repenning, A., & Ioannidou, A. (2004). **Agent-based end-user development**. Communications of the ACM , 47 (9), pp. 43-46.

Stephen Cooper, Michael Kölling, John Maloney, and Mitchel Resnick. 2010. **Alice, Greenfoot, and Scratch -- A Discussion**. Trans. Comput. Educ. 10, 4, Article 17 (November 2010), 11 pages. DOI=10.1145/1868358.1868364 http://doi.acm.org/10.1145/1868358.1868364

Turkle, S. (2005). The Second Self. MIT Press.

Tomohiro Nishida, S. K. (2009). **A CS unplugged design pattern**. ACM technical symposium on Computer science education (SIGCSE '09). New York.

Wilensky, U. (1999). **NetLogo**. Center for Connected Learning and Computer-Based Modeling. Evanston: Northwestern University.

WING, J. M. (2006). **Computational thinking**. Commun. ACM .

Apêndices

Apêndice 1 – COMANDOS DO AGENTSHEETS

Tabela 5 - Comandos AgentSheets (Condições)

Condições	
SEE	A condição SEE verifica se o agente na célula indicada pelo parâmetro direção tem a mesma representação especificada no parâmetro representação.
KEY	A condição KEY verifica se a tecla exibida em seu parâmetro está pressionada.
ONCE-EVERY	A condição ONCE-EVERY verifica se uma quantidade N de segundos se passaram desde a última vez que ela foi verdadeira.
IS	A condição IS verifica se o valor de um atributo é igual ao valor passado pelo parâmetro.
STACKED	A condição STACKED verifica se o agente está na mesma célula que outro cuja representação foi especificada no parâmetro representação.
%-CHANCE	A condição %-CHANCE é verdadeira com uma determinada percentagem que é especificado pelo parâmetro por um número.
STACKED-A	A condição STACKED verifica se o agente está na mesma célula que outro cujo tipo foi especificado no parâmetro tipo.
SEE-A	A condição SEE-A verifica se o agente na célula indicada pelo parâmetro direção tem a mesmo tipo especificado no parâmetro tipo.
EMPTY	A condição EMPTY verifica se a célula indicada pelo parâmetro direção está vazia.

Tabela 6 - Comandos AgentSheets (Ações)

Ações	
MOVE	A ação MOVE move um agente de uma célula na direção indicada pelo o parâmetro. O ponto (.) Vai deixar o agente na sua posição atual.
MESSAGE	A ação MESSAGE transmite uma mensagem a todos agentes para que eles executem o método passado como parâmetro.
ERASE	A ação ERASE apaga o agente localizado na célula indicada pelo parâmetro.
CHANGE	A ação CHANGE muda a representação do agente na célula para a representação indicada pelo parâmetro.
WAIT	A ação WAIT faz com que o AgentSheets espere a quantidade de segundos indicado por um número antes de executar o próximo comando.
SET	A ação SET define o valor de um atributo do agente ou o valor de uma propriedade, para o valor calculado de uma fórmula.
RESET-SIMULATION	A ação RESET-SIMULATION para a simulação e retorna a planilha para o estado original
BROADCAST	A ação BROADCAST transmite uma mensagem a todos os agentes da classe indicado no parâmetro de classe da ação para que eles executem o método passado como parâmetro.
NEW	A ação NEW cria um novo agente na célula indicada pelo parâmetro de direção passado.
SHOW_MESSAGE	A ação SHOW_MESSAGE exibe uma Janela com a mensagem passada no parâmetro.
SWITCH-TO-WORKSHEET	A ação SWITCH-TO-WORKSHEET muda para uma planilha especificada pelo parâmetro e altera o foco da simulação para a nova planilha.
SAY	A ação SAY fala o texto localizado em seu

	parâmetro usando o sintetizador de voz Macintosh.
STOP-SIMULATION	A ação STOP-SIMULATION para a simulação.
TRANSPORT	A ação TRANSPORT faz com que um agente transporte outro que esteja na mesma célula que ele.
PLAY-SOUND	A ação PLAY-SOUND reproduz o som escolhido em seu parâmetro.
MOVE-RANDOM-ON	A ação MOVE-RANDOM-ON permite que um agente se mova aleatoriamente para qualquer um dos seus vizinhos imediatos que tenham a representação passada no parâmetro.

Apêndice 2 – TERMO DE CONSENTIMENTO

TERMO DE CONSENTIMENTO

Título da pesquisa: Investigação sobre modelo proposto para transferência de raciocínio computacional entre duas linguagens de programação
Pesquisadores: João Antonio Dutra Marcondes Bastos e Clarisse de Souza (SERG/DI – PUC-Rio http://www.serg.inf.puc-rio.br)
Participante: _____

Caro (a) participante, solicitamos seu consentimento para participar de um estudo a respeito de uma ferramenta de promoção de transferência do raciocínio computacional. Nossas pesquisas envolvem a condução de estudos exploratórios relacionados ao modelo proposto. Esse modelo tem como objetivo levar o usuário a aprender uma nova linguagem de programação baseado em uma anterior que ele já conheça.

A atividade que lhe propomos terá duração estimada de 60 minutos e está dividida em três partes. A primeira parte você poderá interagir com o sistema PoliFacets que você já conhece, explorando todas as facetas do jogo "Corredor de Bolinha". A seguir, você deve migrar para o sistema de transferência do raciocínio computacional onde faremos uma navegação guiada pelas facetas de transição que estão sendo propostas. Por último será pedido que você recrie o projeto "Corredor de Bolinhas" na nova linguagem de programação chamada GreenFoot.

O importante para nossa pesquisa é perceber como usuários navegam pelo modelo de transferência proposto. Queremos investigar se o modelo de interação proposto é eficiente e pode levar o usuário a aprender uma nova linguagem de programação baseado nos conhecimentos já adquiridos em outra linguagem.

As informações solicitadas neste estudo serão tratadas dentro das normas éticas de conduta em pesquisa. Os nomes dos participantes não serão divulgados em nenhuma hipótese, os resultados da pesquisa serão apresentados respeitando-se rigorosamente a privacidade e o anonimato dos participantes. Você tem pleno direito de solicitar esclarecimentos adicionais e de interromper o experimento quando e como quiser. Não há qualquer impedimento para isto nem qualquer necessidade de apresentar uma justificativa ou explicação.

Desde já, agradecemos sua participação, caso esteja interessado(a). Para prosseguir, porém, pedimos que manifeste por escrito o seu consentimento para realizarmos as atividades descritas acima.

Li os termos da pesquisa acima e consinto em participar:

_____
(Assinatura do participante)

Comprometo-me a seguir rigorosamente a conduta ética neste experimento:

_____
João Antonio D. M. Bastos (jbastos@inf.puc-rio.br)