Aprendizagem de Programação Através de Ambientes Lúdicos em um Curso de Engenharia de Computação: Uma Primeira Incursão

Roberto A. Bittencourt, Anderson S. Rocha, Bianca L. Santana, Carla S. Santana, Douglas A. Carneiro, Gutemberg A. Borges, Henderson S. Chalegre, Jadson F. J. Silva, Jéssica Magally J. Santos, Lucas A. Silva, Pedro H. M. O. Andrade

Universidade Estadual de Feira de Santana (UEFS)
Av. Transnordestina, s/n – 44.036-900 – Feira de Santana – BA – Brasil

roberto@uefs.br

Abstract. *Difficulties to learn computer programming in Computing undergraduate programs contribute to high retention rates and an early dropout in these programs. Such difficulties may be caused by the short time frame required to learn complex concepts and by the lack of motivation, which is worsened by the use of teacher-centered learning approaches. This paper proposes a solution to these issues through the use of programming learning environments that are more playful and entertaining. We carried out two active learning workshops based on the Scratch, Greenfoot and BlueJ tools during one week. Results suggest an increase in student motivation and in the ease to learn central programming concepts.*

Resumo. *Dificuldades de aprendizagem em programação de computadores em cursos de graduação em Computação contribuem para a evasão e o abandono precoce destes cursos. Estas dificuldades podem ser causadas pela complexidade dos conceitos a serem aprendidos em um curto período de tempo e pela falta de motivação, agravada pelo uso de metodologias de aprendizagem centradas no professor. Este artigo propõe uma solução para enfrentar estas dificuldades através da utilização de ambientes lúdicos para a aprendizagem de programação. Duas oficinas de aprendizagem ativa baseadas nas ferramentas Scratch, Greenfoot e BlueJ foram realizadas durante uma semana. Os resultados sugerem um aumento na motivação dos estudantes e na facilidade em aprender conceitos centrais de programação.*

1. Introdução

A aprendizagem de programação perpassa uma quantidade razoável de conhecimentos, habilidades e competências. Exige capacidade de abstração, raciocínio lógico e o domínio de noções de álgebra abstrata e sintaxe e semântica de linguagens. Demanda ainda a aquisição de conhecimentos, habilidades e competências próprios de algoritmos, estruturas de dados, linguagens de programação e teoria de sistemas. Para complicar ainda mais, esta diversidade de conceitos, habilidades e competências é trabalhada em um período bastante intensivo em cursos de graduação em Computação. De modo geral, dois semestres letivos são normalmente o tempo destinado ao aprendizado de tipos de dados primitivos, variáveis, atribuição, computação, estruturas de seleção, estruturas de

repetição, funções e procedimentos, interpretação, compilação, ligação, bibliotecas, modularização, estruturas de dados homogêneas e heterogêneas, dados em memória e em arquivos, tipos de dados abstratos, encapsulamento, ocultação de informação, classes, objetos, mensagens, coesão, acoplamento, associação, composição, agregação, herança, interfaces, polimorfismo, interfaces com o usuário (gráficas ou em linha de comando), depuração e testes. Além disso, quando se inicia a aprendizagem com o paradigma imperativo, há ainda o problema adicional da transição para o paradigma orientado a objetos, que provoca um conflito cognitivo nos aprendizes, geralmente demorado de resolver [Kölling et al. 1995]. Percebe-se que este período de um ano é normalmente muito curto para a formação das competências necessárias ao desenvolvimento de software.

Por outro lado, há ainda a questão da motivação para a aprendizagem de programação. A existência de sistemas de software cada vez mais complexos e a utilização destes sistemas desde a infância e adolescência faz com que haja um descompasso entre o tipo de software que os alunos gostariam de desenvolver com aquele que eles são, a princípio, capazes de desenvolver com o restrito conjunto de competências adquiridas nos primeiros semestres de um curso de graduação em Computação. Esta questão é comumente agravada por uma tendência em se utilizar metodologias de ensino-aprendizagem excessivamente formais, normalmente devida à formação rigorosa dos professores responsáveis pelo ensino de programação [Rapkiewicz et al. 2006]. Neste contexto, é difícil para boa parte dos estudantes se interessarem por problemas abstratos propostos pelos professores. Mesmo problemas mais práticos, porém considerados obsoletos para a realidade dos estudantes enquanto usuários de sistemas podem também reduzir a motivação para o aprendizado.

Para a maioria dos alunos de cursos de computação, os conceitos e habilidades requeridos para programar têm poucos paralelos com suas experiências prévias na educação básica. Como resultado frequente, observa-se alto nível de evasão e o abandono precoce destes cursos. Pesquisa recente do Observatório SOFTEX aponta a baixa ocupação de vagas e a alta evasão nos cursos da área de tecnologia de informação (TI) [SOFTEX 2012]. Dados desta pesquisa informam que, em 2008, havia 198.905 vagas em cursos *core* da área de tecnologia de informação. Neste ano, estas vagas foram aproveitadas por apenas 91.448 ingressantes, levando a uma taxa média de ocupação de 46%. Já a taxa anual de evasão em 2008 foi de 21%, levemente inferior aos anos anteriores. No acumulado, a evasão levou à formação de apenas 33.709 egressos em 2008. Este baixo nível de formação redundou em um déficit de 38.102 profissionais em 2008. O mesmo estudo projetou ainda um déficit de 116.362 profissionais no ano de 2012.

Em nosso curso de Engenharia de Computação, a evasão de alunos entre 2009 e 2011 foi de 20%, embora dados preliminares apontem para um aumento nestes índices. Já a retenção nas disciplinas introdutórias de programação oscila entre 50 e 75%. No ano de 2012, para um ingresso de 40 alunos por semestre letivo, a disciplina de introdução à programação do primeiro semestre contou com 60 a 70 alunos matriculados por período, enquanto que o componente curricular do segundo semestre, que trata de programação orientada a objetos (POO) e estruturas de dados, teve entre 50 a 60 alunos matriculados por período.

Este trabalho propõe uma solução para estes problemas baseada em uma metodologia ativa de ensino-aprendizagem combinada com o uso de ambientes lúdicos projetados para pessoas mais jovens. As ideias centrais da solução proposta são:

- Aprender programação através do desenvolvimento de jogos e outras atividades divertidas em ambientes lúdicos pode fortalecer a motivação dos ingressantes em cursos de graduação em Computação;
- A complexidade da aprendizagem de programação pode ser reduzida com a utilização de ambientes de aprendizagem projetados especialmente para o ensino de programação de adolescentes e jovens adultos;
- A aprendizagem centrada no aluno, onde este dirige o seu processo de aprendizagem e os tutores acompanham o andamento do processo, interferindo apenas quando necessário, permite a cada aprendiz evoluir no seu próprio passo.

Numa incursão inicial, realizamos duas oficinas de aprendizagem antes da primeira semana de aulas do primeiro período de 2013. Na primeira, os alunos calouros de nosso curso aprenderam a programar um jogo usando a ferramenta Scratch, explorando conceitos básicos de programação. Na segunda, os alunos do segundo semestre aprenderam noções básicas de POO e Java fazendo um pequeno jogo e montando diagramas de objetos usando Greenfoot e BlueJ. Os resultados são encorajadores, tanto em relação à motivação como à aprendizagem de novos conceitos.

2. Trabalhos Relacionados

Pesquisadores da área de educação em computação tem se debruçado sobre estas questões com o objetivo de reduzir as barreiras à aprendizagem de programação [Keleher e Pausch 2005] [Guzdial 2004]. Estes estudos têm produzido ferramentas voltadas ao ensino e aprendizagem de programação para estudantes iniciantes. Algumas destas ferramentas realçam a visualização de conceitos, relações e estruturas, permitindo concretizar a sua dimensão abstrata de forma compreensível aos iniciantes [Kölling et al. 2003] [Gondim e Ambrósio 2008]. Outras ferramentas procuram desenvolver um espaço lúdico para a aprendizagem da programação, permitindo que seus usuários criem jogos, cenários e atividades de interesse de crianças, adolescentes e jovens adultos. Este é o caso de Scratch [Resnick et al. 2009] e Greenfoot [Kölling 2010], ambientes que têm sido usados com sucesso com crianças [Resnick et al. 2009], adolescentes [Al-Bow et al. 2008] e em disciplinas introdutórias de programação em cursos de graduação em computação [Moskal et al. 2004].

No Brasil, há relatos de algumas experiências de ensino-aprendizagem de programação utilizando o lúdico como elemento motivador nos primeiros períodos de cursos da área de tecnologia de informação [Rebouças et al. 2010] [Rapkiewicz et al. 2006]. Uma destas experiências utiliza a linguagem Python, que permite eliminar a complexidade inicial da programação, devido a seu interpretador e a seu sistema de tipos dinâmico [Rebouças et al. 2010], mas não encontramos relatos aprofundados de utilização de ferramentas projetadas especificamente para a aprendizagem de programação como Scratch e Greenfoot.

3. Ambientes Lúdicos de Aprendizagem de Programação

A seguir, são descritos os ambientes de aprendizagem utilizados neste trabalho.

3.1. Scratch

Scratch é um ambiente desenvolvido pelo Lifelong Kindergarten Research Group do MIT que objetiva: (i) tornar a programação de computadores fácil de aprender; (ii) oportunizar a realização de projetos com complexidade crescente, e (iii) dar suporte a diferentes tipos de projetos para atender a diferentes interesses e estilos de aprendizagem [Resnick et al. 2009]. Inspirado na filosofia dos brinquedos Lego e em uma pedagogia socioconstrutivista, Scratch foi projetado para atender três princípios básicos de design: ser mais manipulável, mais significativo e mais social. A gramática de Scratch possui um conjunto de blocos de programação similares a tijolos Lego que permite aos aprendizes montar scripts para objetos através de manipulação de blocos de controle e entrada e saída, dentre outros. Para ser mais significativo, Scratch baseia-se em diversidade e personalização: vários tipos de projetos (e.g., jogos, estórias, animações) podem ser criados e diferentes mídias podem ser criadas, importadas e personalizadas. Finalmente, projetos Scratch podem ser guardados em uma rede social própria, que funciona como uma comunidade de compartilhamento, cooperação e socialização.

3.2. Greenfoot

Greenfoot é um ambiente de desenvolvimento integrado educacional voltado para a aprendizagem de programação de estudantes a partir de 14 anos de idade, combinando programação orientada a objetos em Java com gráficos interativos [Kölling 2010]. Para engajar os estudantes, Greenfoot foi desenvolvido para ser fácil de usar e de se descobrir funcionalidade, ser flexível, prover *feedback* rápido, permitir interações sociais, e ser extensível. Para auxiliar os professores, o ambiente permite fácil visualização e interação, um modelo mental consistente, privilegia conceitos em vez de sintaxe, evita sobrecarga cognitiva e provê suporte aos professores em uma comunidade virtual. O ambiente provê um framework de classes (e.g., *World* e *Actor*) que permite desenvolver interações e jogos através de classes derivadas, criadas através de manipulação direta e modificadas via um editor de código-fonte.

3.3. BlueJ

BlueJ é um ambiente de desenvolvimento integrado projetado para ensinar orientação a objetos através de uma pedagogia específica [Kölling et al. 2003]. Segundo seus autores, a aprendizagem de orientação a objetos padece da falta de ferramentas adequadas e experiência pedagógica com este paradigma. No ambiente BlueJ, estudantes aprendem a programar em Java primeiramente através de interações gráficas utilizando diagramas UML de classes e de objetos. Os estudantes interagem com os objetos e visualizam o sistema através de uma interface simples e fácil de aprender. A pedagogia do BlueJ propõe algumas diretrizes: aprender objetos primeiro, não iniciar com uma tela vazia, aprender a ler código, usar projetos “grandes”, não começar com “main”, não usar “Hello World”, mostrar a estrutura do programa e tomar cuidados específicos com interface de usuário.

4. Oficina de Programação Básica com Scratch

A seguir são descritos o planejamento e os resultados da oficina de programação básica para alunos de primeiro semestre usando a ferramenta Scratch.

4.1. Planejamento

Esta oficina objetivou o aprendizado de conceitos básicos de algoritmos e programação por estudantes calouros de um curso de Engenharia de Computação através de uma metodologia ativa e lúdica. Como objetivo secundário, a oficina pretendeu reduzir problemas de evasão e retenção nas primeiras disciplinas de programação. A oficina, de 4 horas de duração, foi realizada antes do início regular das aulas. Contou com 26 participantes, foi facilitada por 6 estudantes, sendo 2 tutores e 4 monitores, e foi observada por um professor. Dos participantes, apenas cinco tinham algum conhecimento de programação (e.g., PHP, C, Pascal).

A metodologia utilizada consistiu, em poucas palavras, de aprender fazendo e aprender brincando. Os tutores propunham desafios e os participantes procuravam descobrir o conhecimento e resolver os problemas navegando no ambiente Scratch. Foram auxiliados pelos monitores apenas quando o solicitaram.

A oficina foi baseada em um jogo de arco e flecha, inspirado no clássico jogo *Arrow & Bow*, disponível no Windows 95 e em versões posteriores. Ao desenvolver o jogo, os participantes adquiriram as competências de compreender requisitos, expressar uma solução algorítmica e programar em uma linguagem visual. Este jogo simples permitiu aprender a utilizar os conceitos disponíveis nos blocos funcionais fornecidos pelo Scratch tais como estruturas de controle (dentre elas, seleção e repetição), leitura de sensores, saída através de aparência e áudio, eventos de desenho e movimento, operadores lógicos e relacionais, além do uso de variáveis.

O observador seguiu um protocolo de pesquisa qualitativa, anotando eventos da oficina, entremeando as observações com curtas reflexões. Os tutores e monitores fizeram uma autoavaliação na forma de reflexão. Os participantes foram convidados a responder um questionário após a oficina, apontando pontos positivos e negativos.

4.2. Resultados

A oficina começou com uma curta apresentação do ambiente do Scratch, como criar projetos nele e como compartilhá-los com a comunidade. Em seguida, o jogo de arco e flecha foi proposto como desafio, de modo a permitir a prática de conceitos inicialmente considerados difíceis, do ponto de vista dos próprios tutores e monitores, na primeira disciplina de programação (e.g., como estruturas de seleção e repetição). Além disso, conceitos de comunicação entre objetos, algo trivial no Scratch, também foram trabalhados, permitindo conhecer elementos de programação orientada a objetos.

Mesmo sem conhecimentos prévios de programação pela maioria dos participantes, eles conseguiram utilizar o ambiente de forma intuitiva para criar os cenários e os objetos do jogo. As dúvidas iniciais dos participantes eram mais no sentido de como montar os scripts de cada objeto, utilizando os elementos de sequência, repetição e seleção. Explicações curtas foram usadas inicialmente pelos tutores, enquanto os monitores acompanhavam os participantes individualmente. O restante da

oficina transcorreu suavemente, com independência relativa dos participantes e acompanhamento, quando solicitado, pelos tutores e monitores.

O jogo proposto foi apenas um desafio geral, que podia ser personalizado de acordo com os interesses dos estudantes. Alguns avançaram mais rápido, introduzindo cenários e comportamentos diferentes (e.g., um participante fez o balão furado pelo arco ser restaurado à condição normal após algum tempo).

Dos comandos disponíveis no Scratch, todos os participantes utilizaram os comandos de controle, aparência, movimento, sensores e operadores. Nem todos usaram variáveis, embora a maioria o tenha feito para controlar a quantidade de flechas e para sinalizar o final do jogo. Os monitores auxiliavam os participantes em seus questionamentos, bastante comuns em relação a sensores e operadores, mas menos comuns em relação a estruturas de controle.

Para obter *feedback* dos participantes, um curto questionário foi enviado a eles, contendo questões gerais sobre a oficina. Ao serem questionados se gostaram da ferramenta Scratch, foram unânimes em relação a isto, como na resposta de um deles: "Curti muito, achei simples e que dá uma ideia lógica boa". Também foram questionados sobre o que mais e o que menos gostaram. A maioria respondeu que o que mais gostaram foi aprender noções da lógica de programação. Sobre o que menos gostaram, poucos opinaram, geralmente mencionando o pouco tempo e conteúdo. A maioria considerou o desafio proposto do arco e flecha como satisfatório, embora alguns o acharam muito básico. Finalmente, acerca do interesse deles em fazer um curso mais completo sobre Scratch, todos demonstraram bastante interesse em continuar o aprendizado com a ferramenta.

A reflexão dos tutores e monitores foi de que a oficina de Scratch superou as expectativas, especialmente o receio que tinham de que o interesse dos participantes seria pequeno em desenvolver software em uma plataforma desenvolvida para crianças e adolescentes, com seu aspecto "quase infantil". Porém, este receio foi afastado com a oficina, quando puderam ver o alto interesse dos participantes. Para os tutores e monitores, através do desafio proposto, os participantes desenvolveram as competências esperadas e compreenderam os conceitos em questão. Embora num nível elementar, conceitos importantes de programação tais como laços, estruturas condicionais e variáveis foram aprendidos.

5. Oficina de Programação Orientada a Objetos com Greenfoot e BlueJ

A seguir são descritos o planejamento e os resultados da oficina de programação orientada a objetos para alunos de segundo semestre usando as ferramentas Greenfoot e BlueJ.

5.1. Planejamento

Esta oficina objetivou o aprendizado de conceitos programação orientada a objetos (POO) por estudantes do segundo semestre de um curso de Engenharia de Computação através de uma metodologia ativa e lúdica. Como objetivo secundário, a oficina pretendeu reduzir problemas de evasão e retenção nas disciplinas de programação e projeto orientado a objetos, através da suavização da transição do paradigma de programação procedimental para o orientado a objetos. A oficina, de 6 horas de duração,

foi realizada antes do início regular das aulas. Contou com 19 participantes, foi facilitada por 4 estudantes que se revezaram nas funções de tutores e monitores, e foi observada por um professor. Como a audiência era opcional em uma semana de integração, alguns participantes já haviam cursado (e sido reprovados em) uma disciplina de POO, embora a maioria tenha sido de alunos de segundo semestre que ainda iriam cursar a disciplina de POO.

A metodologia utilizada foi a de aprender fazendo através de desafios maiores de construir um jogo e modelar um pequeno sistema divididos em desafios menores relacionados à programação em Java nos ambientes Greenfoot e BlueJ. Os tutores propunham desafios em níveis de complexidade crescente e gradual, e os participantes procuravam descobrir o conhecimento e resolver os problemas. Eventuais dúvidas e erros eram discutidos. Alguns erros mais relevantes eram discutidos pelos tutores com todos os participantes, provocando-os a apontar os problemas e as soluções, e, algumas vezes, apresentando algumas soluções mais elegantes.

Quatro horas foram dedicadas ao ambiente Greenfoot, utilizando como desafio a construção de um jogo de labirinto, onde uma formiga deveria alcançar um doce no final. Duas horas foram dedicadas ao ambiente BlueJ, onde os estudantes modelaram um problema de contas bancárias. Ao desenvolver o jogo, os participantes adquiriram as competências de compreender requisitos, e projetar e desenvolver uma solução orientada a objetos utilizando um ambiente integrado de desenvolvimento. Estes desafios permitiram aprender a utilizar os conceitos de classes, objetos, atributos, métodos, construtores, passagem de mensagens entre objetos, herança e polimorfismo.

Como na oficina de Scratch, o observador seguiu um protocolo de pesquisa qualitativa, anotando eventos e fazendo curtas reflexões. Tutores e monitores se autoavaliaram através de reflexões. Os participantes foram convidados a responder um questionário após a oficina, apontando seus aspectos positivos e negativos.

5.2. Resultados

A sessão da oficina que tratou de Greenfoot foi dividida em três etapas: i) a apresentação da ferramenta; ii) a apresentação informal e discussão de conceitos básicos de POO, ilustradas com um exemplo usando formigas e um formigueiro, e uma tentativa de concretizar o exemplo usando a ferramenta e iii) a utilização da ferramenta para construir um jogo de labirinto, onde uma formiga devia alcançar um *cupcake*.

Para auxiliar na compreensão dos conceitos básicos, um exemplo de labirinto simples foi mostrado, e os participantes tiveram 20 minutos para construir os seus, sendo auxiliados pelos tutores e monitores. Com isto, criaram classes e instanciaram objetos. O passo seguinte foi fazer uma formiga se movimentar, escrevendo os métodos para isto, reaproveitando métodos da classe *Actor*. Os tutores mostraram a API de *Actor* e lançaram o desafio para os participantes descobrirem os métodos apropriados e os codificarem, em um período de 5 minutos. As soluções foram então discutidas, apontando eventuais erros. Problemas de controle de colisões com as paredes e suas soluções foram também discutidos. Na etapa final, foi lançado o desafio de a formiga reconhecer o *cupcake* e o jogo ser terminado com uma tela diferente da do labirinto. Dúvidas surgiram, foram discutidos os métodos de acesso a posições, e os participantes acabaram atingindo os objetivos.

Já a sessão da oficina que tratou de BlueJ teve três etapas: i) formalizar os conceitos de classes, objetos, atributos, métodos e construtores através de um exemplo; ii) adicionar o conceito de herança ao exemplo; iii) adicionar comportamento polimórfico no exemplo modificado.

O uso de diagramas UML em BlueJ permitiu rever de modo mais formal os conceitos de POO. Um problema de contas bancárias foi lançado como desafio para trabalhar habilidades de modelagem. Cada etapa dava um tempo para reflexão e discussão. Primeiro, os participantes descobriram os atributos necessários. Em seguida, o comportamento. Dúvidas surgiram e foram esclarecidas neste momento. Em seguida, os participantes instanciaram objetos no BlueJ, procurando realçar a diferença entre classes e instâncias, e destacar a necessidade de construtores. Na segunda parte, foi lançado o desafio que levaria ao conceito de herança em contas bancárias, o que podia ser praticado imediatamente na ferramenta, alterando o diagrama de classes. Aspectos de implementação do código-fonte foram também discutidos neste momento. Finalmente, comportamento polimórfico foi discutido usando um método de saque das contas bancárias, e os participantes tiveram oportunidade de instanciar objetos de superclasses e subclasses e testarem o seu comportamento.

Tentou-se captar a opinião dos participantes através da observação e de um questionário com questões gerais sobre a oficina. Os resultados apontaram que eles gostaram das ferramentas, da oficina e da atuação dos tutores e monitores. Durante a oficina, um participante que já tinha estudado POO mencionou: “Poxa, por que na minha época não tinha uma ferramenta legal como essa para eu aprender POO e Java?”. Outros afirmaram que “É divertido ficar mexendo nisso aqui!” ou “Vou fazer um jogo massa!”. Por outro lado, alguns participantes sugeriram ainda que a oficina foi muito curta e que seria importante discutir os conceitos de modo mais formal.

A reflexão dos tutores e monitores, junto com a opinião dos participantes, ajuda a sustentar as hipóteses de que é possível aprender conceitos de POO de modo intuitivo e lúdico, que a interação entre pares (estudantes experientes e novatos) ajuda no aprendizado, e que o formato desafio/solução auxilia na aprendizagem.

6. Lições Aprendidas

Dentre as lições importantes, percebemos que é importante propor bons problemas, tanto de modo mais amplo, para a construção de software, como para a discussão de conceitos específicos. Para estudantes novatos, é importante manter uma boa razão monitor-aluno (em nosso caso, de 1 para 5 ou 6). Notamos benefícios claros em discutir falhas nas soluções dos estudantes, o que os levou a reverem seus conceitos. Além da discussão, o uso constante de perguntas para incentivar o raciocínio mostrou-se efetiva. A utilização de jogos como desafios aumenta a motivação: observamos, durante todo o tempo, os participantes sempre ativos e motivados; vários não queriam sair dos computadores nos intervalos. Finalmente, há dois aspectos que observamos facilitar a aprendizagem: o uso de diagramas para compreender o mundo dos objetos; e o uso constante de analogias com assuntos conhecidos deles, fortalecido pelo universo comum compartilhado entre os pares em uma oficina conduzida por estudantes de graduação.

Dentre os problemas observados, as dificuldades de infraestrutura e logística foram os maiores, gerando alguns atrasos no andamento das oficinas. A participação de

vários monitores requer espaços adequados para circulação, nem sempre disponíveis em laboratórios didáticos. Por conta da curta duração, é necessário planejar com antecedência a instalação de software, bem como a disponibilização de arquivos auxiliares a serem usados pelos participantes. Houve também problemas específicos de cada oficina. Na primeira, alguns monitores não estavam totalmente conscientes do tempo necessário ao ciclo desafio-resposta, e deram respostas precoces às questões dos participantes. Na segunda, os participantes, estudantes de segundo período já acostumados ao modelo formal universitário, sentiam ausência de formalização de conceitos. Além disso, questionaram o tempo curto da oficina.

Em relação a ofertas futuras de oficinas, o que aprendeu desta experiência é de que deve haver um calendário de planejamento para garantir que equipamentos, software e materiais de apoio estejam disponíveis no início das oficinas. Ainda em relação a planejamento, vale destinar tempo para discutir elementos de uma pedagogia de aprendizagem ativa entre os estudantes tutores, para evitar que alguns deles reproduzam modelos de aprendizagem centrada no professor. Uma última sugestão para ofertas futuras seria a preparação ainda mais cuidadosa de desafios, começando com desafios bem simples, para situar os participantes na ferramenta, seguidos então de desafios gradativamente mais complexos tais como os jogos das oficinas realizadas.

Dentre os maiores benefícios alcançados com esta abordagem, podemos citar: i) para os tutores e monitores, a experiência de ensinar a seus pares, contribuindo para um ambiente de colaboração no curso; ii) a motivação elevada dos participantes e seu interesse em continuar aprendendo; iii) os indícios de que o embasamento conceitual precoce pode ajudar a enfrentar dificuldades em disciplinas introdutórias; e iv) a participação ativa dos estudantes durante todo o tempo de realização das oficinas.

7. Conclusões

Este trabalho apresentou um relato de experiência de duas oficinas de aprendizagem de programação com o apoio de ambientes educacionais lúdicos: uma voltada para o aprendizado de conceitos básicos de programação por ingressantes de um curso de Engenharia de Computação; e outra voltada para o aprendizado de noções de programação orientada a objetos por estudantes de segundo semestre. As oficinas foram facilitadas por estudantes, que atuaram como tutores e monitores, e usaram elementos de aprendizagem ativa e lúdica. Espera-se que o amadurecimento deste tipo de experiência permita reduzir os níveis de evasão e reprovação em disciplinas introdutórias de programação.

Acreditamos que os elementos principais que levaram ao sucesso desta incursão inicial foram: i) a preparação cuidadosa de desafios interessantes que permitiram tratar de conceitos de programação em níveis crescentes de complexidade e através de discussões interativas entre tutores e estudantes, e ii) a liberdade de ação dada aos participantes da oficina de atuar ativamente na solução de desafios que são de seu interesse, através da navegação de ambientes adaptados a programadores iniciantes.

Pretendemos continuar as atividades das oficinas de aprendizagem de programação em nosso curso de Engenharia de Computação. Atualmente, estamos oferecendo duas oficinas mais longas e voltadas para o mesmo público participante deste relato, usando as ferramentas Scratch e Greenfoot. Planejamos ainda a realização

de oficinas de programação com alunos da educação básica, como parte de um esforço de atração de talentos para a área de tecnologia de informação.

8. Agradecimentos

Os autores gostariam de agradecer ao professor Pablo Rodrigo Fica Piras, tutor do PET-Engenharias pela participação dos estudantes bolsistas do grupo PET neste projeto. Este trabalho tem o apoio financeiro do CNPq – Conselho Nacional para o Desenvolvimento Científico e Tecnológico, através do processo 454996/2012-8.

Referências

Al-Bow, M., Austin, D., Edgington, J., Fajardo, R., Fishburn, J., Lara, C., Leutenegger, S., e Meyer, S. (2008). Using greenfoot and games to teach rising 9th and 10th grade novice programmers. In Proceedings of the 2008 ACM SIGGRAPH symposium on Video games, pages 55–59. ACM.

Gondim, H. e Ambrósio, A. (2008). Esboço de fluxogramas no ensino de algoritmos. In WEI-Workshop sobre Educação em Computação, pages 109–117.

Guzdial, M. (2004). Programming environments for novices. Computer science education research, 2004:127–154.

Kelleher, C. e Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys (CSUR), 37(2):83–137.

Kölling, M. (2010). The greenfoot programming environment. ACM Transactions on Computing Education (TOCE), 10(4):14.

Kölling, M., Koch, B., e Rosenberg, J. (1995). Requirements for a first year object-oriented teaching language, volume 27. ACM.

Kölling, M., Quig, B., Patterson, A., e Rosenberg, J. (2003). The BlueJ system and its pedagogy. Computer Science Education, 13(4):249–268.

Moskal, B., Lurie, D., e Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. In ACM SIGCSE Bulletin, volume 36, pages 75–79. ACM.

Rapkiewicz, C., Falkembach, G., Seixas, L., dos Santos, N., Rosa, V., e Klemann, M. (2006). Estratégias pedagógicas no ensino de algoritmos e programação associadas ao uso de jogos educacionais. Novas Tecnologias na Educação, 4(2), 2006.

Rebouças, A. D. D. S., Marques, D., Costa, L. F. S., e Silva, M. A. A. (2010). Aprendendo a ensinar programação combinando jogos e Python. In XXI Simpósio Brasileiro de Informática na Educação, pages 1–10. SBIE-UFPB.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al. (2009). Scratch: programming for all. Communications of the ACM, 52(11):60–67.

SOFTEX (2012). Software e Serviços de TI: A Indústria Brasileira em Perspectiva, vol. 2. SOFTEX.