UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS PONTA GROSSA
GERÊNCIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENSINO DE CIÊNCIA E TECNOLOGIA
MESTRADO PROFISSIONAL EM ENSINO DE CIÊNCIA E TECNOLOGIA

ELENA MARIELE BINI

ENSINO DE PROGRAMAÇÃO COM ÊNFASE NA SOLUÇÃO DE PROBLEMAS

PONTA GROSSA
2010

ELENA MARIELE BINI

ENSINO DE PROGRAMAÇÃO COM ÊNFASE NA SOLUÇÃO DE PROBLEMAS

Dissertação apresentada como requisito parcial à obtenção do título de Mestre em Ensino de Ciência e Tecnologia, do Programa de Pós-Graduação em Ensino de Ciência e Tecnologia. Área de Concentração: Ciência, Tecnologia e Ensino, da Gerência de Pesquisa e Pós-Graduação, do Campus Ponta Grossa, da UTFPR.

Orientador: Prof. André Koscianski, Doutor

PONTA GROSSA

2010

Ficha catalográfica elaborada pela Divisão de Biblioteca
da Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa
n.50/10

B612 Bini, Elena Mariele

Ensino de programação com ênfase na solução de problemas / Elena Mariele Bini. – Ponta Grossa: [s.n.], 2010.
84 f.: il. ; 30 cm.
Inclui "Manual: Estratégia de Ensino"

Orientador: Prof. Dr. André Koscianski

Dissertação (Mestrado em Ensino de Ciência e Tecnologia) - Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa. Curso de Pós-Graduação em Ensino de Ciência e Tecnologia. Ponta Grossa, 2010.

1. Programação de computadores - estratégias de ensino. 2. Construcionismo - (Papert , Seymour). 3. Resolução de problemas – (Polya, George). I. Koscianski, André. II. Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa. III. Título.

CDD 507

Universidade Tecnológica Federal do Paraná
Campus de Ponta Grossa
Gerência de Pesquisa e Pós-Graduação
PROGRAMA DE PÓS-GRADUAÇÃO EM ENSINO DE CIÊNCIA E TECNOLOGIA

TERMO DE APROVAÇÃO

Título de Dissertação Nº **05/2010**

ENSINO DE PROGRAMAÇÃO COM ÊNFASE NA SOLUÇÃO DE PROBLEMAS

por

Elena Mariele Bini Ortiz

Esta dissertação foi apresentada às **09 horas e 30 minutos** de **11 de fevereiro de 2010** como requisito parcial para a obtenção do título de MESTRE EM ENSINO DE CIÊNCIA E TECNOLOGIA, com área de concentração em Ciência, Tecnologia e Ensino, linha de pesquisa em **Ciência e Tecnologia no Contexto do Ensino-Aprendizagem**, Programa de Pós-Graduação em Ensino de Ciência e Tecnologia. O candidato foi argüido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Dr. Esteban Walter Gonzalez Clua (UFF)

Prof. Dr. Jean Marcelo Simão (UTFPR)

Profª. Drª. Nilcéia Aparecida Maciel Pinheiro (UTFPR)

Prof. Dr. André Koscianski (UTFPR) -
Orientador

Visto do Coordenador:

Prof. Dr. Guataçara dos Santos Junior
Coordenador do PPGECT

Dedico esta dissertação a Deus, sem Ele, nada teria sido possível...

AGRADECIMENTOS

Primeiramente agradeço a Deus e a Mãe Maria, por terem me conduzido na realização deste trabalho.

Aos meus pais, Arlete e Luis, pelo esforço em oferecer-me o acesso ao ensino de qualidade, também por acreditarem sempre em meu potencial, tendo sido apoio contínuo em minha vida. Sou imensamente grata!

Aos meus irmãos, Jonny e Silvia, pelo apoio e compreensão.

Aos demais familiares e amigos, que direta ou indiretamente acompanharam essa jornada.

Ao meu amado esposo, pela dedicação e companheirismo, por ter sido minha fortaleza nos momentos difíceis, oferecendo seu largo sorriso.

A Luísa, que em meu ventre, foi a força necessária para o término do trabalho.

A todos os professores e funcionário da UTFPR envolvidos com o PPGECT.

Ao meu orientador, prof. Dr. André Koscianski, pela confiança, pela liberdade concedida, pelo aprendizado.

Aos colegas do programa de mestrado, não os esquecerei. Em especial, as amizades construídas: Andréia Hornes e Adriane Acqua. Adriane, seu espírito livre foi inspiração em vários momentos. Andréia, não há palavras para mencionar sua importância durante essa caminhada.

A Faculdade Guairacá, em especial ao seu Diretor Geral e Pedagógico, Juarez Matias Soares, pelo apoio que contribuiu para a viabilização deste trabalho.

E por fim, aos alunos do curso Técnico em Informática, participantes da pesquisa, pelas contribuições e dedicação na realização das atividades propostas.

RESUMO

BINI, Elena Mariele. **Ensino de programação com ênfase na solução de problemas.** 2010. 106 f. Dissertação (Mestrado em Ensino de Ciência e Tecnologia) – Programa de Pós-Graduação em Ensino de Ciência e Tecnologia, Centro Federal de Educação Tecnológica do Paraná. Ponta Grossa, 2010.

O aprendizado de programação de computadores é um processo complexo de aquisição de habilidades. Duas dificuldades principais se manifestam entre alunos novatos: a falta de envolvimento e motivação; e limitações quanto à capacidade de resolução de problemas. Procurando minimizar tais dificuldades são propostas estratégias de ensino de programação, alicerçadas no construcionismo, teoria proposta por Seymour Papert; e no uso de heurísticas para resolução de problemas, propostas por George Polya. Nesse quadro a elaboração de modelos mentais adequados foi considerada como necessária para o aprendizado eficaz. As estratégias de ensino são compostas por exemplos e atividades com caráter lúdico e desafiador. A aplicação do trabalho em uma turma de adolescentes cursando ensino técnico apresentou resultados positivos. As estratégias de ensino estão formalizadas em nove roteiros de aula e compõem um manual, produto deste trabalho.

Palavras-chave: Ensino. Programação de computadores. Resolução de problemas. Construcionismo.

ABSTRACT

BINI, Elena Mariele. **Teaching programming with emphasis on problem solving.** 2010. 106 f. Dissertação (Mestrado em Ensino de Ciência e Tecnologia) – Programa de Pós-Graduação em Ensino de Ciência e Tecnologia, Centro Federal de Educação Tecnológica do Paraná. Ponta Grossa, 2010.

Learning to program a computer is a complex process of skill acquisition. Two main difficulties are present among novice students: the lack of motivation and involvement; and limitations with respect to the capability to solve problems. Teaching strategies are proposed in order to minimize those difficulties, based on the construcionism, the theory developed by Seymour Papert; and on heuristics for problem solution, proposed by George Polya. In this framework, the construction of the adequate mental models was considered as a requirement for effective learning. The teaching strategies are composed of examples and activities with a ludic and challenging character. The application of the method with a group of teenagers of a technical course, showed positive results. The teaching strategies are described in nine phases. They are published in a manual that accompanies the dissertation.

Keywords: Teaching. Computer programming. Problem solving. Construcionism.

LISTA DE QUADROS

LISTA DE GRÁFICOS

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Um programa de computador pode ser definido como uma sequência de instruções, elaboradas a partir de uma linguagem de programação, que serão executadas pela máquina com o objetivo de resolver um problema específico. Elaborar programas é função atribuída aos profissionais da informática geralmente com formação voltada para a programação de computadores.

É justamente com a programação de computadores que o presente trabalho está relacionado, mais especificamente, com a disciplina chamada Lógica de Programação, ofertada na 2ª série do Curso Técnico em Informática, em uma escola da rede estadual de ensino do Estado do Paraná, onde a ementa prevê o ensino dos conceitos iniciais através da programação procedural.

Geralmente, os alunos que ingressam nessa disciplina apresentam características como pouca autonomia, pouca iniciativa e baixa capacidade para resolução de problemas. Em sua maioria, os estudantes apresentam posição de receptores passivos, pois não foram estimulados suficientemente a buscar além dos conceitos apresentados pelo professor. Também não foram estimulados a pensar sobre determinado problema e agir sobre ele, aplicando conhecimentos prévios.

Dunican (2002), a partir da experiência prática afirma ser possível identificar três categorias de alunos novatos em programação. Na primeira categoria estão aqueles alunos que não têm aptidão para compreensão dos conceitos básicos. Na segunda categoria estão aqueles alunos que podem compreender os conceitos fundamentais se esses forem expostos mediante uma abordagem pedagógica eficaz. E na terceira categoria estão aqueles alunos que se sentem confortáveis com a natureza abstrata da programação de computadores.

Ao longo da caminhada como professora da disciplina de Lógica de Programação, tenho percebido inúmeras dificuldades apresentadas pelos alunos novatos e que se apresentam como recorrentes em outros contextos. Essas dificuldades, aliadas ao caráter complexo e abstrato da disciplina, não contribuem com o real aprendizado por parte dos estudantes, desmotivando-os e aumentando os índices de reprovação e evasão escolar.

Minhas constatações aliadas às expressas por Dunican (2002), indicam ser necessário uma maior atenção aos alunos pertencentes à segunda categoria de

estudantes novatos em programação. No entanto, a maneira tradicional como a disciplina introdutória de programação é ensinada contribui para o aumento e/ou permanência das dificuldades presentes no processo.

As metodologias usuais para ensino inicial da programação se concentram em dois aspectos: (i) o desenvolvimento de algoritmos por meio do pseudocódigo ou fluxogramas e (ii) o uso de linguagens de programação voltadas para a indústria. Quando o professor enfatiza o primeiro aspecto citado, acaba contribuindo para aumentar o grau de abstração exigido no processo. No caso do segundo aspecto, o aluno acaba por empregar tempo para aprender uma carga complexa de regras de sintaxe e semântica, em vez de se concentrar em pensar e agir sobre o problema em questão. Assim, o desenvolvimento da habilidade de resolver problemas fica prejudicado pela insistência em um contexto abstrato, em um momento precoce; e ocorre um aprendizado precário de uma linguagem de programação.

A literatura apresenta diversas alternativas para minimizar o impacto das inúmeras dificuldades apresentadas pelos alunos novatos. Uma delas consiste em buscar diminuir os impactos causados pelo grau elevado de abstração exigido pela disciplina. Outra é avaliar a adequabilidade das linguagens de programação profissionais utilizadas, frente ao contexto da sala de aula. Geralmente, a consideração feita sobre a abstração presente no processo e a adequação de linguagens utilizadas, resulta no desenvolvimento de ferramentas computacionais.

Finalmente – mas sem extinguir todas as possibilidades – cabe mencionar também o desenvolvimento de estratégias de ensino. Essa tendência vem ganhando força e é representada por trabalhos relacionados à busca pela motivação dos estudantes, estilos de aprendizagem, ou ainda, estímulo à capacidade de resolução de problemas. Esta é a direção geral adotada neste trabalho.

As propostas já apresentadas por pesquisadores da área não encerram com a presença das dificuldades sentidas pelos alunos novatos em programação. Notadamente, a variedade de situações de ensino e os diferentes contextos em que o problema surge exigem que o professor sempre adapte a abordagem para obter os melhores resultados.

Essa dissertação defende a idéia de que a baixa capacidade para resolução de problemas é um dos fatores geradores das dificuldades presentes no processo de aprendizagem dos conceitos iniciais da programação de computadores.

A experiência como docente da disciplina de introdução ao ensino da programação e os indícios percebidos em vários trabalhos pesquisados, sugerem que essa disciplina enfatize o estímulo à resolução de problemas e não o ensino de uma linguagem de programação específica ou das estruturas básicas da programação. Essa mudança de abordagem se justifica face ao fato de que geralmente, ao ingressar em um curso de programação, os alunos mostram não ter sido suficientemente estimulados em momentos anteriores e apresentam dificuldades desde interpretar enunciados até planejar soluções. Esse estímulo é fundamental para que os estudantes possam atuar no mundo do trabalho, podendo levantar problemas, analisá-los e propor soluções adequadas. Estimular e/ou melhorar a capacidade de resolução de problemas dos alunos, os ajudará a se tornarem mais responsáveis pelo próprio aprendizado, habilidade essencial em profissionais inseridos produtivamente na sociedade contemporânea.

Assim, esta dissertação tem como problema central a seguinte pergunta: “Estratégias de ensino particulares poderão estimular e/ou melhorar a capacidade de resolução de problemas nos alunos, de maneira a diminuir as dificuldades presentes no processo de aprendizagem da programação de computadores?”.

A fim de responder esta questão, são propostas estratégias de ensino de programação de computadores que buscam tratar as principais dificuldades apresentadas pelos alunos no processo.

O trabalho pode ser desdobrado nos seguintes objetivos específicos:

- Implementar estratégias de ensino dos conceitos iniciais da programação de computadores, baseadas no estímulo à resolução de problemas, por meio do desenvolvimento de jogos e animações computacionais;
- Avaliar se a aplicação de estratégias voltadas para a resolução de problemas por meio dos jogos e animações permite estimular os alunos.

Este trabalho buscou inovar ao aliar explicitamente o construcionismo, teoria proposta por Seymour Papert, às heurísticas para resolução de problemas propostas por George Polya e à importância da elaboração de modelos mentais adequados, gerando assim estratégias de ensino que buscam estimular a resolução de problemas, através de atividades lúdicas e desafiadoras, a serem implementadas pelos alunos em ambientes de programação motivadores.

O texto apresentado está dividido em sete partes. Nele, o capítulo 1 apresenta a introdução, onde o problema é justificado, o contexto da pesquisa e seus objetivos são apresentados.

O referencial teórico da dissertação foi distribuído em seções que compõem o capítulo 2. Num primeiro momento apresentam-se os conceitos e processos envolvidos na programação de computadores, bem como as formas em geral empregadas para abordar esse assunto em sala de aula. Em seguida discutem-se as dificuldades existentes no aprendizado.

Buscando subsídios para formulação de roteiros de aula onde estejam presentes a diminuição das abstrações exigidas no processo e o estímulo a resolução de problemas em ambientes capazes de motivar o aluno, buscou-se os princípios de Modelos Mentais e Resolução de Problemas, bem como a teoria proposta por Seymour Papert, denominada Construcionismo, também apresentados no Capítulo 2. Já o Capítulo 3, dividido em duas seções, discute uma nova proposta para o ensino da programação: o Construcionismo aliado à resolução de problemas e o papel do professor nesse novo contexto.

O Capítulo 4, por sua vez, apresenta a metodologia adotada para o desenvolvimento desse trabalho e o contexto no qual está inserido. Já no Capítulo 5 o desenvolvimento da pesquisa realizada é melhor detalhado.

Os resultados obtidos são apresentados e discutidos no Capítulo 6 e as conclusões e sugestões para trabalhos futuros são apresentadas no Capítulo 7.

2 REFERENCIAL TEÓRICO

A primeira seção do referencial teórico apresenta alguns conceitos da programação de computadores, com o objetivo de melhor esclarecer ao leitor o contexto deste trabalho. Em seguida, o processo de ensino da programação é discutido e são destacadas algumas das principais dificuldades apresentadas pelos alunos novatos em programação e as propostas da literatura para amenizá-las.

Na quarta seção são apresentados princípios adotados para fundamentação das estratégias de ensino propostas nesse trabalho, bem como a teoria de Seymour Papert, o construcionismo.

2.1 A PROGRAMAÇÃO DE COMPUTADORES

Programar é a atividade de instruir o computador sobre quais ações ele deve realizar para executar determinada atividade. Para instruir o computador, os programadores utilizam ferramentas computacionais denominadas linguagens de programação. É através das linguagens de programação que são criados textos com as instruções que compõem um programa de computador, também chamado *software*. Esses textos são conhecidos como textos fonte ou simplesmente “fontes”.

A primeira fase para o desenvolvimento de um programa é a compreensão das funções necessárias. Em outras palavras, é preciso entender o problema a ser resolvido. Após o levantamento das necessidades o *software* será planejado e posteriormente construído. Na fase de construção do programa é comum se empregar diagramas que refletem a estrutura e funcionamento do mesmo. No caso de programas pequenos, como os habitualmente trabalhados em uma disciplina introdutória, os diagramas de descrição de arquitetura não são usados. Uma vez disponível o texto fonte, ele passa por um processo denominado compilação. A compilação é realizada por um *software* responsável por traduzir o código fonte escrito por um ser humano em uma representação binária que pode ser executada pela máquina.

A escolha da linguagem de programação a ser utilizada está associada a diversos critérios; um dos mais importantes é um paradigma de desenvolvimento de

software. Um paradigma de desenvolvimento de *software* pode ser entendido como um modelo de programação, um padrão que molda a forma com que o programador deve pensar e expressar esse pensamento, além de orientar outras fases do desenvolvimento, como as pertinentes a engenharia de *software*. O quadro 1 apresenta os principais paradigmas de desenvolvimento de *software*. Para uma leitura mais avançada sugere-se a obra de Sebesta (2006).

Paradigma	Descrição
Paradigma imperativo ou procedimental	A solução de um problema é descrita por um conjunto finito de instruções executadas seqüencialmente.
Paradigma declarativo	Descrevem-se propriedades do problema e da solução, mas não a sequência de instruções. Assim, os resultados esperados são especificados e não os procedimentos para produzi-los.
Paradigma funcional	A solução de um problema é descrita como o resultado de uma função da entrada. Não se emprega o conceito de estado, existente em outros paradigmas.
Paradigma orientado a objetos	Todo problema é analisado em um cenário em que objetos interagem para trocar informações e realizar computações.

Quadro 1: Principais paradigmas de desenvolvimento de *software*
Fonte: Autoria própria

Via de regra, um mesmo problema computacional pode ser resolvido usando qualquer paradigma. A escolha de um ou outro paradigma pode resultar em maior esforço do programador. Para um exemplo o leitor pode consultar Baranauskas (1993).

2.2 O PROCESSO DE ENSINO DA PROGRAMAÇÃO DE COMPUTADORES

A disciplina que aborda os conceitos iniciais da programação de computadores, apresentando as estruturas das linguagens, como utilizá-las de forma a organizar e relacionar os passos que serão executados pelo computador é geralmente denominada Lógica de Programação.

Lógica de programação em processamento de dados é a maneira pela qual se representa em linguagem corrente, ou por meio de símbolos que expressam essa linguagem, instruções, que compõem um programa a ser executado por um computador. (CARBONI, 2003, p.11)

O aprendizado da lógica de programação se dá, geralmente, através da escrita de algoritmos; esse fato é apontado também por Carboni (2003). Um algoritmo é caracterizado por representar sequências finitas de passos para atingir um determinado objetivo.

Existem diferentes representações de algoritmos, como fluxogramas e pseudocódigo. Para ilustrar essas diferentes representações, considere o problema computacional: calcular a média aritmética das notas bimestrais de um estudante do ensino médio. O algoritmo corresponde a pedir ao usuário do computador para que sejam digitadas as quatro notas bimestrais, em seguida, ele deverá apresentar a média aritmética dessas notas e a situação do estudante: aprovado, reprovado ou exame. Para elaborar um algoritmo com a possível solução para este problema será aplicada a técnica de refinamentos sucessivos. Essa técnica consiste em dividir um problema maior em subproblemas, com o objetivo de tratar gradativamente a complexidade do mesmo. Isto é ilustrado no quadro 2 que apresenta os refinamentos realizados até se chegar a uma descrição muito próxima de um programa que possa ser implementado em um computador.

O quadro 2 apresenta, em pseudocódigo, comandos de saída, entrada e de atribuição, além de estruturas denominadas variáveis. O comando de saída é representado pelo comando ESCREVA. Esse comando é o responsável pelo envio de mensagens ao usuário do algoritmo. O comando de entrada, por sua vez, é o comando LEIA. Tal comando é responsável por capturar dados enviados pelo usuário à memória do computador. A memória do computador é acessada através da manipulação de variáveis. Uma variável computacional é um endereço na memória RAM (memória de acesso randômico) dos computadores, cujo conteúdo pode ser alterado no decorrer do programa. Variáveis constituem um dos conceitos fundamentais na construção de programas. No exemplo apresentado no quadro 2 são variáveis Nota1, Nota2, Nota3, Nota4 e Media. O comando de atribuição é aqui representado por uma flecha '←'. O uso do comando de atribuição implica fazer com que a variável que está à sua esquerda receba um valor. Esse valor pode ser obtido do conteúdo de outra variável, de uma operação lógica ou aritmética, ou de um valor constante. No exemplo citado a variável Media recebe a soma das quatro notas bimestrais divida por 4, ao final o resultado de Aluno Aprovado, Reprovado ou em Exame é apresentado ao usuário. A figura 1 apresenta o mesmo algoritmo

representado em fluxograma. O quadro 3 apresenta a descrição dos itens do fluxograma apresentado na figura 1.

1º Refinamento
obtenha notas
calcule média e situação
imprima resultados

2º Refinamento
leia as quatro notas
some os valores e divida o resultado por quatro
verifique situação do aluno de acordo com a média
imprima a média e a situação

3º Refinamento
leia as variáveis n1, n2, n3, n4
média = (n1+n2+n3+n4)/4
se média > 7, aluno aprovado
caso contrário, se média > ... aluno em recuperação
se média < aluno reprovado

4º Refinamento
leia as variáveis n1, n2, n3, n4
média = (n1+n2+n3+n4)/4
imprima média
se média > 7, imprima aluno aprovado
se média > 5 e média < 7 imprima. aluno em recuperação
se média < aluno reprovado

```
Escreva (“Digite a nota referente ao 1º Bimestre”);
Leia (Nota1);
Escreva (“Digite a nota referente ao 2º Bimestre”);
Leia (Nota2);
Escreva (“Digite a nota referente ao 3º Bimestre”);
Leia (Nota3);
Escreva (“Digite a nota referente ao 4º Bimestre”);
Leia (Nota4);
Media ← (Nota1+Nota2+Nota3+Nota4)/4;
Escreva (“A média aritmética das notas apresentadas é”, Media);
Se Media >= 7 Então
    Escreva(“Aluno aprovado!”);
Senão
    Se Media < 4 Então
        Escreva (“Aluno reprovado!”);
    Senão
        Escreva (“Aluno em exame!”);
```

Quadro 2: Exemplo de algoritmo representado em Pseudocódigo
Fonte: Autoria própria

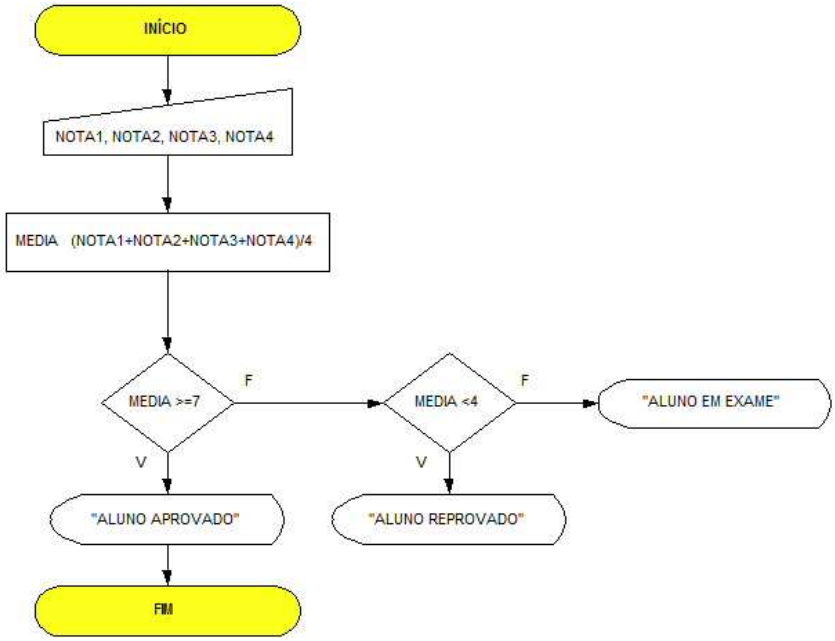
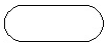
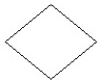


Figura 1: Exemplo de algoritmo utilizando a forma de representação Fluxograma
Fonte: Autoria própria

Símbolo	Descrição
	Início ou fim do fluxograma
	Processo/operação
	Entrada de dados via teclado
	Saída de dados pelo monitor
	Ponto de decisão

Quadro 3: Descrição dos itens do fluxograma apresentado na figura 1
Fonte: Autoria própria

Depois de apresentadas as definições de alguns conceitos fundamentais da programação de computadores, é possível relacioná-los com o processo de aprendizagem.

O ensino da programação de computadores geralmente é dividido em duas fases. Na primeira, os alunos são apresentados aos conceitos elementares da programação como: variáveis, comandos de entrada, de saída e de atribuição, estruturas de seleção e repetição. Depois disso tem-se uma segunda fase,

geralmente mais extensa, subdivida em mais de um ano ou semestre letivo. Nela o aluno é desafiado a desenvolver programas conforme uma escala de níveis de complexidade determinados pelo professor. Em alguns casos, o paradigma de programação é alterado na passagem da primeira para a segunda fase. O quadro 4 apresenta tais fases e recursos de ensino utilizados pelos professores, em especial quando da programação procedural.

Fase	Recursos geralmente utilizados
Primeira fase	• Construção de algoritmos baseados em pseudocódigo e/ou fluxogramas; • Ferramentas computacionais desenvolvidas com objetivos didáticos; • Ambientes de programação comerciais.
Segunda fase	• Ambientes de programação comerciais.

Quadro 4: Fases e recursos geralmente utilizados no ensino da programação de computadores
Fonte: Autoria própria

A figura 2 apresenta um trecho de código criado na linguagem de programação *Python*, correspondente ao algoritmo apresentado no quadro 5. Já a figura 3 apresenta a execução desse programa.

```
Soma, Numero1, Numero2: inteiro;
Escreva (“Digite um número qualquer”);
Leia (Numero1);
Escreva (“Digite outro número qualquer”);
Leia (Numero2);
Soma  Numero1 + Numero2;
Escreva (“A soma dos números digitados é”, Soma);
```

Quadro 5: Exemplo de algoritmo utilizando a forma de representação Pseudocódigo
Fonte: Autoria própria

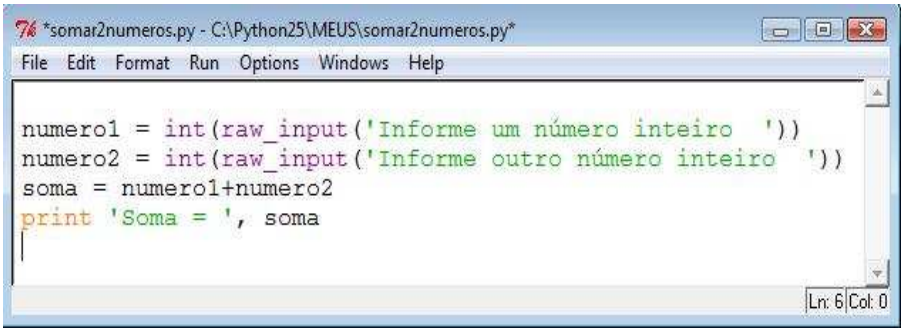


Figura 2: Código em linguagem *Python*
Fonte: Autoria própria

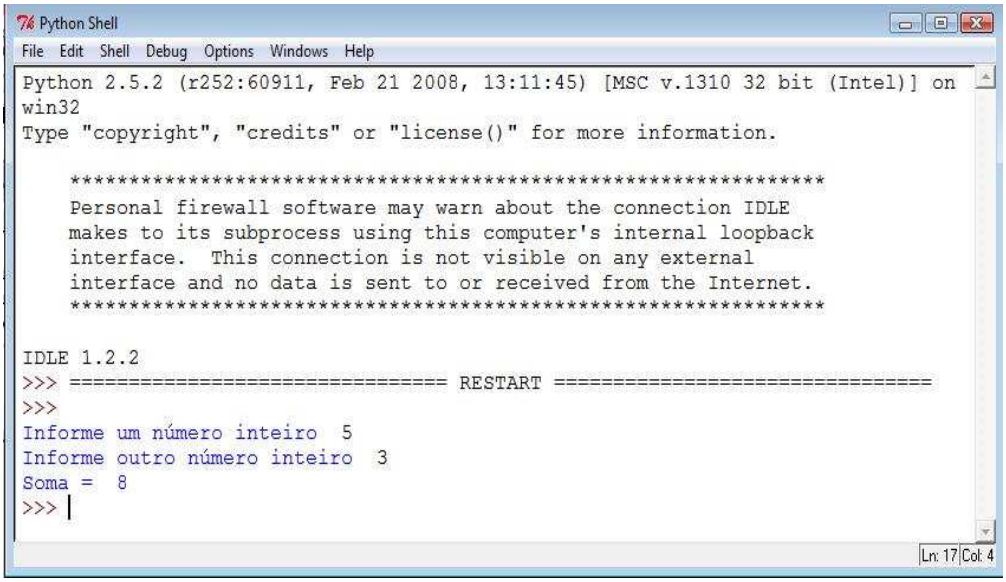


Figura 3: Execução do programa apresentado na figura 2
Fonte: Autoria própria

O paradigma imperativo costuma ser o primeiro ensinado nos cursos de programação, visando apresentar aos alunos conceitos fundamentais como estruturas de seleção e repetição, variáveis, comandos de entrada e saída. Sua utilização é muitas vezes justificada pela estrutura seqüencial, fácil de seguir e que pode tornar o processo de aprendizado menos abstrato. Geralmente, o ensino da programação é marcado pela aplicação do paradigma imperativo e, em seguida, ao considerar apreendidos os conceitos fundamentais da programação de computadores, passa-se ao aprendizado do paradigma orientado a objetos. Todavia, esse cenário vem sendo alterado e algumas instituições de ensino fazem a opção por apresentar aos estudantes o paradigma orientado a objetos desde o primeiro contato com a programação de computadores.

2.3 DIFICULDADES ENCONTRADAS PELOS ALUNOS

Pode-se determinar como objetivo das atividades didático-pedagógicas assegurar que os alunos que apresentam dificuldades no processo de aprendizagem da programação de computadores tenham progresso. No entanto, sabe-se que atingir tal objetivo não é tarefa simples.

Pereira e Rapkiewicz (2004) relacionam as dificuldades apresentadas pelos alunos com as abstrações envolvidas no processo. Vargas e Martins (2005) também citam o elevado nível de abstração exigido quando o aluno não visualiza a execução

por meio do computador. A ausência de comprovação prática como gerador de dificuldades também é citada por Pinheiro (2003). A não compreensão de pseudocódigo e desenvolvimento de fluxogramas é mencionada por Miranda (2004) como propulsora de dificuldades em parte dos alunos novatos em programação. Motil e Epstein (2000), por sua vez, citam as linguagens de programação utilizadas nas disciplinas introdutórias, com muitas e complexas regras de sintaxe, fator de desencadeamento de dificuldades. Complexa sintaxe também é mencionada por Chen e Morris (2005). Gomes et al (2008) corrobora a idéia ao afirmar que metodologias tradicionalmente utilizadas para aprender e ensinar a programação de computadores não se mostram suficientes. A baixa capacidade para resolução de problemas, representada por não entendimento do problema, falta de planejamento, não aplicação de conteúdos conhecidos para solução de novos problemas é citada por diferentes autores como: Gomes et al (2008), Petry (2005), Kelleher e Pausch (2005), Sobral (2008). Falckembach e Araujo (2006) complementa as dificuldades apresentadas ao mencionar uma possível resistência por parte do aluno ao aprendizado, o que pode levar a ainda mais dificuldade. Adicionado a isso, o trabalho menciona a não possibilidade de o professor adequar-se as necessidades de cada aluno. Ala-Mutka (2004) afirma que o não reconhecimento das próprias deficiências pode levar alunos novatos a enfrentar dificuldades relacionadas.

Toda gama de dificuldades apresentadas gera sérias consequências como a desmotivação, reprovação e evasão escolar. Já sobre as possíveis causas de tais dificuldades não existe um consenso, nem tão pouco sobre possíveis soluções para a minimização das mesmas. Assim como encontramos na literatura a defesa pela continuação da construção de algoritmos, baseada no ensino tradicional como em Miranda (2004), encontramos também inúmeras pesquisas que apresentam o desenvolvimento e a aplicação de ferramentas e metodologias para o ensino da programação de computadores. Alguns exemplos são: Almeida et al. (2002) com um conjunto de ferramentas computacionais que auxiliam no ensino de programação; Maloney et al. (2008) com a utilização do ambiente de programação *Scratch*; Petry (2005) com um sistema de suporte ao ensino e aprendizagem no domínio de algoritmo, por meio de um companheiro de aprendizagem virtual; Pinheiro (2003), que sugere aliar a linguagem Pascal ao ambiente de programação LOGO.

O quadro 6 apresenta uma síntese das dificuldades apresentadas pelos estudantes mencionadas na literatura.

Síntese de algumas dificuldades mencionadas na bibliografia	
Dificuldades	Alguns autores
Abstrações envolvidas no processo	Moskal et al. (2000) Naps et al. (2003) Pereira e Rapkiewicz (2004) Vargas e Martins (2005)
Ausência de comprovação prática	Pinheiro (2003)
Não compreensão de pseudocódigo e desenvolvimento de fluxogramas	Miranda (2004)
Linguagens de programação com muitas e complexas regras de sintaxe	Motil e Epstein (2000) Chen e Morris (2005)
Baixa capacidade para resolução de problemas	Kelleher e Pausch (2005), Petry (2005), Gomes et al (2008), Sobral (2008)
Resistência por parte do aluno e não reconhecimento de suas próprias dificuldades	Ala-Mutka (2004) Falckembach e Araujo (2006)
Impossibilidade do professor adequar-se as necessidades de cada aluno	Falckembach e Araujo (2006)

Quadro 6: Síntese de algumas dificuldades mencionadas na literatura
Fonte: Autoria própria

Uma análise mais profunda sobre as dificuldades apresentadas pelos alunos durante o aprendizado da programação de computadores leva a percepção de quão complexo esse tema é; diferentes fatores e diferentes responsáveis contribuem para o surgimento e a permanência de tais dificuldades. Tais fatores estão representados na figura 4.

O sistema de ensino, especialmente o público, muitas vezes com estrutura física não adequada deixa de permitir uma carga horária viável para aulas práticas em laboratório de informática. Já alguns professores despreparados, não conseguem adaptar-se à necessidade de seus alunos, outros ainda não importam-se devidamente com as questões acerca das dificuldades apresentadas pelos estudantes, utilizando metodologias de ensino que prolongam a permanência das dificuldades no ambiente de ensino. Os próprios alunos também podem ser responsáveis pelo surgimento e permanência de dificuldades no processo de ensino, pois alguns apresentam postura não adequada como resistência, pouco interesse e pouca dedicação fora do ambiente escolar.

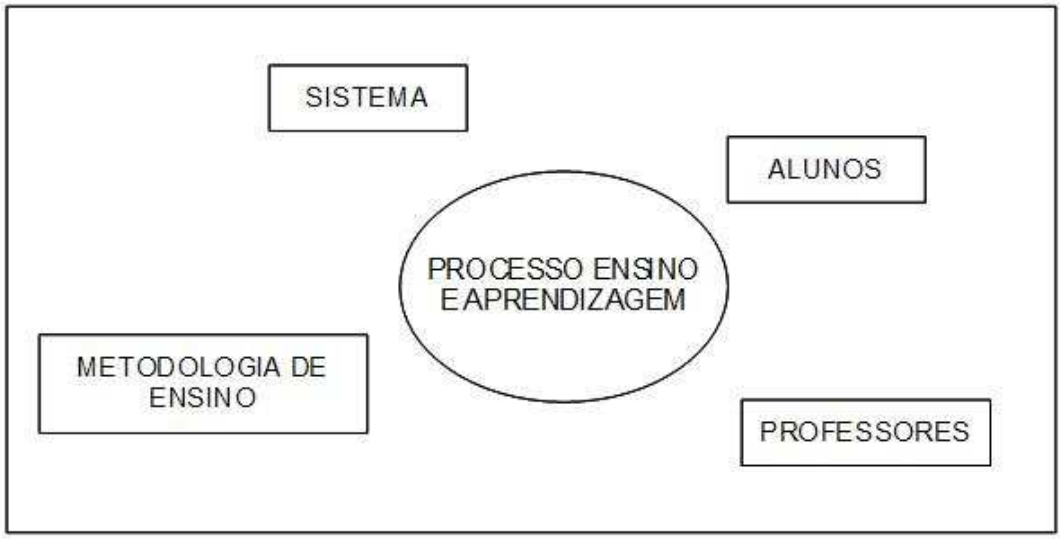


Figura 4: Fatores responsáveis pelo surgimento de dificuldades no processo de ensino e aprendizagem da programação
Fonte: Autoria própria

Há uma série de propostas para diminuir as dificuldades, porém nenhuma se mostrou completa; o problema não foi encerrado e é possível que não venha a ser, pois programar exige habilidades como raciocínio lógico, criatividade e capacidade para resolução de problemas, habilidades que são difíceis de ensinar e estimular.

2.3.1 Propostas da Literatura para Amenizar os Problemas

As dificuldades apresentadas pelos alunos novatos em programação de computadores são objeto de pesquisas no Brasil e no mundo há anos. Há diferentes abordagens e aqui traçaremos uma visão geral.

Pereira e Rapkiewicz (2004) apresentam um estudo sobre as publicações brasileiras relacionadas ao ensino da programação, no período 1999 a 2003. Foram analisados trabalhos do WEI (Workshop sobre Educação em Computação) e do SBIE (Simpósio Brasileiro de Informática na Educação), eventos que fazem parte do conjunto de eventos anuais promovidos pela Sociedade Brasileira de Computação – SBC.

O trabalho não permite comparações e análises profundas, no entanto algumas conclusões são possíveis: percebe-se um aumento gradativo de publicações sobre o tema durante os anos estudados. Sobre os temas dessas publicações, o trabalho classifica em: (i) desenvolvimento de ferramentas, (ii)

propostas de metodologias de ensino; (iii) ferramentas aliadas a metodologias. O número de publicações sobre o desenvolvimento de ferramentas computacionais foi maior que o número de publicações sobre propostas de metodologias de ensino, sejam elas isoladas ou aliadas a ferramentas computacionais; somando-se o número de publicações acerca de metodologias (25%) e metodologias de ensino aliadas à ferramentas computacionais (25%), temos 50% das publicações, mesmo índice representativo sobre o desenvolvimento de ferramentas.

Fazendo um levantamento semelhante para as publicações do WEI durante os anos de 2005 a 2008, obtivemos o gráfico 1 com o número de publicações sobre o ensino da programação e o gráfico 2 seguindo a mesma classificação apresentada em Pereira e Rapkiewicz (2004).

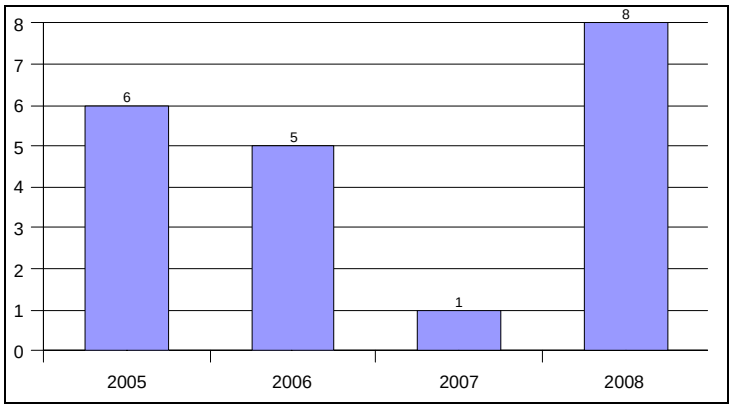


Gráfico 1: Publicações sobre o ensino da programação WEI2005 ao WEI2008
Fonte: Autoria própria

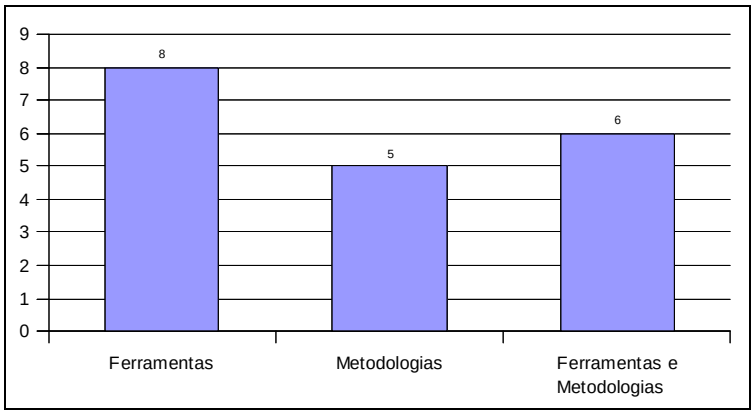


Gráfico 2: Foco das publicações - WEI 2005 ao WEI 2008
Fonte: Autoria própria

Houve ao menos uma publicação relacionada ao ensino da programação de computadores no evento, durante os anos de 2005 a 2008, sendo que o desenvolvimento de ferramentas para o ensino da programação continua a frente de propostas de metodologias de ensino, porém agora com menor representatividade. As publicações acerca do desenvolvimento de ferramentas somam 42%, enquanto publicações sobre propostas de metodologia representam 26% e metodologias de ensino aliadas à ferramentas computacionais correspondem a 32%.

A proposta de metodologia de ensino não está presente apenas em publicações brasileiras, encontrando-se também pesquisas sobre o tema em outros países, como na Finlândia por Ala-Mutka (2004), Estados Unidos por Cooper et al. (2003); África do Sul com Havenga (2008) e Reino Unido com Jenkins (2002).

Características como estilos de aprendizagem do educando e seus conhecimentos prévios, foco na resolução de problemas, ensino baseado em ambientes dinâmicos e motivadores estão, geralmente, presentes em metodologias de ensino propostas em recentes publicações, nacionais e internacionais. Alguns exemplos:

- Delgado et al. (2004): discussões sobre uma metodologia de ensino por competências, que privilegia o trabalho por problemas e projetos onde tarefas complexas e desafios podem incitar os alunos a mobilizar seus conhecimentos e completá-los. Tal metodologia está organizada em três fases: (i) resolução de problemas, (ii) formalização e (iii) construção de algoritmos;
- Deek et al. (1998): proposta e discussões de uma metodologia que visa a independência intelectual dos educandos através da resolução de problemas, aprendizagem coletiva e aprendizagem centrada no aluno;
- Gomes e Mendes (2007): descrição de fases para resolução de problemas e identificação de alguns obstáculos enfrentados por estudantes ao tentar resolver problemas de programação.

Fato interessante percebido ao analisar propostas de metodologias de ensino da programação de computadores é a mudança de visão de pesquisadores que acreditaram, em momento anterior, que o desenvolvimento de uma ferramenta seria suficiente para amenizar as dificuldades dos alunos novatos em programação. Publicações mais recentes desses autores apostam em metodologias de ensino para atingir tal objetivo. Um exemplo é Gomes et al. (2008) no qual os autores citam

o desenvolvimento, realizado por eles, de diferentes ferramentas que foram validadas, porém que não retornaram resultados suficientes. Na data da publicação (em 2008) o grupo trabalhava no desenvolvimento de outra ferramenta aliada a uma metodologia capaz de estimular o aluno, de forma lúdica, a resolução de problemas.

Estudos como Goosen (2004), Mclver e Conway (1996) e Parker et al. (2000) indicam características necessárias a linguagens e ambientes de programação para que possam contribuir com o processo de aprendizado. Entre as características mencionadas estão simplicidade de sintaxe; simplicidade de semântica; estruturas de linguagem de fácil entendimento; fácil diagnóstico de erros; fácil instalação e recursos para o apoio da resolução de problemas.

A alteração no foco das publicações indica que o desenvolvimento e aplicação de ferramentas computacionais para o ensino da programação não pode ser visto como único ingrediente para tratar as dificuldades apresentadas pelos alunos novatos em programação. Faz-se necessário aliar tais ferramentas a uma metodologia de ensino voltada à resolução de problemas criando um ambiente motivador ao aprendizado. Nesse processo, é possível empregar ferramentas computacionais já existentes.

Uma linha de trabalho adotada é a utilização de micromundos. Micromundos são ambientes controlados que oferecem possibilidades bem determinadas para interação aluno e computador, como um repertório limitado de ações. O ambiente de programação LOGO proposto por Seymour Papert na década de 1970 é um exemplo de micromundo aplicado à programação de computadores. Os ambientes de programação Alice e *Scratch* são exemplos atuais de micromundos. Essas ferramentas têm caráter lúdico e motivador, pois através delas os alunos podem desenvolver jogos e animações gráficas. Relatos da aplicação do ambiente de programação *Scratch* para ensino da programação de computadores podem ser encontrados em Cristóvão (2008a) e Wolz et al. (2009). Já relatos da aplicação do ambiente de programação Alice para o ensino da programação de computadores podem ser encontrados em Cooper et al. (2003) e Moskal et al. (2000).

2.4 PRINCÍPIOS E TEORIA ADOTADOS PARA A SOLUÇÃO DOS ASPECTOS DE ENSINO E APRENDIZAGEM IMPLICADOS

Essa seção discute a importância da construção de modelos mentais adequados durante a construção de um programa. A capacidade de resolução de problemas e o papel do professor também são discutidos. A teoria proposta por Seymour Papert denominada por ele de construcionismo também é apresentada.

2.4.1 Modelos Mentais

Ao interagir com a realidade as pessoas constroem modelos mentais do ambiente, sendo capazes de interagir com tais modelos antes mesmo de interagir com a realidade em questão. Assim, um modelo mental pode ser compreendido como uma representação interna do mundo externo.

De acordo com Silva (1991):

> As pessoas quando interagem com o ambiente, com outras pessoas e com artefatos da tecnologia elas [sic] formam um modelo mental de si próprias, das pessoas e das coisas com as quais estão interagindo. (SILVA, 1991, p. 184)

No contexto educacional, a elaboração de modelos mentais inadequados sobre determinado conceito pode levar o indivíduo ao não aprendizado correto. Por exemplo, caso o aluno não seja capaz de construir um modelo mental adequado sobre o relevo paranaense ele não será capaz de identificar e descrever características quando questionado; provavelmente o aluno não responderá uma questão ou responderá de forma incorreta.

Em programação de computadores a elaboração de modelos mentais adequados é imprescindível.

> Enquanto alunos novatos apenas compreendem a sintaxe de uma linguagem de programação sem o aporte de um modelo mental adequado, eles são incapazes de compreender conceitos complexos com êxito ou participar em processos de resolução de problemas mais difíceis. Modelos mentais adequados são de importância crucial para a compreensão dos conceitos da programação. (GEORGE, 2000, p.1)

Silva (1991, p. 186) comenta sobre a dificuldade que alunos novatos encontram em elaborar um modelo mental adequado:

> Também em programação de computadores, principiantes são guiados por modelos mentais de como o código do programa controla as operações do computador. Mas o computador, como sistema físico é extremamente obscuro e fornece um *feedback* muito pobre no sentido de auxiliar um principiante a entender seu funcionamento e conseqüentemente formar um modelo mental adequado, que o possibilite interagir corretamente.

Já Ben-Ari (1998) afirma que todo equívoco tem um modelo mental inadequado que o gerou. Nesse sentido, alunos novatos em programação cometem equívocos por formularem modelos mentais inadequados.

Para Khalife (2006) há equívocos comuns na forma como um programador novato percebe um computador, pois sente dificuldades para distinguir o computador enquanto visão de usuário e enquanto visão de programador. Segundo ele, os alunos precisam desenvolver um modelo mental simples, porém concreto, do funcionamento interno da máquina. Para isso é proposto um modelo de computador que pode ser considerado como o início para a aprendizagem da programação. Isso é realizado pelas fases apresentadas no quadro 7.

Fase	Nomenclatura	Descrição
1	Definir um computador	Os alunos novatos devem ser capazes de compreender e articular a definição de um computador como uma ferramenta que processa dados de acordo com um conjunto de instruções, onde o processamento consiste em entrada, armazenamento e manipulação, e saída;
2	Definir os principais componentes de hardware	A discussão do que é um computador deve ser genérica, breve e limitar-se a condição prévia de compreender como os programas são executados. O entendimento da relação entre memória e processador apresenta-se como importante;
3	Definir um subconjunto de instruções genéricas	São introduzidas as instruções de entrada e saída, bem como atribuição e declaração de variáveis. Novatos podem construir sobre essa base conforme mais instruções são inseridas;
4	Resolução de problemas	A introdução precoce de problemas a serem resolvidos no formato de tarefas, como problemas para compreensão, decomposição, problemas de modelagem, implementação e testes aumentam as chances do aluno adentrar, de fato, na programação.

Quadro 7: Processo de construção do modelo de computador proposto por Khalife (2006)
Fonte: Autoria própria

Para auxiliar os alunos a elaborar modelos mentais, professores utilizam analogias, sejam elas faladas ou através de imagens. No entanto, tais

representações podem ser insuficientes. Para exemplificar isto, considere-se a analogia comum para o funcionamento da memória RAM, como uma grande caixa de correio dividida em pequenas caixas postais numeradas. A deficiência evidente dessa analogia é o fato de que em uma caixa de correio podemos colocar diversas correspondências e de tipos variados (revistas, cartas, boletos bancários), enquanto em um determinado espaço da memória de um computador podemos armazenar um dado por vez e de apenas um tipo.

Outros recursos utilizados para ensinar a programação de computadores, além das analogias, são o pseudocódigo e o fluxograma, já mencionados. Tais representações são consideradas temporárias porque serão substituídas por código para dar origem a um programa de computador.

Ainda na tentativa de auxiliar o aluno novato em programação na elaboração de um modelo mental adequado, outros professores optam pela utilização de sistemas computacionais para auxiliar nesse processo. Algumas ferramentas, como Alice e *Scratch*, podem contribuir.

Durante o processo de ensino e aprendizagem da programação de computadores, há que se considerar o fato de modelos mentais serem evolutivos e alteráveis conforme as interações que ocorrem, ou seja, recursivamente o modelo mental vai ganhando complexidade, sendo melhorado até chegar a funcionalidade desejada. Isso ocorre graças à reflexão do aluno sobre os comandos do algoritmo ou programa elaborado. Assim, permitindo atividades práticas de resolução de problemas que permitam a colaboração e a criatividade os alunos serão capazes de adaptar modelos mentais previamente construídos assim como criar novos modelos conforme apresentarem-se necessários.

2.4.2 Resolução de Problemas

A baixa habilidade em resolver problemas é um dos fatores geradores das dificuldades apresentadas pelos alunos novatos em programação. Essa deficiência tem raízes na formação anterior.

Para que adquiram a habilidade de resolver problemas, é preciso instigar os alunos a pensar sobre determinadas situações problemas e agir sobre elas,

buscando soluções. Instigar os alunos a resolver problemas é motivá-los a percorrer um caminho ainda não explorado.

Para auxiliar o processo de resolução de problemas matemáticos, o matemático Polya apresenta quatro heurísticas em sua obra *How to Solve It*, publicada em 1975: (i) compreender o problema; (ii) estabelecer um plano; (iii) executar o plano e (iv) retrospecto.

A relação entre programação de computadores e o trabalho de Polya já foi objeto de estudos, como os citados no quadro 8.

Heurísticas de Polya	Gomes e Mendes (2007)	Havenga (2008)
(i) compreender o problema; (ii) estabelecer um plano; (iii) executar o plano; (iv) retrospecto.	(i) compreender o problema; (ii) caracterizar o problema; (iii) representar o problema; (iv) resolver o problema; (v) refletir sobre a solução; (vi) comunicar a solução do problema;	(i) compreensão do problema; (ii) concepção do programa; (iii) codificação do programa; (iv) teste do programa;

Quadro 8: Paralelo entre as heurísticas de Polya e fases sugeridas para a programação de computadores
Fonte: Autoria própria

É possível ilustrar as informações contidas no quadro 8 de forma a representar relações intrínsecas entre os trabalhos de Gomes e Mendes (2007) e Havenga (2008) com as heurísticas propostas por Polya. A figura 5 apresenta tais relações.

Percebe-se nos trabalhos de Gomes e Mendes (2007) e Havenga (2008) as etapas de compreensão do problema e de planejamento da solução antes da codificação do programa. Programadores novatos e experientes reagem de formas diferentes diante dessas etapas (Gomes e Mendes 2007): (i) programadores novatos focam atributos superficiais do problema, partindo da definição de metas para o entendimento do problema; (ii) programadores novatos tentam resolver os problemas imediatamente, reservando pouco tempo para a interpretação do mesmo; (iii) programadores novatos não procuram solucionar problemas por refinamentos sucessivos. Cabe destacar que refinamentos sucessivos é uma técnica para resolução de problemas, tendo sido proposta por Niklaus Wirth em 1971. Tal técnica trata a resolução através da divisão de um problema complexo em subproblemas, sucessivamente, até torná-lo suficientemente simples (Wirth 1971).

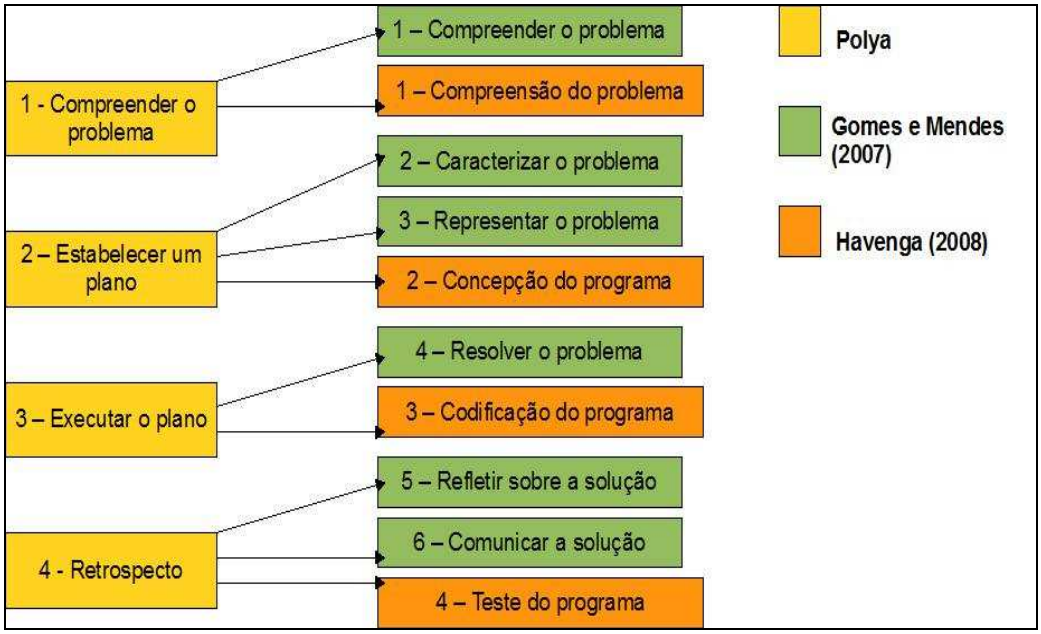


Figura 5: Relações entre as heurísticas propostas por Polya e estudos relacionados ao ensino de programação
Fonte: Autoria própria

Alunos novatos geralmente assumem uma atitude passiva ao cometer erros; deixam de realizar as atividades propostas ficando à espera da resolução apresentada pelo professor ou por colegas. Essa atitude não permite que o aluno exercite ou aprimore sua capacidade para resolução de problemas. Papert (2008) afirma ser necessário o aluno assumir o comando do seu próprio desenvolvimento. Ao assumir tal comando, o aluno tornar-se-á um aprendiz competente não dependendo de ações de terceiros para a realização de suas atividades e conseqüente aquisição do conhecimento.

Dessa forma, estimular os alunos à prática da resolução de problemas de forma consciente pode promover a dedicação necessária ao aprendizado, pois ele estará no comando desse processo visualizando os resultados positivos de suas ações.

2.4.3 Construcionismo

Seymour Papert, matemático e educador, desenvolveu uma teoria denominada por ele de “construcionismo”. O construcionismo recebeu maior

influência da teoria piagetiana. É de Piaget que Papert incorpora a imagem do indivíduo como construtor do seu próprio conhecimento e o processo de tal construção como algo incremental.

Na década de 60 o uso do computador na educação estava baseado na utilização dos *software* CAI (*computer aided instruction*). Os estudos de Papert indicam uma nova possibilidade.

O construcionismo apresenta o computador como a mídia capaz de preparar o indivíduo para atuar em uma sociedade em constante mudança, tornando-o um aprendiz competente. Todavia, para que o computador exerça sua função, o processo de ensino deve enfatizar como se dá o aprendizado. Ou seja, a escola deve buscar proporcionar aos alunos meios para que cada um crie seu próprio formato de aprendizagem e que saiba aplicá-lo sempre que se fizer necessário buscar novos conhecimentos. Assim, cabe a escola ensinar a aprender.

Papert (2008, p.89) introduz um novo conceito: matética, como sendo uma disciplina sobre a arte de aprender. A matética prevê que empregar tempo e discutir determinado problema contribui para melhor resolvê-lo e contribui igualmente para solucionar problemas semelhantes. Um fator impulsionador para a matética é a motivação. Os indivíduos buscarão novas formas de aprender e as formalizarão para que o processo seja repetido outras vezes, apenas quando estiverem motivados; quando as atividades apresentarem sentido e forem capazes de envolvê-los de forma a que percebam estarem participando de uma atividade realmente significativa. De acordo com Papert (2008), a aplicação da matética contribui para que o aprendiz passe mais tempo pensando sobre o problema a ser solucionado e não apenas aplique heurísticas sem refletir sobre o mesmo.

A linguagem de programação LOGO, proposta por Papert, visa levar os alunos à reflexão sobre seus pensamentos e ações em um ambiente computacional motivador. Tal linguagem foi desenvolvida na década de 70 e consiste em instruções que movem a imagem de uma tartaruga na tela.

Através da utilização da linguagem LOGO, é possível criar um ciclo descrição – execução – reflexão – depuração. Tal ciclo oferece ao aluno a possibilidade de reflexão sobre a resposta do computador ao executar um comando, pois o computador oferece um *feedback* fiel e imediato ao usuário. Ao refletir sobre o *feedback* o aluno poderá perceber erros e depurá-los buscando informações em

locais externos ao computador. Ao corrigir um erro o aluno altera modelos mentais previamente elaborados, ocorrendo assim um processo de aprendizagem.

Bricolagem é um termo utilizado com frequência no construcionismo para referenciar, nas palavras de Papert: use o que você tem, improvise. Ao utilizar a linguagem de programação LOGO, ou outros ambientes que ofereçam o ciclo descrição – execução – reflexão – depuração, a bricolagem pode ser exercitada. Isso acontece porque os alunos aplicam conhecimentos já adquiridos e representados por modelos mentais. Caso tais modelos mentais não sejam suficientes, eles precisarão transformá-los e para isso buscarão conhecimentos específicos necessários para tal reformulação. Nesse processo estarão aplicando os princípios básicos da bricolagem. Por meio da bricolagem, o construcionismo defende um currículo menos rígido e que não pode ser totalmente definido previamente.

A união do construcionismo com as heurísticas para resolução de problemas, considerando a importância da elaboração de modelos mentais adequados é a fundamentação das estratégias de ensino propostas neste trabalho. Os pontos-chave adotados para a elaboração dessas estratégias, bem como o papel do professor para o sucesso da aplicação das mesmas, são apresentados no capítulo 3.

3 UMA NOVA PROPOSTA PARA O ENSINO DA PROGRAMAÇÃO DE COMPUTADORES

3.1 PONTOS-CHAVE CONSIDERADOS

Nesta seção são apresentados os pontos-chave considerados para a elaboração das práticas de ensino propostas nesse trabalho. Esses pontos-chave estão fortemente fundamentados no construcionismo proposto por Seymour Papert aliado às heurísticas para resolução de problemas propostas por George Polya e a importância de elaboração de modelos mentais adequados para um aprendizado eficaz.

A aplicação do construcionismo visando melhorar o processo de aprendizagem da programação de computadores não é algo novo. Resultados positivos da aplicação dos conceitos do construcionismo para o ensino da programação de computadores podem ser verificados em Freitas (2008), Lima (2009), Rocha (2003) e Pinheiro (2003). A inovação apresentada nesse trabalho está na união explícita do construcionismo com demais conceitos, como apresentado na figura 6.

Acredita-se que modelos mentais elaborados adequadamente são capazes de auxiliar o estudante no momento da compreensão do problema a ser solucionado. Assim, faz-se necessário meios que auxiliem no processo de criação e adequação de tais modelos. O elevado nível de abstração exigido na programação é tido como uma das principais dificuldades. Assim, o primeiro ponto-chave é a elaboração de exemplos que possam ser apresentados aos alunos e discutidos, sendo posteriormente modificados pelos mesmos diretamente em linguagem de programação visando a diminuição do nível de abstração e conseqüentemente facilitando a construção de modelos mentais. O ensino da programação de computadores baseado em exemplos é foco do estudo de Edwards (2004).

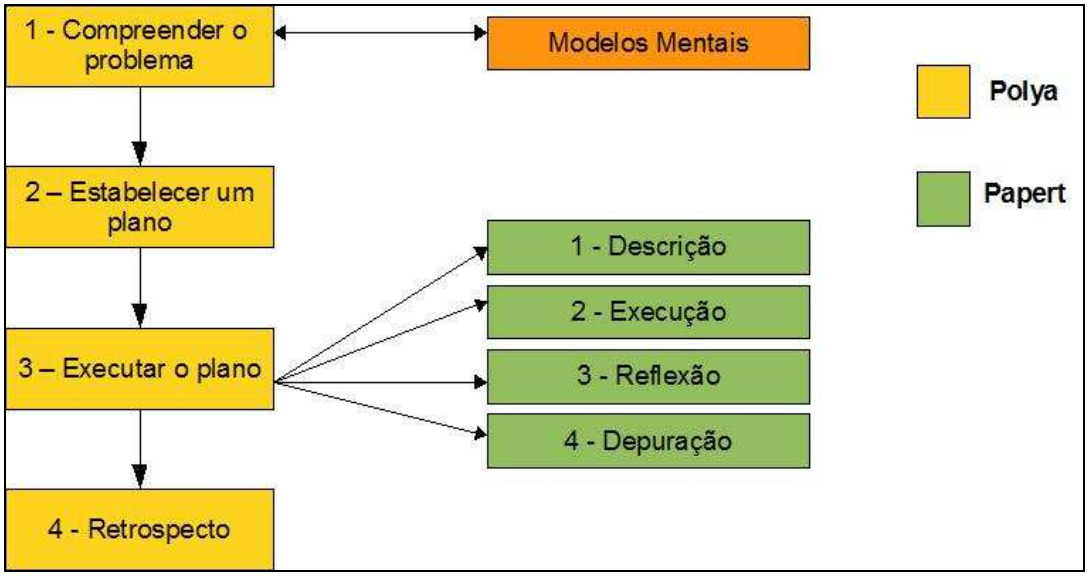


Figura 6: Relação: modelos mentais, heurísticas para resolução de problemas e o construcionismo
Fonte: Autoria própria

O aprendizado se dá através de apropriações realizadas e conseqüentes modificações em modelos mentais previamente elaborados. Visando promover tais modificações o segundo ponto-chave é a elaboração e a proposta de exercícios com níveis de dificuldades crescente. Sempre que o educando implementar a solução para exercícios propostos, ele estará executando o ciclo descrição – execução – reflexão – depuração, proposto pelo construcionismo. Assim, terá a oportunidade de refletir sobre seu pensamento inicial, onde através dessa reflexão, poderá realizar novas apropriações ou modificações em modelos mentais já elaborados.

A proposta dos exercícios é fator crucial para o sucesso das práticas de ensino. Se tais exercícios forem propostos com caráter desafiador são capazes de estimular características como a curiosidade, autonomia e responsabilidade, esse é o terceiro ponto-chave. Já o quarto ponto-chave é que os exercícios tenham como tema central assuntos de interesse dos alunos e que façam uso do lúdico, pois tais elementos exercem função motivadora.

A aplicação prática do princípio da matética “dar-se tempo” é o quinto ponto-chave, pois compreender um problema e planejar sua solução exige quantidade de tempo diferente de indivíduo para indivíduo. Alunos livres para pensar, sem determinação de um tempo restrito, podem produzir melhores resultados.

Por fim, o último ponto-chave é o incentivo ao segundo princípio da matética “uma boa discussão promove aprendizagem”. A organização dos alunos em grupos

para o desenvolvimento das atividades é um incentivo a tal princípio, pois durante esse processo conhecimentos e experiências passam a ser compartilhados.

Como já mencionado, os pontos-chave descritos nessa seção foram aplicados na elaboração das estratégias de ensino da programação de computadores, produto final desse trabalho de pesquisa. Todavia, vale ressaltar o importante papel que o professor assume diante das estratégias de ensino propostas. Esse papel é apresentado na seção 3.2.

3.2 O PAPEL DO PROFESSOR

A forma com que os professores dos conceitos iniciais da programação conduzem as atividades da disciplina exerce impacto nos resultados atingidos pelos alunos. O papel do professor é fundamental para que habilidades sejam estimuladas.

De acordo com Lima (2009, p.129),

> é necessário que o trabalho do professor de programação inclua a capacidade de promover a aprendizagem de uma forma estimulante. Isso no sentido de adequar a técnica da computação a uma dimensão acessível aos discentes.

Diferentes pesquisadores destacam o importante papel representado pelo professor, entre eles estão Dante (2005), Davis et al. (1997), Jacobson et al. (1997), Papert (2008) e Polya (1978).

Tendo o estímulo à capacidade de resolução de problemas como um dos principais objetivos, cabe ao professor dos conceitos iniciais da programação de computadores:

- Motivar os alunos

  O aluno precisa sentir-se suficientemente motivado para envolver-se no processo de resolução de determinado problema. Acredita-se que ao sugerir problemas, o professor deve considerar não apenas o currículo, mas também os interesses dos alunos. Propor atividades como desenvolvimento de programas em formato de jogos e animações é potencialmente mais atraente do que, por exemplo, trabalhar exercícios puramente numéricos.

- Estimular o trabalho independente

Para Papert o pensamento produtivo e independente é um importante fator para melhoria da capacidade dos indivíduos para resolver problemas. Cabe ao professor, ao lançar uma atividade problema, orientar sua execução contribuindo com explicações adicionais sempre que necessário. Todavia, é preciso ter cuidado para que a solução não seja revelada (Polya 1978).

Uma dose certa de auxílio é capaz de deixar nas mãos do aluno o controle do próprio aprendizado, ou como diz Papert o comando de seu próprio desenvolvimento.

- Incentivar a criatividade e o trabalho em equipe

Na vida profissional, programadores se deparam diariamente com novos problemas. Os futuros profissionais precisam ser preparados para esses momentos. Acredita-se que apenas memorizando trechos de código e trabalhando individualmente não se tornarão profissionais criativos e aptos tecnicamente para propor soluções adequadas.

Ao oferecer um ambiente rico em discussões e livre para criação, os alunos serão estimulados a de fato aprender, incorporando os conceitos fundamentais da programação em seus esquemas mentais, podendo posteriormente aplicá-los em novas situações.

- Oferecer a possibilidade da construção de modelos mentais adequados

Ben-Ari (1998) sugere que o professor apresente explicitamente um modelo mental viável ao aluno que esteja um nível abaixo do novo conteúdo que será ensinado. Quando o aluno cometer um erro ou demonstrar falta de compreensão, o professor deve assumir que o estudante tem um modelo mental não viável, mesmo que seja um tanto consistente. Nesse caso, o aluno deve ser guiado para modificar o modelo mental existente.

Modelos mentais adequados podem contribuir com o aprimoramento da capacidade de resolução de problemas.

Ao considerar o papel do professor na elaboração das práticas de ensino, isso se dá por acreditar que o sucesso de tais práticas depende da sua responsabilidade e comprometimento para com a melhoria do processo de ensino.

4 METODOLOGIA

Por meio deste trabalho buscou-se explorar diferentes contextos do ensino da programação de computadores, com o objetivo de elaborar e propor estratégias de ensino que estimulem e/ou melhorem a capacidade de resolução de problemas.

Entende-se, de acordo com Silva e Menezes (2001), uma pesquisa de natureza aplicada como sendo aquela que busca gerar novos conhecimentos buscando a solução de um problema específico. Dessa forma, classifica-se esta pesquisa como de natureza aplicada, pois objetiva gerar novos conhecimentos relacionados ao ensino e aprendizagem da programação de computadores, envolvendo a realidade local da autora.

Do ponto de vista dos objetivos, essa pesquisa classifica-se como exploratória, pois parte importante dela visou proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou a construir hipóteses (Gil, 1991). Para isto aplicaram-se revisão bibliográfica, entrevistas, questionários e observações, como será descrito a seguir. Finalmente, nesta pesquisa a análise foi predominantemente qualitativa.

A coleta de dados aconteceu por meio da observação direta e indireta (Quivy & Campenhoudt 2005).

A observação direta foi aplicada durante a execução de um projeto piloto com o objetivo de verificar o processo de utilização do ambiente de programação *Scratch*. Dados foram coletados e registrados em um diário de campo durante os encontros, que ocorreram em 2008. A mesma técnica foi aplicada durante as aulas da disciplina Lógica de Programação, durante os três primeiros bimestres do ano letivo de 2009. Estes dados foram analisados de forma qualitativa.

A observação indireta, por sua vez, foi aplicada em diferentes momentos:

- Aplicação de questionários para estudantes da disciplina introdutória da programação, de diferentes instituições, com o objetivo de melhor explorar tais contextos. Os dados coletados foram analisados de forma qualitativa e também quantitativa.
- Entrevistas realizadas com alunos participantes do projeto piloto sobre a ferramenta *Scratch*, realizado em 2008. Os dados coletados foram analisados de forma qualitativa.

- Aplicação de questionários para estudantes da disciplina de Lógica de Programação, com o objetivo de verificar o estímulo à resolução de problemas e a repercussão na motivação dos mesmos. Os dados coletados foram analisados de forma qualitativa e também quantitativa.
- Avaliação de jogos e/ou animações computacionais elaboradas pelos alunos, através da utilização do ambiente de programação *Scratch* e da linguagem de programação *Python*. Os dados coletados foram analisados de forma qualitativa e quantitativa.
- Notas bimestrais e média dos três primeiros bimestres do ano letivo de 2009 foram comparadas entre a turma participante da execução dessa pesquisa e turma não participante. Os dados coletados foram analisados de forma quantitativa.

O tratamento qualitativo dos dados aconteceu por análise de conteúdo:

> A análise de conteúdo é um conjunto de técnicas de análise das comunicações visando obter, por procedimentos sistemáticos e objetivos de descrição do conteúdo de mensagens, indicadores (quantitativos ou não) que permitem a inferência de conhecimentos relativos às condições de produção/recepção (variáveis inferidas) destas mensagens. (BARDIN, 1979, p.42)

Finalmente, a pesquisa pode ser classificada como experimental, pois foi determinado um objeto de estudo, selecionadas variáveis que seriam capazes de influenciá-lo e definidas formas de controle e de observação dos efeitos que produziram o objeto (Gil, 1991).

A pesquisa foi realizada com uma amostra de estudantes novatos em programação, conforme apresenta a seção 4.1.

4.1 A AMOSTRA

A pesquisa foi desenvolvida em uma turma de 30 alunos da disciplina Lógica de Programação, ofertada na 2ª série do curso Técnico em Informática, do Colégio Estadual Francisco Carneiro Martins, em Guarapuava – PR.

Durante um primeiro encontro com a turma, alguns dados foram coletados com os 25 alunos presentes, sendo 12 meninas e 13 meninos. Três alunos da turma

trabalhavam em tempo parcial. Sobre as motivações em relação ao curso, 21 afirmaram ter ingressado por vontade própria, no entanto, 15 afirmaram ter interesse em seguir a carreira.

5 DESENVOLVIMENTO DA PESQUISA

A pesquisa realizada pode ser dividida em 6 etapas. A figura 7 apresenta tais etapas e as atividades realizadas em cada uma delas.

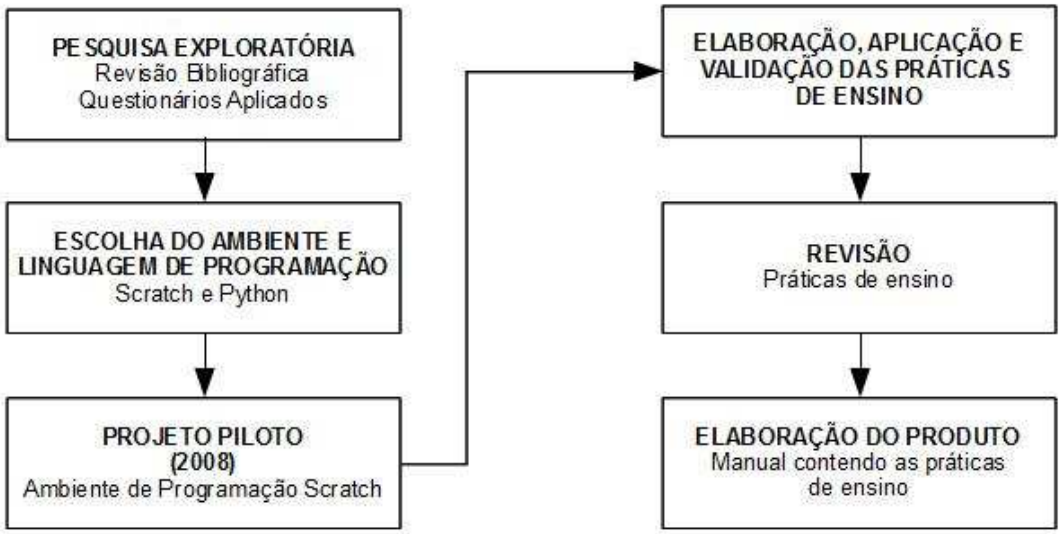


Figura 7: Atividades realizadas para o desenvolvimento da pesquisa
Fonte: Autoria própria

Com o intuito de melhor conhecer como se dá o processo de ensino e aprendizagem dos conceitos iniciais da programação de computadores, foram aplicados questionários (apêndice A) a estudantes de diferentes instituições de ensino.

Alguns pontos-chave foram identificados como prováveis fatores positivos para o bom desempenho na disciplina. Esses fatores passaram a ser considerados na elaboração das práticas de ensino a serem propostas. A seguir esses itens serão brevemente descritos.

É preciso deslocar a ênfase da construção de algoritmos, ou do ensino de uma linguagem de programação voltada para a indústria, para a resolução de problemas. Nesse sentido, optar por linguagens e ambientes de programação pedagogicamente adequados pode auxiliar o desenvolvimento do raciocínio do aluno novato em programação visando à resolução de problemas.

Percebeu-se também que os alunos precisam ser estimulados a resolver problemas. Assim, faz-se necessário destacar a importância do real entendimento

do problema assim como o devido planejamento de sua solução antes da implementação em si. Para facilitar esse estímulo, é interessante que a resolução de problemas apresente sentido aos alunos. Os problemas a serem propostos devem estar aliados aos interesses do indivíduo. Ao agir por seu próprio interesse o aluno sente-se motivado a buscar a resolução dos problemas, inclusive dedicando mais tempo a essa atividade (Maloney 2008).

Diminuir o nível de abstração exigido durante o processo de ensino da programação também apresentou-se como necessário. Nesse caso, a utilização de *software*s, especialmente de visualização, podem favorecer o entendimento das estruturas básicas da programação de computadores. Pesquisas indicam tal observação, como em Naps et al. (2003) e Moskal et al. (2000).

Após a aplicação dos questionários e a análise dos dados coletados, buscou-se uma linguagem e um ambiente de programação que possibilitasse a implementação dos pontos-chave na proposta das estratégias pedagógicas. Para auxiliar essa busca, recorreu-se a trabalhos de Goosen (2004), Mclver e Conway (1996) e Parker et al. (2000), que discutem características desejáveis em linguagens e ambientes de programação para o ensino, como: simplicidade de sintaxe; simplicidade de semântica; estruturas de linguagem de fácil entendimento; fácil diagnóstico de erros; fácil instalação e linguagem e ambiente que apóiam a resolução de problemas.

Uma pesquisa na literatura correlata foi realizada buscando ferramentas voltadas ao ensino da programação de computadores. Essa fase resultou na escolha do ambiente visual de programação *Scratch* e na linguagem de programação *Python*.

O ambiente de programação *Scratch* foi lançado em 2007 como uma ferramenta voltada ao ensino de programação. Esse ambiente está fortemente apoiado nas idéias do *software* LOGO, proposto por Papert, porém sem a necessidade de digitação de comandos, pois a programação tem por fundamento arrastar e soltar blocos de comandos. O ambiente *Scratch* diminui o nível de abstração exigido no processo de aprendizagem da programação de computadores, bem como permite a criação de programas aliados aos interesses dos alunos, pois oferece a possibilidade de inserção de imagens, sons e animações. Como *Python*, em *Scratch* a preocupação com sintaxe é reduzida e é oferecida aos alunos a

possibilidade da reflexão diante de suas escolhas. A figura 8 apresenta a tela inicial do ambiente de programação *Scratch*.

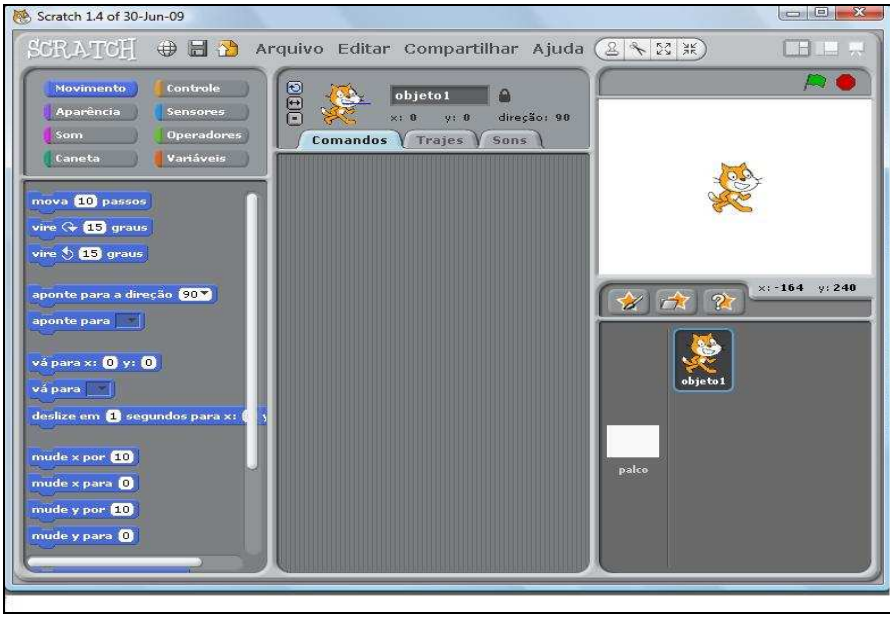


Figura 8: Tela inicial do ambiente de programação *Scratch*
Fonte: Autoria própria

A linguagem de programação *Python* surgiu no final da década de 1980, projetada por Guido Van Rossum, na Holanda. Sua simplicidade permite ao aluno novato em programação canalizar sua preocupação na resolução de problemas em si e não em regras de sintaxe, contando com uma comunidade com discussões ativas na *Internet*.

O interpretador da linguagem é de fácil instalação. Há versões para as plataformas *Windows* e *Linux*.

Python e *Scratch* estão em sintonia com os princípios do construcionismo, pois oferecem o ciclo descrição – execução – reflexão – depuração. Oferecem oportunidade para aplicação da matética e da bricolagem. Permitem aos professores criar situações de aprendizagem concretas, fugindo assim do abstrato.

Após a definição das linguagens e ambiente de programação, optou-se por realizar um projeto piloto com a ferramenta *Scratch*, para verificar o comportamento dos alunos e testar um roteiro preliminar de atividades.

Essa etapa aconteceu durante o quarto bimestre do ano letivo de 2008, duas horas/aulas semanais, em contra turno, por um período de 6 semanas. Participaram 15 estudantes matriculados na segunda série do curso técnico em informática,

cursando regularmente a disciplina de lógica de programação. Os alunos foram organizados em duplas e estimulados a compartilhar o conhecimento adquirido com os demais colegas, visando o princípio da matética segundo o qual uma boa discussão favorece a aprendizagem. Um roteiro de aula utilizando o ambiente de programação *Scratch* foi preparado com base em Cristovão (2008 b) e MIT (2008).

A próxima etapa da pesquisa foi a elaboração dos exemplos e atividades e sua formalização em roteiros de aula que compõem as práticas de ensino propostas em um manual, produto deste trabalho. Tais roteiros contemplam as aulas do primeiro ao terceiro bimestre do ano letivo de 2009.

A elaboração dos exemplos e atividades fundamentou-se no construcionismo proposto de Papert e considerou-se também a classificação sobre as fases interativas para aprendizagem propostas por Glaser (1996) apud Tucker (2003): a primeira é a dependência de apoio externo, que acontece durante as fases iniciais do processo de aprendizado; a segunda fase é a de transição entre a dependência e o auto-regulação; e a terceira fase denominada auto-regulação, onde o aluno, como perito, tem o controle de sua própria aprendizagem.

Dessa forma, os exemplos têm por objetivo ilustrar determinadas situações. Com a aplicação de exemplos prontos, espera-se diminuir o grau das abstrações presentes no processo de ensino e guiar os alunos enquanto estiverem na fase de dependência de apoio externo. Já as atividades estão divididas em exercícios e desafios. A realização dos exercícios visa ampliar o que foi exposto com exemplos, aumentando a dificuldade. Já os desafios são apresentados para motivar os alunos a passar mais tempo programando, pois devem ser realizados extra-classe. Os desafios são propostas de pequenos jogos onde o aluno deverá utilizar estruturas apresentadas nos exemplos e aplicar as estratégias que lhe foram apresentadas, como os conceitos de matética e de bricolagem, as heurísticas para solução de problemas de Polya ou refinamentos sucessivos. Espera-se que com os exercícios e desafios os alunos sejam guiados durante a fase de transição entre a dependência de apoio externo e a auto-regulação.

Estratégias cognitivas nos ajudam a lembrar, selecionar e organizar as informações dentro da memória (Havenga, 2008). Estratégias de repetição utilizam diferentes técnicas para apoiar atividades da mesma, envolvem manter informações ativas e apoiar os alunos na seleção de informações importantes. Quando os estudantes usam estruturas específicas de programação repetidamente como

While.. do, a sintaxe e o funcionamento de tais estruturas é lembrada mais facilmente. O mesmo acontece se os alunos leem muitas vezes os enunciados para descobrir elementos, esse processo se torna mais fácil com a repetição. Algumas práticas para a implementação das estratégias de repetição citadas por Havenga são: focar a atenção, a prática distribuída e usar lembretes:

- Focar a atenção: refere-se ao direcionamento e manutenção da concentração do programador na resolução de problemas;
- Prática distribuída: refere-se ao aprendizado por seções, separadas pelo tempo. Os alunos devem programar em sessões com intervalos razoáveis para otimizar a aprendizagem e reforçar a sua aplicação de habilidades de programação em diversos contextos;
- Usar lembretes: refere-se a utilização de lembretes como memória externa. Eles ajudam a aumentar a probabilidade de que as pessoas recordarão informações específicas.

As estratégias de ensino elaboradas pela autora como produto deste trabalho de pesquisa consideram as estratégias cognitivas de repetição apresentadas por Havenga (2008): a organização dos exemplos e exercícios tem por objetivo que tão logo novos conceitos sejam introduzidos, exercícios sejam realizados para fixação dos mesmos. Dessa forma, espera-se que o que está na memória de trabalho seja assimilado permanentemente. Os manuais a serem entregues aos alunos têm espaços para observações no formato de lembretes, sendo o estudante motivado a manter a atenção na resolução do problema.

Durante a aplicação das práticas de ensino propostas várias evidências foram coletadas para o momento da validação das mesmas. As primeiras evidências foram coletadas durante 10h/a onde os alunos, em equipes, desenvolveram um jogo ou animação computacional, utilizando as estruturas e os conceitos estudados em aulas anteriores, no ambiente de programação *Scratch*. Na sequência evidências foram coletadas na transição do uso da ferramenta *Scratch* para a linguagem de programação *Python*. Tais evidências foram coletadas mediante observação e questionário aplicado, ver apêndice C. Evidências também foram coletadas durante 10h/a onde os alunos, em equipes, desenvolveram um jogo de perguntas e respostas, utilizando a linguagem de programação *Python*.

Após a validação das práticas de ensino, correções necessárias foram realizadas, a fim de cumprir com o objetivo de propor uma práticas capazes de

amenizar as dificuldades apresentadas por alunos novatos em programação de computadores.

A última atividade consistiu na formalização das práticas de ensino propostas através da elaboração de um manual, através do qual o professor poderá orientar os alunos na prática de resolução de problemas, por meio da elaboração de jogos computacionais, utilizando a linguagem *Python* e o ambiente de programação *Scratch*.

6 ANÁLISE E DISCUSSÃO DOS RESULTADOS

6.1 PESQUISA EXPLORATÓRIA

Paralelamente ao levantamento bibliográfico uma pesquisa foi realizada para verificar dificuldades apresentadas pelos alunos e é apresentada no apêndice A. Questionários foram distribuídos em instituições públicas e particulares do ensino médio e graduação. De 200 questionários enviados foram retornados 137 questionários respondidos, dos quais 55% correspondem a respostas de alunos da graduação de instituições particulares (três diferentes instituições), 32% de alunos do ensino médio subseqüente em Informática (duas diferentes instituições) e 13% de alunos da graduação de instituições estaduais de ensino (uma instituição).

A pesquisa indicou que a representação de algoritmos mais utilizada nas instituições pesquisadas é o pseudocódigo: 89% dos alunos do ensino médio e 64% dos alunos da graduação em instituições particulares afirmam o pseudocódigo ter sido técnica ensinada e bastante usada. A instituição estadual revelou dados diferentes, 72% indicam que a técnica de fluxogramas foi ensinada e utilizada, 72% também indicam que o pseudocódigo foi técnica ensinada e pouco usada.

Quando questionados sobre a frequência das aulas em laboratório, 100% dos alunos da graduação estadual e 69% da particular afirmaram que houveram aulas práticas desde o início da disciplina; outros 31% da graduação particular afirmaram que as aulas práticas foram precedidas de teoria sobre a linguagem de programação Pascal e desenvolvimento de algoritmos em pseudocódigo. Já a realidade do curso médio subseqüente é diferente: 70% dos alunos afirmaram não ter ido ao laboratório durante as aulas. Essa discrepância entre as instituições pode ocorrer devido à estrutura de laboratórios de informática das instituições. Alguns alunos, do ensino médio subseqüente, deixaram como observação em seus questionários que o professor levou um microcomputador para a sala de aula e demonstrou o funcionamento de um *software*.

A linguagem de programação citada em todos os questionários foi Pascal, tendo sido mencionadas as ferramentas PascalZim, FreePascal e Delphi. As ferramentas Visualg e Múltipla também foram mencionadas. A primeira permite a edição e execução de pseudocódigo. A segunda permite a construção de fluxogramas e os traduz de forma automática para a linguagem Pascal. Ferramentas

não específicas para o ensino de programação de computadores também foram citadas: o ambiente virtual de aprendizagem Moodle e jogos elaborados em flash.

Enquanto alguns estudantes acabam por não ter contato com sistemas computacionais durante o aprendizado dos conceitos iniciais da programação, outros chegam a criar programas em ambientes voltados ao desenvolvimento de *software*s comerciais, como é o caso do ambiente de programação Delphi. Entre esses dois extremos verificamos ainda a utilização de ferramentas desenvolvidas com objetivos pedagógicos como o VisuAlg e o Múltipla.

A pesquisa também indica diferença no perfil dos estudantes de informática, enquanto 85% dos alunos da graduação estadual e 63% das instituições particulares dispunham de tempo para rever conteúdo extra-classe, 59% dos alunos do curso técnico afirmam não dispor de tempo para estudos fora da escola.

Mesmo em ambientes marcados por diferenças na estrutura física das instituições, perfil dos alunos e ferramentas utilizadas para o ensino, os questionários mostraram que os problemas computacionais propostos aos alunos são muito próximos. Alguns exemplos: construir um programa que apresente a tabuada de um número qualquer ou, construir um programa, ou algoritmo, para cadastro de livros em uma biblioteca.

Os questionários pediram para classificar em uma escala dada o nível de dificuldade para uma série de itens: interpretar o enunciado e assim entender qual o real problema a ser solucionado; dificuldades em relação à sintaxe; falta de tempo para estudar; planejar previamente a solução para o problema e organizar logicamente os passos necessários para a solução; necessidade de prática constante; necessidade de implementação em laboratório para visualização do resultado.

Um total de 89% dos estudantes da instituição estadual e 43% das instituições particulares afirmaram achar a sintaxe fácil; no ensino técnico 48% afirmaram ser um pouco difícil. Quanto ao entendimento do enunciado dos problemas propostos, 56% da instituição estadual, 53% das instituições particulares e 52% do ensino técnico, afirmaram ser um pouco difícil tal compreensão. Sobre o planejamento antes de propor a solução a um problema, 50% dos alunos da instituição estadual, 41% das instituições particulares e 52% do ensino técnico afirmaram ser um pouco difícil tal planejamento. Em nossa experiência essa dificuldade pode motivar o não planejamento de soluções; de fato, 89% dos alunos

da instituição estadual afirmaram não planejar a solução e testá-la antes de implementá-la computacionalmente ou através da construção de algoritmos, 34% dos alunos das instituições particulares e 48% do ensino técnico afirmam o mesmo.

Ao considerar as dificuldades apresentadas pelos alunos em compreender o problema a ser solucionado e planejar tal solução, acredita-se que ao oferecer mecanismos para a elaboração de modelos mentais adequados do computador e do *software* a ser desenvolvido, bem como explorar técnicas de resolução de problemas durante as aulas, os professores dos conceitos iniciais da programação estarão possibilitando aos alunos maiores chances de uma solução adequada ao problema proposto.

6.2 PROJETO PILOTO COM O AMBIENTE DE PROGRAMAÇÃO *SCRATCH*

Depois de selecionadas linguagem e ambiente de programação que fariam parte das práticas de ensino, produto deste trabalho de pesquisa, optou-se por realizar um projeto piloto com o ambiente de programação *Scratch*. Essa ferramenta não havia sido usada anteriormente pela instituição ou pela proponente. Assim, procurou-se verificar o comportamento dos estudantes e testar um roteiro de aulas.

Este projeto piloto aconteceu durante o quarto bimestre do ano letivo de 2008, com 15 estudantes. Dessa forma, os alunos participantes já conheciam os conceitos de variáveis, estruturas de seleção, estruturas de repetição, comandos de entrada, saída e de atribuição. Em um primeiro momento procurou-se deixá-los livres para produzir; assim, em cada encontro o material da aula era entregue e os alunos deveriam estudar os exemplos e realizar as atividades propostas, com o acompanhamento da professora. Essa postura seguiu a linha de Papert sobre o aluno exercer o comando de seu próprio aprendizado.

Algumas atividades propostas aos alunos já haviam sido trabalhadas em sala com pseudocódigo, bastando aos alunos implementá-las em *Scratch*. Percebeu-se, no entanto, que os alunos viam-se sem saber por onde começar a solução de um problema, mesmo já o tendo solucionado em outro momento. Eles apresentavam dificuldades pelo motivo do problema ser apresentado em um contexto diferente; não conseguiam adaptar um conhecimento prévio para novas situações.

Durante a realização do projeto piloto dados foram coletados através da observação direta e também de observação indireta, através da realização de entrevistas com uma amostra de 7 alunos participantes.

A observação mostrou que a utilização do ambiente de programação *Scratch* torna o aprendizado mais dinâmico, divertido e motivador. Durante os encontros do projeto os alunos demonstraram muito interesse e vontade de continuar com a utilização da ferramenta, mesmo após o encerramento do projeto. Eles estavam visivelmente mais motivados para trabalhar em laboratório, desenvolvendo animações computacionais e pequenos jogos, do que em sala de aula desenvolvendo algoritmos apenas com lápis e papel.

Perceberam-se também duas posturas antagônicas: alguns alunos demonstraram interesse em avançar na complexidade das atividades sugeridas, criando animações mais arrojadas que aquelas previamente solicitadas, enquanto outros estavam muito preocupados com uma possível avaliação sobre o *Scratch*.

Apenas a utilização da ferramenta não foi suficiente para modificar a atitude em relação às dificuldades para solucionar determinados problemas computacionais. Alguns alunos, ao se depararem com a dificuldade de planejamento para a solução do problema, desistiram. Ou ainda, tendo solucionado problema semelhante não foram capazes de buscar nessas soluções prévias embasamento para a solução do novo problema. Todos esses indícios reforçam a conclusão de que o cerne das dificuldades apresentadas pelos alunos novatos parece estar na baixa capacidade para resolução de problemas. Tais alunos tem dificuldades para transferir conhecimentos adquiridos na resolução de problemas para outros contextos.

As entrevistas realizadas ao término do projeto piloto indicam que o ensino dos conceitos iniciais da programação de computadores, baseado em linguagens interpretadas pode favorecer a aprendizagem o *feedback* fiel e imediato exige dos estudantes maior planejamento e/ou correções durante a implementação do programa, assim como contribui para diminuição das abstrações presentes nesse processo.

Foi revelado também que o ambiente de programação *Scratch* é capaz de tornar a programação fácil por não apresentar regras complexas de sintaxe. Sua utilização também é capaz de tornar a programação divertida, já que os alunos podem inserir em seus programas elementos como sons e imagens.

A utilização de linguagens interpretadas como apoio ao ensino dos conceitos iniciais da programação de computadores, apesar de útil no que se refere à diminuição das abstrações, não é o fundamental. A fonte das dificuldades apresentadas pelos alunos, de acordo com as entrevistas e a observação realizadas, está no fato dos alunos não conseguirem solucionar de forma adequada um problema computacional. Isso acontece por não saberem interpretar o enunciado do problema e deles extrair as informações necessárias. Segundo alguns alunos, por não entender o que é realmente o problema a ser solucionado, o planejamento ou não é feito ou é feito de forma inadequada. Todavia, alguns alunos reconheceram, expressando verbalmente, a importância da utilização de linguagens interpretadas.

6.3 ROTEIROS DE AULA

Nesta seção será apresentada a aplicação de uma série de roteiros de aula, concebidos a partir do projeto piloto e de todas as análises realizadas. Esses roteiros foram organizados visando atender ao contexto do trabalho, o que significa adequação à disciplina ministrada e a faixa etária dos alunos. Contudo o professor interessado poderá adaptar este material para seu próprio uso. Os roteiros estão dispostos em uma sequência crescente de dificuldade e cobrem as aulas da disciplina desde o primeiro contato dos alunos com a mesma.

Os roteiros foram divididos em blocos de assuntos correlacionados e são apresentados a seguir.

6.3.1 Roteiros de Aula I até Roteiro de Aula V

As atividades previstas para a introdução da disciplina foram distribuídas em cinco roteiros de aula, totalizando 14 h/a. Nesta seção, são apresentados resultados obtidos durante a implementação de tais roteiros.

O **Roteiro de Aula I** contem as atividades relacionadas à apresentação da disciplina de programação de computadores e seus conceitos fundamentais, de forma a possibilitar aos alunos a elaboração de modelos mentais adequados sobre ela.

Dentre as atividades realizadas durante o Roteiro de Aula I, na aula quatro, pequenos programas criados na linguagem de programação *Python* foram apresentados aos alunos por meio de um projetor multimídia. Conceitos da programação de computadores foram explicados e ilustrados diretamente no código apresentado. Em seguida, os alunos foram organizados em equipes. Foi solicitado que a organização em equipes fosse mantida na sequência das aulas.

Na aula seguinte, aula cinco, uma lista de exercícios foi entregue aos alunos. Divididos em equipes, eles deveriam analisar trechos em *Python* e alterá-los com o objetivo de gerar novos programas, conforme solicitado. O quadro 9 apresenta parte da lista de exercícios mencionada.

Código apresentado (em linguagem de programação *Python*)	Proposta para alteração de códigos
#Programa para informar a média final das notas de um aluno soma = 0 n = 1 while n<=4: nota=float(raw_input('Digite a nota:')) soma=soma+nota n=n+1 media=soma/4 print media	a) Altere o código de forma a que o programa informe se o aluno está aprovado ou reprovado. Considere média 6.0 para aprovação. (b) Altere o código de forma a que o programa leia as quatro notas bimestrais de 30 alunos e informe se cada aluno está aprovado ou reprovado. Considere média 6.0 para aprovação.

Quadro 9: Parte da lista de exercícios entregue aos alunos durante a aula 4
Fonte: Autoria própria

Através da observação realizada durante a execução da lista de exercícios, foi possível perceber que, mesmo sem a manipulação direta da linguagem de programação *Python*, os alunos, em sua maioria, sentiram-se à vontade para realização de tal atividade. As equipes demonstraram, nas discussões em grupo, entendimento sobre a função das estruturas básicas: variáveis, comandos de entrada e saída, comandos de seleção e repetição. Tal observação foi confirmada com a entrega da lista de exercícios e posterior correção, onde os resultados foram tidos como positivos. A correção dos exercícios mostrou também que algumas considerações eram necessárias e deveriam ser retomadas junto aos alunos: (i) a falta de atenção relacionada à identação do código em *Python*, exigida pela linguagem; (ii) o uso de estruturas de repetição aninhadas, necessárias para a solução de determinados problemas computacionais. Vale ressaltar que o

tratamento de blocos de comandos em *Python*, usando identação, é mais simples que usando marcas como "*begin-end*" de Pascal ou chaves de C.

As atividades realizadas durante as aulas 4 e 5 foram suficientes para apresentar de forma dinâmica e envolvente conceitos fundamentais da programação. Houve a diminuição do grau de abstração exigido no processo e os alunos mostraram-se motivados e compartilharam conhecimentos em equipe. É possível inferir que essas atividades foram suficientes para elaboração de modelos mentais adequados sobre a função das estruturas básicas. Todavia, nesse momento, os alunos ainda não adquiriram familiaridade com o uso prático e a manipulação dessas estruturas.

As heurísticas para resolução de problemas propostas por Polya, e discutidas na seção 2.4.2. desta pesquisa, são o tema do **Roteiro de Aula II**. Durante 02 h/a atividades foram desenvolvidas com o objetivo de apresentá-las aos alunos.

Primeiramente, uma sequência de imagens foi apresentada, ilustrando um problema a ser solucionado. Em resumo, o problema consistia em um indivíduo que precisava atravessar um penhasco. A solução criada foi apresentada aos alunos, sendo destacado cada fase em que as heurísticas de Polya foram aplicadas, ressaltando a importância de "pensar" sobre o problema, dispensando tempo suficiente para essa fase. Outros exemplos também foram utilizados.

As heurísticas para resolução de problemas foram entregues aos alunos juntamente com questionamentos que auxiliam na sua aplicação. Na sequência três novos problemas foram entregues às equipes para resolverem.

Em geral, as equipes se mostraram confusas no momento da aplicação prática das heurísticas. Quando questionados sobre as dificuldades, as equipes afirmaram ter entendido as heurísticas isoladamente, mas que na prática, "tudo" ficava confuso. Assim, o conteúdo foi retomado outras vezes com o objetivo de favorecer o entendimento.

Pode-se supor que a dificuldade apresentada pelos estudantes esteja relacionada à baixa capacidade em resolução de problemas: alguns alunos não costumam pensar sobre o processo e refletir sobre as fases realizadas. Outros, todavia, possivelmente não foram suficientemente estimulados em séries anteriores e, ao se deparar com a situação de resolução de problemas, mesmo tendo as heurísticas como referencial, não sabiam sequer começar. Esse fato ocorrido durante o roteiro de aula II confirma pesquisas que indicam que os alunos com

frequência chegam às disciplinas de programação sem o devido estímulo e capacidade para resolução de problemas.

O **Roteiro de Aula III**, por sua vez, contem atividades relacionadas à apresentação de conceitos do construcionismo: matética e bricolagem. Optou-se por inserir tal conteúdo nos roteiros de aula com o objetivo de discutir com os estudantes a importância de vários aspectos, como “pensar” sobre o problema a ser solucionado, o trabalho em equipe, o estímulo à criatividade e a busca por conhecimentos anteriores para aplicação em novos problemas. Após uma discussão foi solicitado às equipes o desenvolvimento de uma história em quadrinhos tendo como tema central a resolução de problemas. Ao final da atividade, o resultado foi tido como positivo. As equipes demonstraram entendimento sobre o que é um problema e houve reforço das heurísticas propostas por Polya, elas estavam direta ou indiretamente presentes em 80% das histórias em quadrinhos entregues.

O **Roteiro de Aula IV** traz a técnica de refinamentos sucessivos para resolução de problemas. Dois exemplos foram preparados para a explicação da técnica, no entanto, devido ao questionamento feito por um aluno, a sequência da aula foi alterada. O quadro 10 apresenta o primeiro exemplo utilizado, o questionamento realizado e o novo exemplo elaborado.

Exemplo 1	Questionamento realizado	Novo exemplo elaborado
Problema: Programa para ler dois valores inteiros para variáveis A e B. E apresentar esses valores ordenados de forma crescente.	E se o problema fosse ler as notas de todos os alunos da nossa sala, em lógica de programação, e apresentá-las em ordem crescente?	*Problema*: Programa para ler as notas na disciplina de Lógica de Programação, de 30 alunos, e apresentá-las de forma ordenada, da menor para a maior.

Quadro 10: Exemplos para explicação da técnica de refinamentos sucessivos
Fonte: Autoria própria

Partindo do questionamento feito, um novo exemplo foi elaborado. Foi possível introduzir o conceito de vetores aos alunos; tendo em vista já terem tido contato com estruturas de repetição, a compreensão do funcionamento de tal estrutura tornou-se menos abstrato e aparentemente os alunos demonstraram entendimento sobre o assunto.

O questionamento feito por um dos estudantes demonstrou a atenção e o interesse no problema apresentado. Isso foi visto como algo positivo, pois indicou que as aulas introdutórias, mesmo distante da prática em laboratório, foram suficientes para despertar a motivação dos estudantes.

O **Roteiro de Aula V** proporciona uma síntese integradora dos conteúdos trabalhados anteriormente. Em laboratório de informática os estudantes receberam links para sites com testes lógicos. A aula foi descontraída e os alunos foram muito participativos. A observação realizada sobre essa atividade mostrou resultados positivos em relação à resolução de problemas: (i) os alunos, em sua maioria, planejaram a solução para os testes; (ii) em geral buscaram aplicar na solução do segundo problema os conhecimentos adquiridos para solucionar o primeiro. Tais atitudes demonstram a utilização por parte dos alunos, das heurísticas para resolução de problemas propostas por Polya.

6.3.2 Roteiro de Aula VI

A partir do **Roteiro de Aula VI** os alunos passaram a desenvolver atividades práticas em laboratório de informática, com o uso do ambiente de programação *Scratch*. Tal roteiro emprega 10 h/a.

Primeiramente, um manual contendo as atividades a serem realizadas em *Scratch* foi entregue às equipes. Esse manual, como já mencionado, contém exemplos e propostas de atividades, no formato de exercícios e desafios. Os alunos foram orientados a seguir todas as atividades propostas, iniciando pela implementação dos exemplos, realização dos exercícios e, por último, desenvolvimento dos desafios. Essas atividades foram propostas seguindo uma ordem didática, visando o aprendizado real dos conceitos apresentados.

Alguns grupos realizaram a sequência correta proposta, outros, todavia, não implementaram os exemplos passando diretamente aos exercícios e outros ainda, ficaram entretidos criando animações conforme vinham idéias em suas mentes, esquecendo por completo os exemplos e as atividades. Em um primeiro momento, foi permitido que os alunos que não estavam seguindo o manual permanecessem explorando a ferramenta e estimulando sua criatividade, porém, a partir da terceira aula passaram a ser conduzidos a seguir a sequência estabelecida.

A figura 9 apresenta alunos em laboratório de informática utilizando a ferramenta *Scratch*.

Figura 9: Alunos utilizando a ferramenta *Scratch*
Fonte: Autoria própria

O ponto negativo percebido durante as primeiras aulas deste roteiro foi o não planejamento para resolução dos problemas propostos. Aproximadamente 70% das equipes estavam programando por tentativa, apenas arrastando blocos de comando sem raciocinar sobre suas funções e o impacto desta ação no código que estava sendo criado. Foi necessário retomar a importância de um bom planejamento para programação de computadores. Após o reforço por parte da professora, houve a tentativa de planejamento das soluções.

Ponto positivo percebido durante a execução do **Roteiro de Aula VI** foi o interesse apresentado pelos estudantes. Durante as aulas, não houve a realização de atividades paralelas, ou seja, a *Internet* e *software*s para edição de som e imagem foram utilizados somente para localização e/ou edição de objetos a serem utilizados para realização das atividades propostas. Não houve o uso de outros programas, tão pouco realização de atividades de outras disciplinas durante as aulas pertencentes a esse roteiro. A motivação mostrou-se clara, ao ponto de alunos não participativos em sala de aula, realizarem de forma entusiasmada e criativa todas as atividades propostas em laboratório de informática.

Acredita-se que os exemplos apresentados foram suficientes para elaboração de modelos mentais adequados em relação às estruturas básicas da programação.

Já os exercícios propostos através de desafios lúdicos, cumpriram seu objetivo de estimular a curiosidade, a responsabilidade e a autonomia dos estudantes.

O trabalho em equipe nessa fase foi satisfatório, os alunos discutiram e aqueles com maior facilidade de aprendizado auxiliaram os demais.

6.3.3 Roteiro de Aula VII

Após a implementação das atividades propostas no roteiro de aula VI, os alunos passaram a desenvolver, ainda em equipes e durante 10 h/a, um *software* em *Scratch*. Essa atividade compõem o **Roteiro de Aula VII**. O *software* a ser desenvolvido poderia ser um jogo ou uma animação computacional.

Durante a execução da atividade proposta no roteiro de aula VII, foi possível observar que a distribuição dos exemplos e atividades propostas no manual entregue no roteiro de aula anterior, foi suficiente para apresentar as principais estruturas e recursos do ambiente de programação *Scratch*.

A curiosidade e autonomia fizeram-se claras ao considerar que as equipes exploraram tal ambiente sem recorrer à ajuda da professora, buscando recursos além daqueles apresentados no manual.

O interesse e motivação por parte dos alunos continuaram sendo percebidos nessa fase. Além da dedicação em ambiente escolar, houve programação extra-classe. Ao menos um aluno de cada equipe instalou o ambiente de programação em computadores fora da escola, onde a atividade proposta foi, por muitas vezes, desenvolvida.

Os *software*s entregues foram considerados satisfatórios na maioria dos casos. Foi possível observar a compreensão dos principais recursos do ambiente de programação *Scratch*, bem como o entendimento sobre estruturas de repetição em 60% dos *software*s entregues, estruturas de seleção 80% e variáveis, em 80% dos *software*s entregues.

O gráfico 3 apresenta as notas obtidas pelas equipes conforme a entrega do *software*. Os critérios para avaliação quantitativa desse *software* estão no apêndice B.

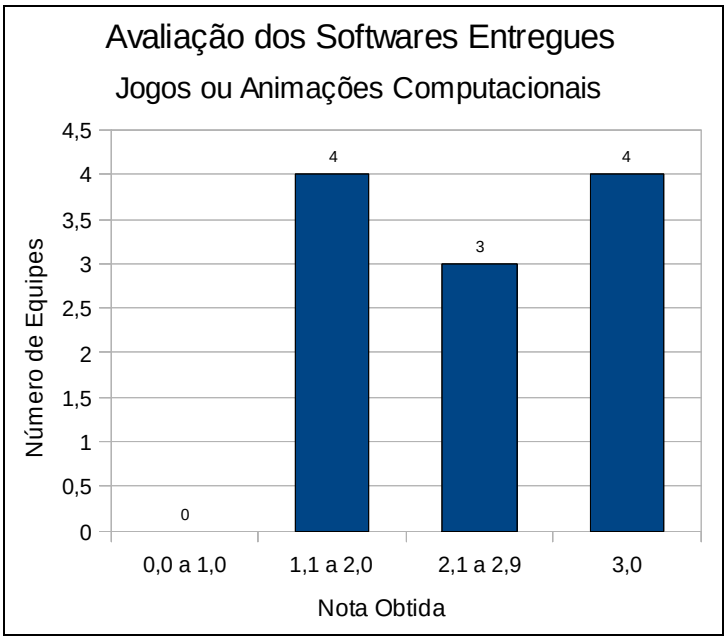


Gráfico 3: Notas obtidas pelas equipes conforme a entrega do *software*
Fonte: Autoria própria

O gráfico 4 apresenta as notas obtidas pelos alunos em avaliação oral, e individual, sobre o *software* entregue. Durante a avaliação oral questionamentos foram realizados conforme o código do *software* apresentado; eles tinham por objetivo verificar o entendimento sobre estruturas de repetição, seleção e variáveis. Durante tal avaliação a maioria dos alunos demonstrou entendimento dos conceitos fundamentais da programação, obtendo a nota máxima.

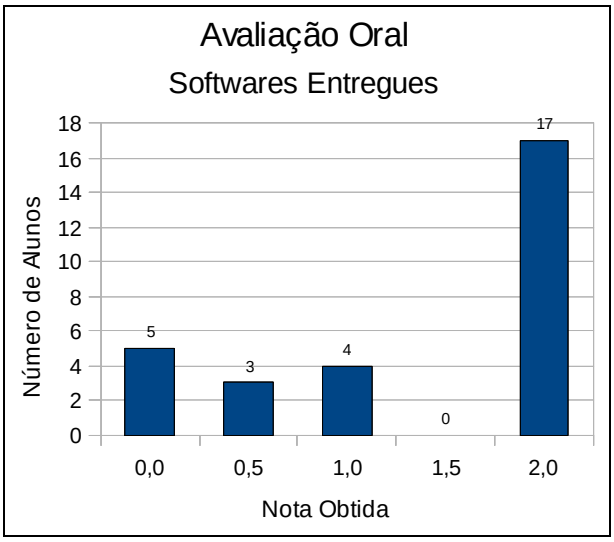


Gráfico 4: Notas obtidas em avaliação oral e individual
Fonte: Autoria própria

Acredita-se que a capacidade de resolver problemas foi estimulada nesse roteiro de aula, pois o desafio foi lançado e os alunos procuraram resolvê-lo com entusiasmo e autonomia, buscando conceitos previamente utilizados na proposta das atividades do manual entregue, além de recursos próprios da ferramenta. Infelizmente, a avaliação oral mostra que alguns alunos não acompanharam o desenvolvimento do *software*, ou mesmo acompanhando nada entenderam. Foram 17% dos estudantes que não obtiveram nota alguma em tal avaliação. Existe a possibilidade de que estes estudantes estejam na primeira categoria de estudantes de programação identificada por Dunican (2002), ou seja, aqueles alunos que não têm aptidão para compreensão dos conceitos básicos.

6.3.4 Questionário aplicado aos alunos participantes da pesquisa

Após o término das aulas referentes ao roteiro de aula VII, um questionário foi aplicado com o objetivo de verificar a motivação dos alunos em relação ao ambiente de programação *Scratch* e o estímulo à resolução de problemas. As mesmas questões foram adaptadas e aplicadas aos alunos de outra turma de lógica de programação, não participante da aplicação dos roteiros de aula propostos neste trabalho. Nesta seção, a turma participante será chamada Turma P e a turma não participante será chamada Turma NP.

O questionário aplicado encontra-se no apêndice C e foi respondido por 31 estudantes da Turma P e 25 alunos da Turma NP.

Com o objetivo de verificar a aplicação prática das heurísticas propostas por Polya, os alunos foram questionados sobre o que foi realizado antes de criar seus programas ou algoritmos. Os dados são apresentados na tabela 1.

Tabela 1: Aplicação prática das heurísticas propostas por Polya

Heurísticas	Turma P	Turma NP
Compreender o problema antes de resolvê-lo	94%	92%
Após compreensão, buscar por problema semelhante já resolvido	68%	88%
Planejar a solução e testá-la antes da programação em si	48%	60%
Rever a solução elaborada	74%	56%

Fonte: Autoria própria

Os dados coletados nas duas turmas, sobre a aplicação das heurísticas, são parecidos. Acredita-se que os alunos foram sinceros ao responder ao questionário tendo em vista o interesse apresentado pela maioria em aprender o conteúdo trabalhado na disciplina.

Compreender o problema antes de buscar sua solução é a primeira heurística proposta por Polya. Em relação ao entendimento dos enunciados, os alunos foram questionados sobre o nível de dificuldade. As respostas dos alunos da Turma P foram, 3% dos alunos disseram achar o entendimento do enunciado bastante difícil, 53% afirmaram ser um pouco difícil e 35% afirmaram ser fácil entender o enunciado dos problemas a serem resolvidos. Já as respostas da Turma NP foram 12% dos alunos disseram achar o entendimento do enunciado bastante difícil, 72% afirmaram ser um pouco difícil e 16% afirmaram ser fácil entender o enunciado dos problemas a serem resolvidos. Comparando-se as turmas não há diferenças muito pronunciadas no que diz respeito à interpretação dos problemas, mas com uma diferença positiva em favor da turma P.

O planejamento da solução do problema é a segunda heurística proposta por Polya. Os alunos foram questionados sobre o grau de dificuldade desta ação. As respostas da turma P, 10% responderam ser bastante difícil tal planejamento, 52% um pouco difícil, 6% responderam ser fácil planejar a solução e 32% responderam fazer pouco ou nenhum planejamento. As respostas da turma NP foram, 40% responderam ser bastante difícil tal planejamento, 48% um pouco difícil, 8% responderam ser fácil planejar a solução e 4% responderam fazer pouco ou nenhum planejamento. Vale ressaltar que a visão de planejamento das turmas é diferente. Enquanto para a turma P planejamento está relacionado à aplicação das heurísticas para a turma NP o planejamento está relacionado à construção de passos lógicos em linguagem natural. É possível que os alunos da turma P tenham percebido a fase de planejamento como algo mais complexo.

Os estudantes foram também questionados sobre o que despertou o gosto pelas aulas durante o 1º semestre letivo. Essa pergunta aberta foi respondida por 22 alunos da Turma P; as respostas foram 32% dos alunos responderam que o que mais gostaram foi aprender programando, 41% responderam que foi o ambiente de programação *Scratch*, 5% responderam que foi aprender a resolver problemas e 23% responderam que as aulas em sala foi o que mais gostaram durante o primeiro semestre letivo. Foram 12 os alunos da Turma NP que responderam essa questão,

50% dos estudantes responderam que o que mais gostaram foi construir algoritmos, 33% responderam que foi o momento em que concluíram corretamente um algoritmo e 17% responderam que o que mais gostaram foi o interesse dos alunos em aprender. As respostas da turma P confirmam o interesse geral da classe (mais de 70%) em aprender fazendo.

As principais dificuldades apresentadas por 18 alunos da Turma P foram, 17% a fundamentação teórica, 28% alguns comandos do *Scratch*, 11% baixa capacidade para resolução de problemas, 22% compreensão dos problemas, 17% compreensão das explicações e 6% refinamentos sucessivos. As principais dificuldades apresentadas segundo 26 alunos da Turma NP, 19% responderam ser criar algoritmos, 23% planejar as soluções, 35% entender o enunciado dos problemas, 15% acertar a sintaxe e 8% baixo raciocínio lógico. Ao comparar esses dados com os coletados da Turma NP, onde as estratégias de ensino estão baseadas na escrita de pseudocódigo, de um total de 26 alunos, 92% das dificuldades sentidas por eles estão relacionadas a uma das dificuldades mencionadas na seção 2.3 desta pesquisa; lembrando: abstrações envolvidas, ausência de comprovação prática, não compreensão de pseudocódigo e/ou desenvolvimento de fluxogramas, linguagens de programação com muitas e complexas regras de sintaxe, baixa capacidade para resolução de problemas, resistência por parte do aluno e impossibilidade do professor adequar-se a necessidade de cada aluno. Os resultados indicam ser possível que o estímulo à prática de resolução de problemas, através de atividades desafiadoras e de interesse dos alunos, em um ambiente lúdico, possa contribuir para diminuir as dificuldades.

Dados relacionados à motivação dos estudantes são apresentados na tabela 2.

Tabela 2: Motivação apresentada pelos estudantes

Motivação	Turma P	Turma NP
Motivados a estudar no ambiente escolar	90%	92%
Motivados a estudar fora do ambiente escolar	84%	76%

Fonte: Autoria própria

Os dados apresentados na tabela 2 mostram os estudantes das duas turmas igualmente motivados a estudar tanto no ambiente escolar quanto fora dele. No

entanto, a observação realizada mostra que a motivação apresentada pelos alunos da turma NP estava relacionada à obtenção de notas; já os alunos da turma P estavam de fato motivados a programar, sendo a nota algo secundário. Enquanto os alunos da primeira refaziam exercícios já vistos em sala de aula, em pseudocódigo, os alunos da segunda criavam jogos e animações complexos motivados pela sensação de aprender o novo, nesse caso, a programação.

Os resultados obtidos com a aplicação dos questionários sugerem que, os roteiros de aula aplicados no primeiro semestre letivo de 2009, foram suficientes para reforçar a aplicação das heurísticas para resolução de problemas, propostas por Polya. Mesmo a Turma P tendo através das respostas referentes a tal aplicação, demonstrado aplicá-las em menor escala, as respostas sobre a compreensão do enunciado e planejamento da solução, indicam maior facilidade para execução de tais atividades. É possível que essas atividades tenham sido realizadas em grau maior, no entanto, por terem sido feitas mentalmente não tenham sido mencionadas quando questionado.

A motivação, essencial para despertar o interesse dos alunos se fez presente também nas respostas coletadas. Tal motivação se fez essencial para estimular a criatividade e autonomia nos estudantes e estes passaram mais tempo programando em ambiente não escolar. Dessa forma, é possível reconhecer que os roteiros aplicados foram suficientes para estimular características buscadas pelo construcionismo.

6.3.5 Roteiro de Aula VIII

O **Roteiro de Aula VIII** corresponde as aulas práticas em laboratório de informática, através da implementação de *software*s em linguagem de programação *Python*. Esse roteiro de aula é composto por 6 h/a.

A linguagem *Python* foi usada como passo intermediário antes dos alunos começarem a usar uma das linguagens adotadas pelo curso: Pascal. Esse passo intermediário tem o objetivo de fazer à transição de uma representação pictórica (*Scratch*) a outra textual (*Python*) da forma mais suave possível. Posteriormente em Pascal, os alunos estarão sujeitos a uma sintaxe um pouco mais carregada de sinais de pontuação. Além disso, a representação de blocos de comandos e de

aninhamento de estruturas que era evidente em *Scratch* é mais clara em *Python* do que em Pascal, contribuindo assim para uma passagem gradativa de uma linguagem a outra.

Inicialmente foi entregue aos alunos um manual contendo exemplos e atividades a serem realizadas utilizando a linguagem de programação *Python*. Assim como o manual *Scratch*, o manual *Python* tem atividades divididas em exercícios e desafios, sendo os últimos propostos como desenvolvimento de jogos.

As aulas ocorreram sem tumulto. Chegando ao laboratório, as equipes se organizavam e iniciavam a implementação. As aulas utilizando *Scratch* parece ter contribuído para tornar os alunos mais focados, sendo a dinâmica das aulas diferente daquelas em sala de aula.

Em alguns momentos, alunos ficaram dispersos das atividades, em conversas paralelas ou navegando na *web*. Diferente das aulas com *Scratch*, as atividades em *Python* foram encaradas como obrigação e não como algo interessante. Alunos com maior aptidão para programação participaram das atividades propostas; os demais, mesmo sendo incentivados pela professora, muitas vezes nem ao menos procuraram compreender o que estava sendo feito.

A figura 10 apresenta alunos em laboratório utilizando *Python*.

Figura 10: Alunos utilizando a linguagem *Python*
Fonte: Autoria própria

Em relação ao aprendizado das estruturas básicas da programação, houve reforço do conteúdo para aqueles alunos que de fato implementaram os exemplos e realizaram os exercícios e desafios propostos.

Sobre a resolução de problemas, verificou-se o planejamento da solução em 25% das equipes. Tais equipes criaram esboços dos *software*s a serem desenvolvidos e realizaram testes prévios antes da implementação direta em *Python*. Através da observação realizada acredita-se que outra parte da sala, por uma maior aptidão em programar, tenha realizado igual planejamento, porém sem expressá-lo de forma escrita. Equipes que desenvolveram as atividades propostas programando por tentativa, gastaram mais tempo para chegar à solução que outras.

6.3.6 Roteiro de Aula IX

O **Roteiro de Aula IX** é o último proposto. Esse roteiro compreende 10 h/a e os alunos devem desenvolver um *software* em linguagem de programação *Python*. Esse *software* deve ser um jogo de perguntas e respostas sobre um determinado tema escolhido pela equipe.

Os critérios para avaliação quantitativa do *software* estão no apêndice D. O gráfico 5 apresenta as notas obtidas pelas equipes em relação ao *software* entregue. O gráfico 6 apresenta as notas obtidas com avaliação oral, e individual, sobre o *software* entregue. Os questionamentos feitos no momento de tal avaliação tinham por objetivo verificar a aplicação de estruturas de repetição, seleção e variáveis. Eles foram elaborados mediante o código do *software* entregue.

A elaboração de *software*s em *Scratch* e *Python* representam processos diferentes não permitindo a comparação direta entre ambos. Todavia, vale destacar que houve queda acentuada das notas obtidas pelos jogos entregues em *Python* em comparação com os *software*s, jogos ou animações, entregues em *Scratch*. Enquanto em *Scratch* 100% dos *software*s entregues obtiveram notas acima de 50% do total, em *Python* esse índice foi de 83%. Acredita-se que a queda das notas tenha sido causada pelo desinteresse por parte dos alunos tão logo perceberam a linguagem *Python* como textual e não gráfica como *Scratch*.

Em relação a avaliação oral sobre o *software* desenvolvido em *Python*, é possível identificar um maior número de alunos que não participou do processo de criação. Enquanto em *Scratch* esse número representava 17%, em *Python* passou a representar 29%. Analisando particularmente cada equipe, percebe-se pela

avaliação oral que em 50% das equipes, ao menos um aluno não participou do desenvolvimento. Em *Scratch* esse índice era de 33%.

Gráfico 5: Notas obtidas pelas equipes com a entrega do *software* desenvolvido em *Python*
Fonte: Autoria própria

A dinâmica durante as aulas do Roteiro de Aula IX foi semelhante à percebida durante as aulas do roteiro anterior. Alunos antes motivados utilizando o ambiente de programação *Scratch* não demonstraram o mesmo interesse; algumas equipes que desenvolveram *software*s interessantes em *Scratch* não entregaram programas em *Python*. Mesmo tendo sido proposto o desenvolvimento de um jogo, é possível que a não presença de elementos como imagens e som tenha tido uma repercussão negativa.

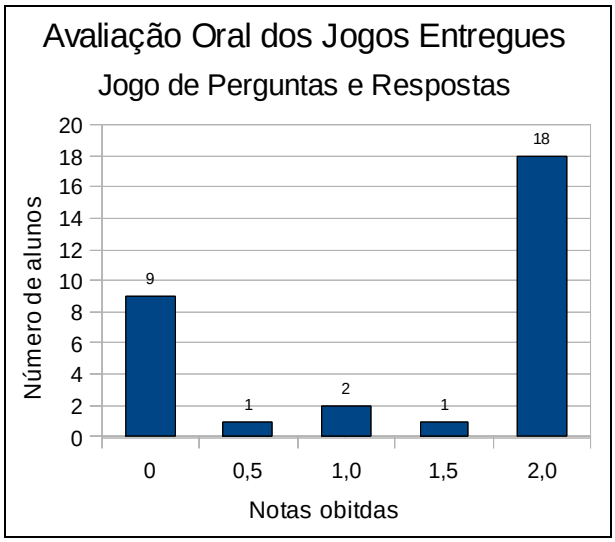


Gráfico 6: Notas obtidas individualmente pelos estudantes em avaliação oral
Fonte: Autoria própria

6.3.7 Média Aritmética dos Três Bimestres Letivos de 2009

É possível fazer uma comparação não rigorosa entre os resultados obtidos pelas equipes, por meio da média das notas dos três primeiros bimestres. O resultado é apresentado no gráfico 7.

Percebe-se que a média da turma participante da pesquisa foi 0,6 pontos superior à turma não participante e satisfez o critério mínimo para aprovação (6,0).

É provável que se os alunos mantivessem o mesmo envolvimento demonstrado com *Scratch*, a diferença entre as médias poderia ser ainda maior.

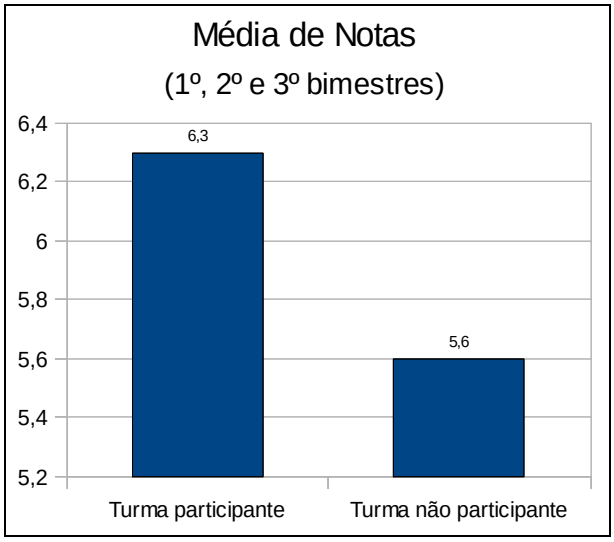


Gráfico 7: Média Notas (comparativo entre turmas)
Fonte: Autoria própria

O resultado geral da comparação entre as turmas foi bastante positivo. A melhora nas médias de avaliação pode ser encarada como um reflexo de vários efeitos observados. Os alunos se mostraram mais interessados e quando deixados livres para criar, apresentaram programas de complexidade maior do que a encontrada habitualmente em turmas da disciplina.

Houveram indícios de que os alunos dedicaram mais tempo extra-sala para as atividades. As aulas práticas em laboratório usando *Scratch* foram mais prazerosas.

7 CONSIDERAÇÕES FINAIS

Durante a execução deste trabalho as diferentes etapas permitiram apreciar vários aspectos do problema e obter informações para delinear algumas conclusões.

A pesquisa exploratória mostrou que o ensino dos conceitos fundamentais da programação, nas classes pesquisadas, enfatiza principalmente (ou tão somente) a escrita de pseudocódigo. No ensino superior percebe-se que em seguida ao pseudocódigo os alunos são levados ao laboratório de informática e passam a usar linguagens de programação e ambientes de programação comerciais. Essas ferramentas não tratam dificuldades de natureza pedagógica; exemplos são inadequabilidades de ergonomia, como mensagens de erro de difícil interpretação; inadequabilidade de regras sintáticas e de pontuação que desviam a atenção e sobrecarregam alunos iniciantes. Essa forma de trabalho introduz no processo de aprendizagem algumas dificuldades extras que poderiam ser evitadas em um primeiro momento. É possível detectar também o estímulo ao aprendizado de uma linguagem ou ambiente de programação específico em detrimento ao estímulo à resolução de problemas.

Segundo o levantamento feito, no ensino médio a primeira disciplina de programação é apresentada apenas em sala de aula, sem contato com ferramentas computacionais, o que aumenta a carga de abstração. As dificuldades relacionadas à sintaxe e semântica, bem como resolução de problemas, são mencionadas em todos os contextos onde a pesquisa foi aplicada.

Finalmente, existem esforços pontuais para uso de ferramentas mais apropriadas; foram detectadas na pesquisa Múltipla e Visualg. No entanto, exercícios de caráter tradicional como construção de sequências numéricas e cadastros, são geralmente propostos. Tais problemas dificilmente despertarão o mesmo interesse que construir um jogo ou uma animação gráfica em que o aluno tem liberdade para criar regras e personagens.

O projeto piloto com o ambiente de programação *Scratch* também revelou outras dificuldades apresentadas pelos alunos. Pode-se citar a baixa capacidade de resolver problemas, dificuldades para adaptar conhecimentos prévios em novas situações, a má interpretação de enunciados e o não planejamento de soluções. O

ambiente *Scratch* mostrou-se motivador, tornando o aprendizado mais dinâmico e divertido.

Ao analisar os dados coletados através de observações e entrevistas, antes da elaboração dos roteiros de aula, percebe-se que as dificuldades apresentadas pelos alunos correspondem a relatos da literatura brasileira e internacional. Esse fato sugere que existem problemas de fundo que devem ser intrínsecos a essa disciplina, não determinados pela realidade local de uma classe específica de alunos. Em particular, torna-se difícil sustentar que o uso ou não de português, inglês ou uma língua artificial (como C++, Java ou uma das várias versões de pseudocódigo) tenha uma influência decisiva ou mesmo preponderante no processo de ensino-aprendizagem.

Roteiros de aula foram elaborados, tendo por fundamento estimular a capacidade de resolução de problemas em ambientes motivadores. A escolha de ambientes e linguagens buscou critérios pedagógicos, como reduzir a complexidade sintática e a carga de abstração envolvida. Após a experiência com a aplicação desses roteiros tornou-se possível tecer algumas conclusões sobre o papel da abstração, da sintaxe, da resolução de problemas e a influência de características do construcionismo como motivação, autonomia e responsabilidade.

A apresentação e discussão de trechos curtos de código já nas aulas iniciais da disciplina, é capaz de fornecer modelos mentais adequados das estruturas básicas da programação de computadores, tais como laços e condições booleanas. Isso contribui para diminuir a carga de abstração exigida em momentos subsequentes, na medida em que os estudantes já tenham internalizado um conceito e encarem C++, fluxograma ou qualquer outra linguagem textual ou pictórica como uma representação possível daquele conceito. Isto contrasta com a aula que enfatiza a escrita de programas (usando uma daquelas representações), sem que os alunos já tenham criado um modelo mental adequado. A apresentação de exemplos de trechos de códigos pela professora e a implementação pelos alunos em ambientes de programação, antes da proposta de exercícios e desafios, também contribui para a elaboração de tais modelos.

O ambiente *Scratch* favorece que os estudantes não se preocupem com regras de sintaxe. Percebeu-se que, em segundo momento, quando da utilização da linguagem de programação *Python*, os alunos já possuíam entendimento básico das

estruturas fundamentais da programação e por isso não apresentaram dificuldades para empregá-las, respeitando as regras sintáticas impostas pela linguagem.

Em relação à resolução de problemas percebeu-se que os alunos o fazem diariamente sem pensar sobre o processo em si e sobre as etapas envolvidas. Ao apresentar as heurísticas de Polya incentivando o investimento de tempo pensando sobre o problema, percebeu-se nos alunos a tentativa de proceder de forma mais consciente. O assunto tornou-se interessante, sendo inclusive mencionado em questionários como o item que mais agradou a alguns estudantes. Em relação à etapa de planejamento exigiu maior atenção tendo sido necessário o reforço da sua importância em alguns momentos da disciplina.

O interesse e a motivação dos alunos se fez presente de forma intensa na maioria das aulas. Houve uma queda desse interesse quando da utilização da linguagem de programação *Python*; é possível que esse não interesse seja resultado das notas obtidas no primeiro semestre, paralisações ocorridas devido a Gripe A (H1N1), ou ainda pela linguagem *Python* não oferecer o apelo visual oferecido por *Scratch*. Ainda durante a aplicação dos roteiros de aula baseados em *Python*, o trabalho em equipe também não ocorreu de forma tão satisfatória. Os alunos que apresentaram desinteresse não programaram, deixando a tarefa a cargo dos demais colegas. Esse abandono da atividade, mesmo com incentivo por parte da professora, foi detectado em avaliação oral a respeito de um jogo desenvolvido em sala. Foi sem dúvida, a aplicação de *Scratch* que retornou conclusões mais satisfatórias. Durante as aulas os alunos estiverem concentrados nas atividades propostas, incluindo aqueles alunos que não conseguiram alcançar todo o conhecimento necessário. Os exercícios e desafios propostos, tendo caráter lúdico, cumpriram o objetivo de estimular a curiosidade, a responsabilidade e a autonomia dos estudantes. Os desafios em especial, proporcionaram aos alunos a iniciativa de buscar novos conhecimentos e aplicar os já adquiridos, para isso empregando mais tempo em horário extra-classe.

A observação mostra que em relação à resolução de problemas a turma participante apresentou menos dificuldades, em especial em relação à interpretação de enunciados e no planejamento. Quando questionados sobre dificuldades experimentadas, sintaxe e abstração foram mencionadas diversas vezes por alunos da turma não participante, mas em menor escala na turma onde os roteiros de aula foram aplicados. Embora as notas obtidas não representem um avaliação fidedigna

de todo o processo, vale também ressaltar a média mais elevada da turma participante da pesquisa: diferença superior a 0,6 pontos.

Esse trabalho de pesquisa considerou a proposta de estratégias de ensino, contendo exercícios lúdicos e desafiadores, como elemento capaz de motivar o educando, estimulando sua capacidade em resolver problemas. O fator inovador está em aliar explicitamente os princípios do construcionismo, heurísticas para resolução de problemas e modelos mentais, como fundamento para elaboração de tais estratégias. Os resultados positivos obtidos indicam essas estratégias como uma alternativa para professores das disciplinas iniciais da programação de computadores, que busquem minimizar as dificuldades apresentadas pelos alunos novatos em programação, em especial no contexto da programação procedural.

Como trabalhos futuros sugere-se o acompanhamento dos alunos participantes da pesquisa nas disciplinas de programação subsequentes, com o objetivo de verificar os resultados posteriores da aplicação das estratégias de ensino. Sugere-se também, a aplicação de tais estratégias no ensino superior, mesmo onde a orientação a objetos é o paradigma sugerido.

REFERÊNCIAS

ALA-MUTKA, K. **Problems in learning and a teaching programming – a literature study for developing visualizations**. In: Codewitz-Minerva projetct. Universidade da Finlândia, 2004. Disponível em http://www.cs.tut.fi/~codewitz/literature_study.pdf. Acesso em outubro de 2008.

ALMEIDA, E. S.; COSTA, E. B.; SILVA, K. S.; PAES, R. B.; ALMEIDA, A. A. M.; BRAGA, J. D. H. **AMBAP: Um ambiente de apoio ao aprendizado de programação**. In: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO, 10., 2002, Florianópolis. Anais... Florianópolis: SBC, 2002.

BARANAUSKAS, M. C. C. **Procedimento, função, objeto ou lógica: linguagens de programação vistas pelos seus paradigmas**. In: VALENTE, José Armando, Org. Computadores e conhecimento: repensando a educação. Campinas, NIED/UNICAMP, p.45-63, 1993.

BARDIN, L. **Análise de conteúdo**. Lisboa: Ed. 70, 1979.

BEN-ARI, M. **Constructivism in computer science education**. In: SIGSCE - Technical Symposium on Computer Science Education, Atlanta, GA, USA. p. 257-261, 1998.

CARBONI, I. F. **Lógica de programação**. Editora Cengage Learning, 2003.

CHEN, S.; MORRIS, S**. Iconic Programming for Flowcharts, Java, Turing, etc. Iconic programming for flowcharts, java, turing, etc**. In: Proceedings of the 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education. ACM Press, p. 104-107, 2005.

COOPER, S.; DANN, W.; PAUSCH, R. **Teaching objects first in introductory computer science**. In: Proceedings of the 34th Technical Symposium on Computer Science Education, Reno, Nevada, USA, 2003.

CRISTOVAO, H. M. **Aprendizagem de algoritmos num contexto significativo e motivador: Um relato de experiência**. In: Anais do XXVIII Congresso da SBC. Belém do Pará – PA, p. 30–40, 2008a.

_____. **Roteiros de exercícios para o *Scratch***. FAESA – Unidade de Computação e Sistemas, 2008b. Disponível em http://algoritmo.wikidot.com/. Acesso em agosto de 2008.

DANTE, L. R. **Didática da resolução de problemas de matemática**. São Paulo: Editora Ática, 2005.

DAVIS, E. J.; MCKILLIP, W. D. **Aperfeiçoando a resolução de problemas-história na matemática da elementary school**. In: A resolução de problemas na matemática escolar. Editora Atual. São Paulo. Organização: Stephen Krulik e Robert E. Reys. p. 114–130, 1997.

DEEK, F.; KIMMEL, H; McHUGH, J. **Pedagogical changes in the delivery of the first-course in computer science: problem solving, then programming**. Journal of Engineering Education, p. 313-320, 1998.

DELGADO, C.; XEXÉO, J. A. M.; SOUZA, I. F.; CAMPOS, M., RAPKIEWICZ, C. E. **Uma abordagem pedagógica para a iniciação ao estudo de algoritmos**. In: Anais do XII Workshop de Educação em Computação. Salvador, BA, Brasil, 2004.

DUNICAN, E. **Making the analogy: Alternative delivery techniques for first year programming courses**. In: 14th Workshop of the Psychology of Programming Interest Group, Brunel University, p. 89-99, 2002.

EDWARDS, J. **Example centric programming**. In: OOPSLA'04, Vancouver, British Columbia, Canada, 2004.

FALCKEMBACH, G. A M.; ARAUJO, F.V. **Aprendizagem de algoritmos: dificuldades na resolução de problemas**. In: Simpósio Internacional de Informática Educativa – SIIE, 2006.

FREITAS, H. B. **Uma linguagem de programação LOGO para dar apoio ao aprendizado de jovens programadores**. 2008. 49f. (Monografia) Graduação – Bacharelado em Engenharia da Computação, Escola Politécnica de Pernambuco, 2008.

GEORGE, C. E. **Experiences with novices: The importance of graphical representations in supporting mental models**. In: 12th Workshop of the Psychology of Programming Interest Group, Cozenza Italy, p. 33-44, 2000.
GIL, A. C. **Como elaborar projetos de pesquisa**. São Paulo: Editora Atlas, 1991.

GOMES, A.; HENRIQUES, J.; MENDES, A. J. **Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores**. In: Educação, Formação & Tecnoloigias; vol.1, p. 93-103, 2008.

GOMES, A.; MENDES, A. J. **Problem solving in programming**. In: The Proceedings of PPIG as a Work in Progress Report, 2007.

GOOSEN, L. **Criteria and guidelines for the selection and implementation of a first programming languagem in high schools**. 2004. Tese (Doutorado) – Doutorado em Filosofia). North-West University, 2004.

HAVENGA, H. M. **An investigation of students knowledge, skills and strategies during problem solving in object-oriented programming**. 2008. 306f. Tese (Doutorado) – Doutorado em Filosofia da Matemática, Ciência e Educação Tecnológica. University of South Africa, 2008.

JACOBSON, M. H.; LESTER JR, F.; STENGEL, A. **Tornando a resolução de problemas mais animada nas séries intermediárias**. In: A resolução de problemas na matemática escolar. Editora Atual. São Paulo. Organização: Stephen Krulik e Robert E. Reys. p. 177–187, 1997.

JENKINS, T. **On the difficulty of learning to program**. In: Proceedings of 3rd Annual LTSN_ICS Conference (Loughborough University, United Kingdom, August 27-29). The Higher Education Academy. p. 53-58, 2002.

KELLEHER, C.; PAUSCH, R. **Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers**. In: ACM Computing Surveys (CSUR), v.37 n.2, p.83-137, 2005.

KHALIFE, J. T. **Threshold for the Introduction of Programming: Providing Learners with a Simple Computer Model**. In: 18th Workshop of the Psychology of Programming Interest Group, University of Sussex, September 2006. p. 244-254, 2006.

LIMA, M. R. **Construcionismo de Papert e ensino-aprendizagem de programação de computadores**. 2009. 153f. Dissertação (Mestrado) – Mestrado em Educação. Universidade Federal de São João Del-Rei – Minas Gerais, 2009.

MALONEY, J.; PEPPLER, K.; KAFAI, Y.; RESNICK, M.; RUSK, N. **Programming by choice: urban youth learning programming with *Scratch***. In: ACM SIGCSE Bulletin archive, Vol 40, p. 367-371, 2008.

McLVER, L.; CONWAY, D. **Seven deadly sins of introductory programming language design**. In: Proceedings, *Software* Engineering: Education & Practice (SE:E&P'96). p. 309-316, 1996.

MIRANDA, E. M. **Uma ferramenta de apoio ao processo de aprendizagem de algoritmos**. 2004. 128f. Dissertação (Mestrado) - Mestrado em Ciência da Computação. Universidade Estadual de Santa Catarina, 2004.

MIT, Lifelong Kindergarten Group. **Getting stardet with *Scratch***, 2008. Disponível em: http://*scratch*.mit.edu. Acesso em agosto de 2008.

MOSKAL, B.; LURIE, D.; COOPER, S. **Evaluating the effectiveness of a new instructional approach**. In: SIGCSE 2000, Norfolk, Virgina, USA, 2000.

MOTIL, J.; EPSTEIN, D. **JJ: a Language designed for beginners (Less Is More)**, 2000.

NAPS, T. L.; ROBLING, G.; ALMSTRUM, V.; DANN, W.; FLEISCHER, R.; HUNDHAUSEN, C.; KORHONEN, A.; MALMI, L.; McNALLY, M.; RODGER, S.; VELASQUEZ-ITURBIDE, J.A. **Exploring the role of visualization and engagement in computer science education**. In: ACM SIGCSE Bulletin, v.35. n.2, 2003.

PAPERT, S. **A máquina das crianças: repensando a escola na era da informática**. Edição revisada. Porto Alegre: Editora Artmed, 2008.

PARKER, K. R.; CHAO, J. T.; OTTAWAY, T. A.; CHANG, J. **A formal language selection process for introductory programming courses**. In: Journal of Information Technology Education, vol. 5, 2000.

PEREIRA JÚNIOR, J. C. R.; RAPKIEWICZ, C. E. **O Processo de ensino e aprendizagem de algoritmos e programação: Uma visão crítica da literatura**. In: III Workshop de Educação em Computação e Informática do Estado de Minas Gerais, WEIMIG'04, Belo Horizonte – MG, 2004.

PETRY. P. G. **Um sistema para o ensino e aprendizagem de algoritmos utilizando um companheiro de aprendizagem colaborativo**. 2005. 94f. Dissertação (Mestrado) - Mestrado em Ciência da Computação – Universidade Federal de Santa Catarina, 2005.

PINHEIRO, M. C. **Uma experiência no ensino de lógica de programação para cursos de engenharia usando o pascal e o logo**. In: WEIMIG, 2003.

POLYA, G. **A arte de resolver problemas**. Rio de Janeiro: Editora Interciência, 1978.

QUIVY, R.; CAMPENHOUDT, L.V. **Manual de investigação em ciências sociais**. Gradiva Publicações, 2005.

ROCHA, R. Programação **orientada a objetos: uma abordagem lúdica**. In: Revista Educação & Tecnologia, CEFET-MG, Belo Horizonte, v. 8, n. 1, p. 45-49, 2003.

SEBESTA, R. W. **Concepts of programming languages**. 7th ed. Boston: Pearson Addison Wesley, 2006.

SILVA. E.L.; MENEZES. E.M. **Metodologia da pesquisa e elaboração de dissertação**. 3 ed. Revista Atual. Florianópolis: Laboratório de Ensino à Distância da UFSC, 2001.

SILVA, H. V. R. C. **Representações computacionais auxiliares no entendimento de conceitos de programação**. 1991. 461f. Tese (Doutorado) – Doutorado em Engenharia Elétrica. Universidade Estadual de Campinas, 1991.

SOBRAL, S. C. R. L. **B-Learning em disciplinas introdutórias de programação**. 2008. 171f. Tese (Doutorado) Doutorado em Tecnologias e Sistemas de Informação – Universidade do Minho – Portugal, 2008.

TUCKER, R. **A developmental study of cognitive problems in learning to program**. In: 15th Workshop of the Psychology of Programming Interest Group, Keele UK, p. 325-332, 2003.

VARGAS, K.S.; MARTINS, J. **Ferramenta para apoio ao ensino de introdução à programação**. In: Anais do IX Seminário de Computação, Blumenau, 2005.

WIRTH, N. **Program development by stepwise refinement**. Communications of the ACM, vol. 14. p. 221-227, 1971.

WOLZ, U.; LEITNER, H. H.; MALAN, D. J.; MALONEY, J. **Starting with *Scratch* in CS 1**. In: SIGSCE'09 Chattanooga, Tennessee, USA, 2009.

APÊNDICES

Apêndice A – Questionários aplicados

Sou aluna do Mestrado em Ensino de Ciência e Tecnologia, do Programa de Pós-graduação da Universidade Tecnológica Federal do Paraná. Gostaria de conhecer como está a sua disciplina de Introdução à Programação. Sua opinião é muito importante. Então, peço a sua colaboração para que responda às seguintes perguntas.

Elena Mariele Bini

Instituição de Ensino: _____Turno:_____

1 Idade: _____

2 Sexo: ( ) feminino ( ) masculino

3 Trabalha: ( ) não ( ) sim, período integral ( ) sim, meio período

4 Marque tudo o que for verdade sobre Fluxogramas:

( ) não foi mostrado na aula ( ) foi ensinado e pouco usado ( ) foi ensinado e bastante usado

5 Marque tudo o que for verdade sobre Portugol / Português Estruturado:

( ) não foi mostrado na aula ( ) foi ensinado e pouco usado ( ) foi ensinado e bastante usado

6 Marque tudo o que for verdade sobre Refinamentos Sucessivos:

( ) não foi mostrado na aula ( ) foi ensinado e pouco usado ( ) foi ensinado e bastante usado

7 Você teve aula prática, dos conceitos iniciais da programação, em laboratório?

( ) não, nunca ( ) sim, desde o começo

( ) sim, mas só depois de aprender certas coisas. Quais? _____

8 Quais os programas e as linguagens usados em laboratório?

9 Mais ou menos quantas linhas de código tinha o maior programa que você fez? _____

10 Se você precisa escrever um programa, antes você faz um plano?

( )Sim. Antes faço um plano (fluxograma, ou português estruturado, ou texto + refinamentos sucessivos)

( )Não. Escrevo já na linguagem de programação

( )Às vezes faço um planejamento, às vezes escrevo direto na linguagem. (Explique quando)

11 Assinale os itens que representam exemplos de problemas resolvidos por você, durante as aulas na disciplina introdutória de programação:

( ) apresentar a tabuada de um número qualquer digitado pelo usuário;

( ) fazer um cadastro de livros em uma biblioteca;

( ) fazer jogos de perguntas e respostas;

( ) fazer jogos com animações computacionais.

12 Marque aquilo que você costuma fazer:

Compreender o problema antes de tentar elaborar um algoritmo ou programa para solucioná-lo:

( ) Sim ( ) Não

Após compreender o problema, procurar um problema semelhante já resolvido:

( ) Sim ( ) Não

Planejar a solução e testá-la (usando fluxograma ou algoritmo) ANTES de programar no laboratório:

( ) Sim ( ) Não

Rever a solução elaborada, mesmo ela sendo correta, com o objetivo de melhorá-la:

( ) Sim ( ) Não

13 Para a maioria dos exercícios da disciplina, o que você achou destes itens:

Acertar a sintaxe: sinais de ponto e vírgula, parênteses, aspas...

( ) bastante difícil ( ) um pouco difícil ( ) fácil

Entender o enunciado: o que deve ser feito

( ) bastante difícil ( ) um pouco difícil ( ) fácil

Planejar o Algoritmo: resolver o problema

( ) faço pouco planejamento, ou nunca faço e programo direto

( ) bastante difícil ( ) um pouco difícil ( ) fácil

14 Você foi incentivado a fazer os exercícios sozinho?

( ) Sim ( ) Não

15 Você dispunha de algum tempo extra-classe para rever os conteúdos ministrados pelo professor?

( ) Sim ( ) Não

16 Você se sentiu motivado a estudar, fora do ambiente escolar, a programação de computadores?

( ) Sim ( ) Não, por quê? _____

17 Você conseguiu acompanhar o andamento das atividades propostas pelo professor da disciplina?

( ) Sim ( ) Não

18 Para você, quais as principais dificuldades para o aprendizado da disciplina introdutória de programação?

19 Você, provavelmente, será aprovado nessa disciplina?

( ) Sim ( ) Não

Apêndice B – Critérios para avaliação quantitativa do *software* desenvolvido em *Scratch*

Descrição da atividade:
Vocês utilizarão sua criatividade e conhecimentos adquiridos da ferramenta *Scratch* para criar um programa de interesse do grupo. Esse programa poderá ser um jogo, uma animação ou algo similar, porém ele deverá conter:

- no mínimo dois sprites, sendo que nenhum deles deverá ser o gato;
- no mínimo três scripts, não necessariamente um por spryte, porém o background deverá conter pelo menos um script;
- o projeto deverá ter no mínimo uma estrutura condicional e uma variável;
- o projeto precisa ser mais complexo do que os exercícios já realizados.

Critérios para avaliação:

Categoria	4	3	2	1
Elementos exigidos	Todos os elementos exigidos estão presentes: 2 sprites, 3 scripts (um em background), 1 estrutura condicional, 1 variável	Um elemento faltando	Dois elementos faltando	Três ou mais elementos faltando
Clareza dos scripts	Scripts lógicos e eficazes. A lógica do programa é facilmente percebida através dos scripts	Scripts em sua maioria são lógicos e eficazes. A lógica do programa é percebida através dos scripts	Scripts são lógicos mas não eficazes. A lógica do programa não é facilmente percebida através dos scripts	Os scripts não são lógicos nem eficazes. Muito difícil compreender a lógica do programa através dos scripts
Criatividade	O programa é interessante e divertido. Background, sprites e cores tornaram o programa agradável	Programa interessante e divertido, porém faltou criatividade em alguns elementos	Tentativa de tornar o programa interessante e divertido, porém alguns elementos tornaram o *software* de difícil compreensão	O programa apresenta falta de criatividade. O programa não é interessante e não leva a diversão
Recursos visuais	Escolha dos objetos mostra preocupação com o design do programa	Escolha dos objetos mostra uma tentativa de adequação do design do programa	Escolha dos objetos mostra pouca preocupação com a adequação do design do programa	Escolha dos objetos mostra nenhuma preocupação com o design do programa
Conhecimentos da linguagem *Scratch*	O projeto demonstra um sofisticado conhecimento do ambiente *Scratch*	O projeto demonstra um conhecimento funcional do ambiente *Scratch*	O projeto demonstra um conhecimento limitado do ambiente *Scratch*	O projeto demonstra pouco conhecimento do ambiente *Scratch*

Adaptação de:
Projeto Alg-Geo disponível em http://www.box.net/shared/ks2oqmkk5z.

Apêndice C – Questionário aplicado ao final da utilização do *software Scratch*

Como aluna do Mestrado em Ensino de Ciência e Tecnologia, do Programa de Pós-graduação da Universidade Tecnológica Federal do Paraná, gostaria de conhecer sua opinião sobre a disciplina de Lógica de Programação. Sua opinião é muito importante. Então, peço a sua colaboração para que responda às seguintes perguntas, com o máximo de sinceridade. Você NÃO precisa colocar seu nome - Elena Mariele Bini.

1 Sexo: ( ) feminino ( ) masculino

2 Trabalha: ( ) não ( ) sim, período integral ( ) sim, meio período

3 Marque aquilo que você realizou antes de criar seus programas em *Scratch*:

Compreender o problema antes de tentar elaborar o programa para solucioná-lo:

( ) Sim ( ) Não

Após compreender o problema, procurar um problema semelhante já resolvido:

( ) Sim ( ) Não

Planejar a solução e testá-la antes de programar em si:

( ) Sim ( ) Não

Rever a solução elaborada, mesmo ela sendo correta, com o objetivo de melhorá-la:

( ) Sim ( ) Não

4 Para a maioria dos exercícios da disciplina, o que você achou destes itens:

Entender o enunciado: o que deve ser feito

( ) bastante difícil ( ) um pouco difícil ( ) fácil

Planejar o Programa: resolver o problema

( ) faço pouco planejamento, ou nunca faço e programo direto

( ) bastante difícil ( ) um pouco difícil ( ) fácil

5 Você dispunha de algum tempo extra-classe para rever os conteúdos ministrados pelo professor?

( ) Sim ( ) Não

6 Você se sentiu motivado a estudar, no ambiente escolar, a programação de computadores?

( ) Sim ( ) Não, por quê? _____

7 Você se sentiu motivado a estudar, fora do ambiente escolar, a programação de computadores?

( ) Sim ( ) Não, por quê? _____

8 O que você mais gostou das aulas de lógica de programação durante o primeiro semestre?

9 O que você menos gostou nas aulas de lógica de programação durante o primeiro semestre?

10 O que você acredita que poderia ser melhorado na metodologia utilizada pela professora durante as aulas do primeiro semestre?

11 O que você acredita que deve permanecer na metodologia utilizada pela professora durante as aulas do primeiro semestre?

12 Para você, quais as principais dificuldades para o aprendizado da disciplina introdutória de programação?

13 Quando iniciou o curso Técnico em Informática, você o fez por vontade própria? Você considera trabalhar na área de programação de computadores?

Apêndice D – Critérios para avaliação quantitativa do *software* desenvolvido em *Python*

Linguagem de Programação *Python*

Descrição da atividade:
Vocês utilizarão sua criatividade e conhecimentos adquiridos da linguagem de programação *Python* para criar um jogo de perguntas e respostas sobre um tema de interesse do grupo. O programa deverá conter:

- no mínimo cinco perguntas;
- três tentativas para acerto de cada questão;
- pontuação apresentada ao final.

Critérios para avaliação:

Categoria	4	3	2	1
Número de questões e alternativas para acerto	5 ou mais	4	2	1
Número de tentativas	5 ou mais	4	2	1
Score (pontuação final)	Implementado e eficaz	Implementado e não eficaz. Porém, alunos demonstraram entendimento sobre a estrutura	Implementado e não eficaz. Porém, sem entendimento da estrutura	Não implementado
Clareza do código	Código claro e eficaz	Código, em sua maioria, claro e eficaz	Código não é claro ou não é eficaz	Código não claro e não eficaz
Criatividade	Jogo interessante e divertido	Jogo interessante e divertido. Porém, faltou criatividade em alguns elementos	Houve a tentativa de tornar o jogo interessante e divertido, o que não aconteceu	Jogo apresenta falta de criatividade