Um Modelo Lúdico para o Ensino de Conceitos de Programação de Computadores

Title: A Ludic Model for Teaching Computer Programming Concept

Almir D. V. Santiago
Universidade Salvador – UNIFACS
david.valente.santiago@gmail.com

Artur H. Kronbauer
Universidade Salvador – UNIFACS
arturhk@gmail.com

Resumo

As matérias introdutórias de programação de computadores têm sido fator de desmotivação para os alunos universitários iniciantes dos cursos de computação e engenharias. Neste contexto, diversas metodologias lúdicas de ensino têm sido apresentadas com o intuito de mitigar o problema. Contudo, identificou-se a possibilidade de explorar propostas lúdicas de ensino para conceitos computacionais que ainda não foram bem explorados na bibliografia científica. Este artigo descreve o processo de construção e validação de um modelo lúdico para o ensino do conceito de algoritmos recursivos. O modelo visa fornecer um mecanismo alternativo de visualização algorítmica por meio do uso de metáforas lúdicas. O modelo foi validado por meio de um estudo experimental com 38 alunos universitários iniciantes dos cursos de computação e engenharias da Universidade Salvador. No estudo foram utilizadas as avaliações quantitativa e qualitativa, como forma de obter os dados indicadores para a proposta. Os resultados quantitativos indicam que a utilização da metodologia lúdica visual forneceu aos alunos um melhor esclarecimento acerca do conceito de recursividade. Os resultados qualitativos indicam que houve satisfatório nível de aceitação da proposta. Entendemos, com base nos resultados, que as metáforas lúdicas possuem potencial metodológico para o ensino de conceitos de programação de computadores nas universidades. Sugerimos a integração da metodologia lúdica visual baseada em metáforas com a metodologia de ensino tradicional baseada em código textual.

Palavras-Chave: Ensino de Programação - Metodologias Lúdicas - Psicologia da Programação - Recursividade.

Abstract

The introductory computing subjects have been a factor of demotivation for the undergraduate students of computing and engineering courses. In this context, several ludic teaching methodologies have been presented in order to mitigate the problem. However, we have identified the possibility of exploring ludic teaching proposals for computing concepts that have not yet been well explored in the scientific literature. This article describes the process of constructing and validating a ludic model for teaching the concept of recursive algorithms. The model aims to provide an alternative mechanism of algorithm visualization through the use of ludic metaphors. The model was validated through of an experimental study with 38 undergraduate students in computer science and engineering of Universidade Salvador. In the study, we used quantitative and qualitative assessments as a way of obtaining the indicator data for the proposal. The quantitative results indicate that the use of visual ludic methodology made the students better understand the concept of recursion. The qualitative results indicate that there was a satisfactory level of acceptance of the proposal. We understand, based on the results, that ludic metaphors have great methodological potential as teaching tool for the of computing concepts in universities. We suggest the integration of the visual ludic methodology based on metaphor with the traditional teaching methodology based on textual code.

Keywords: Programming Teaching - Ludic Methodologies - Psychology of Programming - Recursion.

1 Introdução

A problemática do ensino de programação de computadores nas universidades é um assunto que tem sido alvo de discussões em diversos trabalhos científicos recentes, tais como, Amaral et al. (2015), Ambrosio et al. (2011), Eagle e Barnes (2008), Galdino et al. (2015), Hernandez et al. (2010) e Ramos et al. (2015). Tais trabalhos reconhecem que a programação é um assunto complexo e envolve uma grande quantidade de conhecimentos. Além disso, eles relatam grande preocupação com os elevados índices de desistência existentes nos cursos de computação.

A discussão na área do ensino/aprendizagem de programação de computadores reconhece que as matérias introdutórias de programação são responsáveis pelas altas taxas de desistência, já nos primeiros semestres. A quantidade de alunos que desistem dos cursos de computação após o contato com as primeiras matérias da área chega a 40% (Eagle e Barnes, 2008). Galdino et al. (2015) apontam a preocupação existente com estes elevados índices de evasão, e os atribuem às dificuldades enfrentadas pelos iniciantes com as matérias de lógica de programação. Além disso, existem evidências de que grande parte destes alunos não conseguem escrever programas razoáveis após dois semestres com matérias de programação e que os livros didáticos abordam apenas processos de programação sem a devida contextualização do problema (Caspersen e Bennedsen, 2007).

De maneira resumida, o estudo da bibliografia científica sobre o ensino/aprendizagem de programação de computadores apresentou evidências de que:

- Uma metodologia de ensino de conceitos de programação de computadores baseada puramente em sintaxe dificulta o processo de aprendizado dos alunos iniciantes.
- Os alunos são desestimulados devido às dificuldades com as matérias introdutórias de programação, pois não conseguem criar os corretos modelos mentais acerca do conhecimento que está sendo aprendido.
- Existe a necessidade de novas propostas de metodologias instrucionais, as quais não sejam puramente baseadas em código textual.

A partir da identificação da problemática em questão, percebeu-se a oportunidade de trabalhar em propostas direcionadas a facilitar o aprendizado de conceitos introdutórios de programação de computadores. Desta forma, o principal objetivo deste trabalho é propor a criação e validação de uma ferramenta de visualização, direcionada aos alunos, para ilustrar os aspectos cognitivos dos principais conceitos inerentes ao aprendizado de lógica de programação.

A contribuição esperada é disponibilizar aos alunos uma abordagem baseada em metáforas lúdicas visuais, que possa auxiliar a abordagem textual amplamente utilizada nos dias atuais. No lugar de propor a compreensão de algoritmos apenas por meio de resolução de exercícios que utilizam a sintaxe de linguagem de programação, propomos ao aluno uma abordagem metafórica baseada na ludicidade. Segundo nossa linha de pesquisa e nossos trabalhos correlatos (Covington e Benegas, 2005) e (Eagle e Barnes, 2008), esta abordagem permite que os alunos compreendam os conceitos de forma mais fácil, possibilitando o acompanhamento da execução da resolução do algoritmo de maneira visual.

Neste contexto, este artigo descreve a abordagem visual utilizando os conceitos de recursividade e valida a sua eficiência com a execução de um experimento prático, contemplando alunos iniciantes na área de programação.

Na próxima seção, serão discutidos alguns trabalhos correlatos a esta proposta. Na seção 3, serão apresentados os aspectos do modelo proposto. Em seguida, serão apresentadas as implementações do modelo em três algoritmos recursivos. Na seção 5, será mostrado o processo metodológico utilizado. Na seção 6, serão apresentados os resultados obtidos no estudo

experimental. Por fim, na seção 7, serão relatadas as conclusões sobre os resultados do estudo experimental, bem como as propostas para a continuidade deste trabalho.

2 Trabalhos Relacionados

Diversos trabalhos científicos relatam iniciativas que implementam métodos para facilitar o processo de ensino/aprendizagem de conceitos de programação de computadores. Para um aprofundamento mais amplo, podem ser destacadas três revisões sistemáticas Caldeira e Vilela (2016), Francisco et al. (2016) e Zanetti et al. (2016) publicadas no XXVII Simpósio Brasileiro de Informática na Educação (SBIE 2016).

A primeira foi proposta por Zanetti et al. (2016) que faz um levantamento sobre a utilização do termo Pensamento Computacional e sua aplicação para o ensino de programação de computadores.

A segunda revisão sistemática, foi elaborada por Francisco et al. (2016) e traz a discussão da utilização de Juiz Online no ensino de computação e como estas ferramentas podem automatizar o *feedback* oferecido aos alunos e tornar o trabalho do professor mais produtivo. Nesta mesma área de pesquisa, Galvão et al. (2016) analisa a utilização de um Juiz Online mesclado a aulas presenciais. A conclusão dos autores é que a ferramenta possibilitou que os estudantes pratiquem mais exercícios de programação, com *feedback* imediato, proporcionando um aumento na taxa de aprovação.

A terceira revisão sistemática, desenvolvida por Caldeira e Vilela (2016), aborda a utilização de Ambientes Virtuais de Aprendizado (AVA) para auxiliar no ensino de Programação de Computadores. Como resultado os autores relatam as diferenças e os recursos disponíveis nos AVAs para auxiliar o aprendizado de Lógica de Programação.

Outro trabalho que merece destaque foi proposto por Oliveira et al. (2016), que apresenta uma estratégia baseada em técnicas de *clustering* e análise de componentes principais para reconhecimento automático de boas soluções, predefinidas pelo professor e por meio de um sistema de rubricas, em exercícios textuais de programação criados pelos alunos. Segundo os autores, a proposta não facilita apenas o trabalho do professor no processo de correção dos exercícios, mas também proporciona aos alunos maior clareza acerca do método avaliativo utilizado.

O artigo de Mattos et al. (2016) analisa a utilização do Scratch para o aprendizado de programação e seus benefícios para incentivar pessoas do sexo feminino a aderirem aos cursos de computação. O Scratch é uma linguagem de programação gráfica desenvolvida pelo MIT, especialmente para crianças, que oferece, aos pequenos programadores, um ambiente de desenvolvimento acolhedor que permite criar animações, histórias interativas ou jogos em um *browser*.

Outra linha de pesquisa que tem se apresentado promissora em propostas para facilitar o ensino/aprendizagem de programação são as metodologias lúdicas. O uso de abordagens lúdicas foi observado em diversos trabalhos na bibliografia científica.

Na revisão sistemática realizada por Ramos et al. (2015) é apontada a redução do índice de evasão dos alunos iniciantes em computação decorrente do uso de jogos digitais. Os autores afirmam que a abordagem lúdica funcionou como fator motivacional para o ensino.

Amaral et al. (2015) realizaram um estudo experimental sobre o uso de uma plataforma denominada Robocode como ferramenta de apoio ao ensino de algoritmos e programação para alunos do ensino médio. Eles chegaram à conclusão de que a ferramenta se mostrou como

facilitadora para o processo de aprendizagem, pois o uso da ferramenta lúdica aumentou o interesse dos alunos pelo assunto.

Outro trabalho científico apresenta um jogo denominado Wu's Castle (Eagle e Barnes, 2008), que propõe o uso de metáforas lúdicas para o ensino de conceitos de programação (Figura 1). Os autores do trabalho propuseram metáforas para o ensino dos conceitos computacionais de *arrays* (lado esquerdo da Figura 1) e estruturas de repetição (lado direito da Figura 1). Os alunos, por meio da manipulação de um personagem, executam as ações referentes às execuções algorítmicas impostas pelos respectivos conceitos computacionais, conseguindo, desta forma, melhor compreendê-los. Os autores concluíram que os alunos que aprenderam por meio das metáforas lúdicas obtiveram ganhos significativos de aprendizado quando comparados ao grupo de controle.

Figura 1: Metodologia lúdica para o ensino de arrays e estruturas de repetição (Eagle e Barnes, 2008).

Covington e Benegas (2005) propuseram uma mudança da abordagem do ensino de programação de computadores baseada em sintaxe para uma abordagem baseada no desenvolvimento de modelos mentais sobre padrões de programação. Os autores criticam a abordagem vigente para o ensino de programação, denominada por eles de *syntax-driven*, e sugerem uma nova, denominada *schema-driven*.

A principal prerrogativa para a proposta de uma mudança de abordagem de ensino, segundo os autores, é que a *syntax-driven* foca apenas em modelos textuais de programação, quando deveriam desenvolver nos alunos a habilidade de reconhecer padrões de resolução de problemas do mundo real. Os autores concluíram que o ensino da sintaxe só deve ser aplicado quando o aluno já tiver desenvolvido a capacidade de reconhecer e resolver problemas. Dessa forma, a sintaxe poderia ser contextualizada em um domínio mais específico e bem conhecido.

3 O Modelo Proposto

A proposta deste trabalho teve o objetivo de especificar a construção de uma ferramenta que substituísse a linguagem textual por uma linguagem metafórica baseada em modelos visuais, a ser aplicada como metodologia lúdica para o ensino de conceitos de programação de computadores.

Dentre os diversos conceitos introdutórios, inerentes aos cursos superiores de computação, foi escolhido o da recursividade, devido à sua complexidade de aprendizado por parte dos alunos iniciantes em programação, conforme afirma Alaoutinen e Smolander (2010) (Figura 2).

Figura 2: Dificuldades dos conceitos computacionais. Adaptado de Alaoutinen e Smolander (2010).

3.1 A Complexidade do Algoritmo Recursivo

Segundo Odifreddi (1999), o conceito de algoritmo recursivo é uma generalização da noção matemática de recursividade, a qual tem sua origem na Teoria dos Números. A Figura 3 ilustra uma metáfora para a percepção deste conceito no mundo real. Nela a recursividade aparece como uma série de imagens aninhadas, que representam a repetição infinita dos reflexos consecutivos entre dois espelhos.

Em programação este conceito é observado nas funções recursivas, que chamam a si mesmas, direta ou indiretamente, no mínimo uma vez (Guimarães e Lages, 2012). A Equação 1 demonstra matematicamente o conceito de função recursiva aplicado na resolução do fatorial de um número inteiro.

O Código 1 ilustra uma implementação em PORTUGOL para a resolução computacional do cálculo fatorial usando uma função recursiva. Nela, a função FAT recebe um valor N, que representa o valor cujo fatorial será calculado. A partir de então, a função FAT será chamada de forma recursiva por meio da passagem do valor N decrementado (N - 1). Este processo se repetirá até que a condição de parada seja satisfeita, que neste caso acontece quando N é igual a zero.

Figura 3: Recursividade entre espelhos (Inception, 2010).

$$n! = \begin{cases} n(n-1)!, & n > 1 \\ 1, & n = 0 \end{cases} \quad (1)$$

```

FAT(N) inteiro;
inteiro: N;
início
    se N = 0 então FAT ← 1;
    senão FAT ← N * FAT (N - 1);
    fim se;
fim;

```

Código 1: Implementação fatorial recursiva. Adaptado de Guimarães e Lages (2012).

Supondo que seja submetido inicialmente o valor '3' à função recursiva ilustrada no Código 1, se espera como resultado o valor '6', pois o fatorial do número três é igual a "3 x 2 x 1". Isso é logicamente correto e fácil de verificar matematicamente. Contudo, para um melhor entendimento do fluxo de execução dessa função, vamos analisar os estados de cada chamada recursiva, a fim de identificar como estas sucessivas multiplicações são realizadas e como elas resultam no valor final.

A Figura 4 ilustra o processo de resolução do fatorial de '3' pelo algoritmo ilustrado (Código 1). Nela é possível perceber, ao lado esquerdo, que as chamadas recursivas criam uma estrutura hierárquica de quatro níveis, representando em cada um deles uma chamada recursiva subsequente. Em cada chamada o valor n , inicialmente igual a '3', vai sendo decrementado, até que a condição de parada " $n = 0$ " seja satisfeita. Quando o algoritmo alcança a condição de parada não há mais chamadas recursivas e o fluxo de execução passa a retroagir, retornando valores para os fluxos de maior nível. No lado direito da Figura 4 é possível perceber o mecanismo utilizado pela recursividade para encontrar o resultado do fatorial. O valor de retorno de cada chamada recursiva será multiplicado pelo valor de n das chamadas de nível superior, até a obtenção do valor '6'.

Esta análise do fluxo de execução do algoritmo fatorial constata o que fora afirmado por Wilcocks e Sanders (1994), que, apesar de possuírem códigos visivelmente simples e enxutos, os algoritmos recursivos guardam grande complexidade na maneira como são executados, pois criam uma estrutura hierárquica durante sua execução. Os autores relatam que a recursividade é um conceito dinâmico e que os métodos instrucionais utilizados para o seu ensino são inadequados. Eles ainda salientam sobre a necessidade de desenvolver no aluno um senso abstrato para que se possa visualizar mentalmente o fluxo de execução e controle recursivo.

Figura 4: Execução do cálculo fatorial recursivo.

É importante observar que, no exemplo do fatorial ilustrado na Figura 4, obtivemos uma estrutura com apenas quatro chamadas recursivas, mas, a depender do valor submetido e da condição de parada, o tamanho dessa hierarquia pode aumentar significativamente.

Outro fator que impacta na complexidade da estrutura hierárquica das chamadas recursivas é a quantidade de chamadas simultâneas, que é representada pela quantidade de chamadas recursivas dentro da função. O Código 2 apresenta um trecho de código escrito em *c++* que ilustra esta situação. Nela é possível observar que a chamada recursiva se encontra repetida três vezes no corpo da função, sendo que cada uma delas resultará em uma nova chamada com outras três chamadas recursivas.

```

01 int level = 1;
02 int stop_condition = 3;
03 void execute_recursion(int level, string side)
04 {
05     cout << "Level = " << level << ", Side = " << side << endl;
06     if(level >= stop_condition) return; //condição de parada
07     level++;
08     execute_recursion(level,"left"); //primeira chamada recursiva
09     execute_recursion(level,"mid"); //segunda chamada recursiva
10     execute_recursion(level,"right"); //terceira chamada recursiva
11 }

```

Código 2: Chamadas recursivas simultânea

Diferente das chamadas recursivas simples, as simultâneas criam uma estrutura hierárquica em forma de árvore, as quais apresentam um comportamento de rápido crescimento ao longo dos níveis de profundidade. A Figura 5 ilustra a hierarquia criada pelo algoritmo demonstrado no Código 2. Nela é possível observar como o número de chamadas é multiplicado por três a cada nível da recursão. Salienta-se que esta estrutura possui treze chamadas para três níveis de recursão. Caso a condição de parada fosse ‘6’, a quantidade de chamadas seria “243”. Se fosse ‘9’, teríamos 6.561 chamadas. Isso denota o crescimento exponencial que impõem as chamadas recursivas simultâneas.

Figura 5: Árvore de chamadas recursivas.

3.2 Caracterização do Modelo

Uma vez compreendida a complexidade da navegação hierárquica imposta pela recursividade, pode-se compreender a relevância da proposta deste trabalho. O modelo proposto neste artigo sugere uma forma simplificada de observar o fluxo de execução dos algoritmos recursivos. A proposta consiste em uma abordagem lúdica, onde o aluno, por meio do controle de um personagem, executa as etapas algorítmicas de maneira puramente visual, ao passo que observa o nível hierárquico no qual se encontra e a representação visual do problema modelado.

A interface do modelo (Figura 6) fornece quatros perspectivas distintas:

- Game – fornece a abstração lúdica, que substitui o código fonte textual do algoritmo.
- Canvas – fornece a representação gráfica modelada pelo algoritmo aplicado ao modelo. Tem o objetivo de ilustrar como as operações modelam o problema a cada etapa da execução recursiva.
- Map – fornece o mapa hierárquico das chamadas recursivas, permitindo que o aluno consiga identificar o estágio atual da resolução algorítmica.

- Info – fornece informações textuais acerca das variáveis que controlam o fluxo recursivo, além de outras informações a serem consideradas relevantes.

Figura 6: Modelo lúdico proposto.

Estas perspectivas devem ser sincronizadas, fornecendo ao aluno a possibilidade de observar em tempo real a navegação (o nível hierárquico da repetição) e a imagem (o que está sendo construído), ao passo que ele interage com o personagem.

O principal objetivo do modelo proposto é desenvolver no aluno os corretos modelos mentais acerca do fluxo de execução dos algoritmos recursivos. Outros objetivos são:

- Fazer com que o aluno complete todas as etapas do algoritmo por meio da manipulação do personagem (abordagem lúdica).
- Permitir que o aluno enxergue o funcionamento da recursividade sob outras perspectivas, além da textual.
- Fornecer *feedback* imediato do fluxo de execução do algoritmo sobre as ações do usuário.

3.2.1 Perspectiva Game

A Perspectiva Game representa a abstração do código de uma função recursiva. Ela tem o objetivo de permitir que o aluno controle o personagem, que representa a forma lúdica com a qual ele interage com o fluxo de execução do algoritmo. A Figura 7 ilustra o cenário proposto pela perspectiva, que deve ser constituída por portais, sendo apenas um na parte superior, representando a entrada na função recursiva e no mínimo um na parte inferior, representando as chamadas recursivas.

Figura 7: Perspectiva Game.

O fluxo de execução consiste em adentrar por quaisquer portais localizados na parte inferior e, pelo portal superior, retornar ao mesmo cenário, simulando o fluxo de repetições impostas pelas funções recursivas. Ao passo que o aluno executa as etapas, são apresentadas alterações nas outras perspectivas, com o objetivo de demonstrar para ele outros pontos de vista sobre a resolução do problema modelado pelo algoritmo recursivo.

3.2.2 *Perspectiva Canvas*

A Perspectiva Canvas exibe a representação gráfica modelada pelo algoritmo aplicado ao modelo. Ela deve refletir a saída gerada pelo conceito que está sendo representado por meio do algoritmo. O objetivo é fornecer um mapa claro de como as operações recursivas, controladas pelo usuário, vão modelando o problema em cada etapa da execução.

Vale ressaltar que a representação gráfica deve refletir exatamente o que o algoritmo realiza em cada uma das etapas. Esta representação depende exclusivamente de cada algoritmo. A Figura 8 ilustra algumas saídas gráficas propostas para o algoritmo recursivo de construção do triângulo de Sierpinski.

Figura 8: Perspectiva Canvas.

3.2.3 Perspectiva Map

Esta perspectiva permite que o aluno se familiarize com a navegação hierárquica de chamadas imposta pelos algoritmos recursivos. A Figura 9 ilustra como ela fornece ao aluno a percepção do nível de navegação em que ele se encontra por meio de um mapa hierárquico.

Com base na complexidade do conceito de recursividade, entende-se que este tipo de visualização é imprescindível para que o aluno consiga concluir todas as etapas do algoritmo. Ela evita que ele se confunda durante o processo de execução, dando-o, a todo momento, a consciência de qual etapa recursiva ele se encontra.

Figura 9: Perspectiva Map

3.2.4 Perspectiva Info

A Perspectiva Info fornece, principalmente, informações textuais sobre a variável que controla a condição de parada do fluxo recursivo, além de outras informações algorítmicas relevantes, caso sejam necessárias para o algoritmo.

4 Implementação do Modelo

A implementação foi realizada por meio da criação de três aplicações funcionais, desenvolvidas com a ferramenta Construct 2 (Figura 10), que simulam a resolução recursiva de três problemas matemáticos: (i) o Triângulo de Sierpinski; (ii) a Torre de Hanoi; e (iii) o Cálculo do Fatorial. Todos os três problemas foram implementados com base no algoritmo recursivo para suas resoluções e com base no modelo que foi caracterizado na Seção 3. O principal objetivo desta implementação foi tornar possível avaliar e validar a proposta através de um estudo experimental.

Figura 10: Ferramenta Construct 2.

4.1 Algoritmo do Triângulo de Sierpinski

Segundo Zavala (2007), o triângulo de Sierpinski, é uma figura geométrica fractal, estudada pelo matemático polonês Waclav, cuja construção é realizada por um processo iterativo de divisão de triângulos. A Tabela 1 apresenta a notação matemática para a construção do triângulo de Sierpinski.

Tabela 1: Notação matemática do triângulo de Sierpinski (Zavala, 2007)

Nível	Forma	Número de triângulos	Comprimento de cada lado
0		1	l
1		3	$l \cdot \frac{1}{2}$
2		$9=3^2$	$l \cdot \frac{1}{4} = l \cdot \left(\frac{1}{2}\right)^2$
3		$27=3^3$	$l \cdot \frac{1}{8} = l \cdot \left(\frac{1}{2}\right)^3$
n		3^n	$l \cdot \left(\frac{1}{2}\right)^n$

Com o objetivo de identificar as complexidades inerentes ao algoritmo recursivo para a construção do Triângulo de Sierpinski, foi realizada a análise do fluxo de execução do seu código em *c++* (Código 3). Neste contexto, o algoritmo apresentou-se bastante simples, possuindo, em cada etapa de sua execução, os seus objetivos bem definidos.

```

01 void sierpinski(int n,int x1,int y1,int x2,int y2,int x3,int y3)
02 {
03     if (n == 0)
04     {
05         drawLine(x1, y1, x2, y2);
06         drawLine(x2, y2, x3, y3);
07         drawLine(x3, y3, x1, y1);
08     } else {
09         int mx1 = (x1 + x2) / 2;
10         int my1 = (y1 + y2) / 2;
11         int mx2 = (x3 + x2) / 2;
12         int my2 = (y3 + y2) / 2;
13         int mx3 = (x3 + x1) / 2;
14         int my3 = (y1 + y3) / 2;
15         // chamada recursiva
16         sierpinski(n - 1, x1, y1, mx1, my1, mx3, my3);
17         // chamada recursiva
18         sierpinski(n - 1, mx1, my1, x2, y2, mx2, my2);
19         // chamada recursiva
20         sierpinski(n - 1, mx3, my3, mx2, my2, x3, y3);
21     }
22 }

```

Código 3: Algoritmo, escrito em *c++*, para a construção do triângulo de Sierpinski.

A partir das coordenadas de três pontos de um triângulo (linha 1 do Código 3) o algoritmo calcula os pontos médios de cada lado do triângulo (linhas 9 a 14 do Código 3) com o objetivo de obter as coordenadas necessárias para a geração de três novos triângulos. Por fim, são realizadas três chamadas recursivas (linhas 16, 18 e 20 do Código 3), às quais são passadas as coordenadas obtidas e que irão subdividir o problema em três novos problemas idênticos nas etapas seguintes. A Figura 11 ilustra como o algoritmo de Sierpinski subdivide o problema

em três novos problemas idênticos. Os pontos médios calculados pelo algoritmo servem de base para a geração das extremidades dos novos triângulos.

Figura 11: Subdivisão dos triângulos de Sierpinski

A partir desta análise, observou-se que a complexidade deste algoritmo decorre da quantidade de chamadas recursivas presentes no corpo da função. O fato do algoritmo conter três chamadas impõe uma divergência do fluxo recursivo, criando três “universos simultâneos” de resolução do problema, conforme foi ilustrado na Figura 5.

A Figura 12 ilustra a proposta de Sierpinski implementada. Conforme descrito na caracterização do modelo (Seção 3.2).

Figura 12: Implementação do algoritmo de Sierpinski

A Perspectiva Map apresenta o nível recursivo no qual o jogador se encontra. A Perspectiva Info apresenta o valor da variável n , a qual representa o tamanho do problema, informando ao aluno a existência de uma condição de parada. A Perspectiva Game apresenta a abstração lúdica proposta para o algoritmo. A porta superior abstrai a entrada na função recursiva (linha 1 do Código 3) e as três portas inferiores abstraem as chamadas recursivas existentes no algoritmo (linhas 16, 18 e 20 do Código 3). A Perspectiva Canvas apresenta o

posicionamento dos pontos dos triângulos, bem como, o processo de desenho dos triângulos com base nas interações do aluno com os botões.

Para que o aluno possa perceber o mecanismo utilizado pelo algoritmo, tanto para calcular os pontos médios dos lados do triângulo como para desenhar os triângulos, foi implementado o uso de botões, os quais devem ser ativados pelo personagem, por meio de comando do teclado, para que os cálculos e os desenhos possam ser realizados. A Figura 13 ilustra os possíveis estados dos botões, onde os inferiores representam os comandos de cálculo dos pontos médios dos lados (linhas 9 a 14 do Código 3) e os superiores representam os comandos de desenho dos lados (linhas 5, 6 e 7 do Código 3).

Figura 13: Metáforas para representar os comandos do código.

4.2 Algoritmo da Torre de Hanoi

Segundo Stewart e Eliasmith (2011) a Torre de Hanoi é um problema histórico na área da ciência cognitiva (Figura 14). Ele consiste em três torres (A, B e C), nas quais são colocadas um determinado número n de discos, empilhados por ordem de tamanho. O objetivo é transferir, com o menor número possível de movimentos, os discos da torre A para a torre C, sendo que para isso seja possível mover apenas um disco por vez, e sem que nenhum disco maior fique sobre um disco menor.

Figura 14: Notação da torre de Hanoi.

O algoritmo da resolução da Torre de Hanoi utilizado como referência para a implementação no modelo emprega a “impressão no console” como forma de analogia ao movimento dos discos (linhas 5 e 8 do Código 4). A função (linha 1 do Código 4) recebe quatro parâmetros: (i) n – indica a quantidade de torres, e consequentemente o tamanho do problema; (ii) origem – indica a torre da qual o disco será retirado (A, B ou C); (iii) neutro – indica a torre que será ignorada (A, B ou C); e (iv) destino – indica a torre que receberá o disco movido (A, B ou C).

```

01 void Hanoi(int n, char origem, char neutro, char destino)
02 {
03     if(n == 1)
04     {
05         cout << "mova disco " << n << " de " << origem << " para " <<
06 destino;
07     } else {
08         Hanoi(n-1, origem, destino, neutro);
09         cout << "mova disco " << n << " de " << origem << " para " <<
10 destino;
11         Hanoi(n-1, neutro, origem, destino);
12     }
13 }

```

Código 4: Algoritmo, escrito em c++, para a resolução da Torre de Hanoi

De maneira diferente ao algoritmo de Sierpinski, este possui apenas duas chamadas recursivas no corpo de sua função. Isto implica em uma estrutura hierárquica de complexidade um pouco mais reduzida.

Outro aspecto importante sobre o processo da resolução da Torre de Hanoi diz respeito a relação existente entre os discos e os níveis recursivos da sua estrutura hierárquica. A Figura 15 ilustra este fato, onde é possível perceber que a quantidade de níveis da estrutura é sempre igual a quantidade de discos utilizados no problema, estando cada um deles associados ao seu respectivo nível. O nível um (1) representando o maior disco e o último nível representando o menor disco. Isto implica que, obrigatoriamente, os discos são movidos apenas em seus respectivos níveis. Outra regra inerente a este aspecto é que a quantidade de nós em cada nível representa sempre a quantidade de vezes que o disco, referente àquele nível, é movido durante a resolução do problema.

Figura 15: Relação entre discos e algoritmo

Contudo, não são apenas estes aspectos que tornam a implementação recursiva de Hanoi complexa. A grande complexidade é observada quando são analisadas as chamadas recursivas (linhas 7 e 9 do Código 4), onde há a troca de posição entre as variáveis “origem”, “destino” e “neutro”. Na primeira chamada recursiva há a troca entre as variáveis “neutro” e “destino”. Na segunda há a troca entre as variáveis “origem” e “neutro”.

A princípio pode este parecer um aspecto ignorável, mas faz toda a diferença no entendimento do algoritmo. Estas trocas são responsáveis por redefinir, em cada etapa algorítmica, as movimentações dos discos e resolver o problema. A Figura 16 mostra os estados de cada etapa da resolução do problema de Hanoi, considerando o tamanho n do problema igual ao valor ‘3’.

Figura 16: Troca de parâmetros na resolução da torre de Hanoi

Para a implementação da resolução da Torre de Hanoi foram considerados três discos, como sendo o tamanho n do problema. A Figura 17 ilustra a proposta implementada no modelo. Seguindo a mesma lógica de Sierpinski, a Perspectiva Map apresenta o nível recursivo no qual o jogador se encontra. A Perspectiva Info apresenta o valor da variável n , que representa o tamanho do problema. A Perspectiva Game apresenta a abstração lúdica proposta para o algoritmo, que consiste em duas portas na parte inferior, para representar as chamadas recursivas (linhas 7 e 9 do Código 4) e um botão, representando a ação mover disco (linhas 5 e 8 do Código 4).

Figura 17: Implementação do algoritmo de Hanoi

Para ilustrar a existência de uma troca de parâmetros durante as chamadas recursivas foram criados na Perspectiva Canvas três marcadores, “origem”, “neutro” e “destino” (Figura 18). Eles representam, respectivamente, o marcador para a torre da qual será retirado o disco, o marcador para torre que será ignorada e o marcador para a torre que receberá o disco.

Figura 18: Troca de parâmetros no algoritmo de Hanoi

Para que o aluno consiga relacionar as chamadas às trocas dos parâmetros, foram colocados sob as portas indicadores para informar quais parâmetros são invertidos naquela chamada recursiva. Dessa forma, entende-se que é possível observar imediatamente a relação existente entre a troca de parâmetros, na Perspectiva Canvas, e cada chamada recursiva, na Perspectiva Game. Para mostrar ao aluno a relação existente entre os discos e os níveis hierárquicos da estrutura foram utilizados números nos discos, os quais fazem referência direta a seus respectivos níveis.

4.3 Algoritmo do Cálculo Fatorial

Segundo Laureano (2008), o fatorial de um determinado número n é o produto de todos os números inteiros entre 1 e n . A equação (2) ilustra o exemplo do fatorial do número '4'.

$$\begin{aligned}
 0! &= 1 \\
 1! &= 1 * 0! = 1 \\
 2! &= 2 * 1! = 2 \\
 3! &= 3 * 2! = 6 \\
 4! &= 4 * 3! = 24
 \end{aligned}
 \tag{2}$$

Diferente da implementação de Sierpinski e de Hanoi, o algoritmo fatorial recursivo (Código 5) apresenta-se muito mais simples. Tanto no que diz respeito à quantidade de código, mais enxuto, como no que diz respeito à sua estrutura hierárquica de chamadas. A presença de apenas uma chamada recursiva no corpo da função implica em uma estrutura hierárquica de menor complexidade do que as estruturas dos algoritmos de Sierpinski e Hanoi.

```

01 int fatorial(int n)
02 {
03     if(n == 0)
04         return 1;
05     else
06         return n * fatorial(n-1);
07 }

```

Código 5: Algoritmo recursivo, em c++, para o cálculo Fatorial.

Além da simplicidade textual, o fato de não haver necessidade de monitorar chamadas simultâneas pode aparentar, em uma primeira análise, que se trata de um algoritmo de fácil compreensão. Contudo, quando se observa a ordem de execução das etapas do algoritmo fatorial, fica clara a grande complexidade imposta por ele: a presença da chamada recursiva dentro de uma operação algébrica de multiplicação, apresentada na linha 6 do Código 5. Neste contexto, a chamada recursiva realiza o papel de um segundo operando da operação matemática. A Tabela 2 ilustra este fato, onde é possível observar que o resultado da expressão, cujo segundo operando é a chamada recursiva (“fatorial(n-1)”), é incerto em um primeiro momento, pois ele está diretamente relacionado a uma nova chamada recursiva e à consequente repetição do mesmo procedimento.

Tabela 2: Chamada de função recursiva como um segundo operando de uma expressão algébrica

	operando 1	operador	operando 2	resultado
Exemplo operação algébrica	2	* (multiplicação)	2	4
Expressão do algoritmo recursivo	N	* (multiplicação)	fatorial(n-1)	? (indefinido)

Este fato impede que o aluno consiga identificar o valor da expressão, que somente poderá ser calculada após todas as chamadas recursivas serem realizadas. A Figura 19 explica melhor esta complexidade, por meio do processo de resolução algorítmica para o fatorial do número ‘3’. Nela é possível observar como o valor da expressão só pode começar a ser calculado após a última chamada recursiva e, conseqüentemente, a sua condição de parada ser alcançada. A partir de então, o valor começa a ser retornado para as etapas anteriores e o resultado pode ser encontrado.

Conclui-se, portanto, que a complexidade da implementação recursiva da resolução fatorial reside na incapacidade de se reconhecer o resultado da expressão até que a última chamada recursiva seja alcançada, juntamente com a satisfação de sua condição de parada. A Figura 20 ilustra como foi implementada a representação da resolução algorítmica fatorial recursiva.

Figura 19: Processo de resolução fatorial do valor '3'.

Figura 20: Implementação do algoritmo do cálculo fatorial.

A implementação segue análoga às anteriores. Na Perspectiva Canvas são exibidas as etapas algébricas do cálculo fatorial, para que o aluno possa acompanhar a resolução do problema ao passo que interage com a aplicação. A Perspectiva Map apresenta a estrutura hierárquica inerente a execução do algoritmo. A Perspectiva Info apresenta o valor da variável n , a qual representa o tamanho do problema.

Para ilustrar o aspecto da impossibilidade de se encontrar o valor da expressão algébrica durante a execução das etapas, é apresentada na parte inferior da Perspectiva Game a expressão algébrica presente no algoritmo fatorial (Figura 21), onde o segundo operador é representado pelo portal que direciona o personagem para as etapas seguintes do processo de resolução, fazendo alusão à linha 6 do Código 5. O caractere '?' ao lado direito da Figura 21 representa a incerteza sobre o resultado da expressão para aquele nível recursivo.

Figura 21: Portal com chamada recursiva.

Uma vez alcançada a condição de parada, dá-se início ao processo de retorno de valores, conforme ilustrado na Figura 19. Entende-se que para ilustrar este aspecto, faz-se necessário demonstrar para o aluno que a resolução das expressões em cada etapa depende de um valor a ser “carregado” a partir de uma etapa posterior, o qual deve substituir a chamada recursiva como operando matemático, permitindo, desta forma, a resolução do problema.

Figura 22: Metáfora do carregamento de valores.

A Figura 22 ilustra a metáfora proposta, na qual os valores a serem retornados são carregados pelo personagem como “caixas”, criando uma metáfora de transporte de valores entre as etapas. Entende-se que desta forma o aluno consegue enxergar o mecanismo utilizado pelo fluxo de execução recursiva para calcular o resultado da expressão.

O objetivo da metáfora é fazer com que o aluno enxergue de forma clara o mecanismo utilizado pela implementação recursiva para calcular o fatorial de um número, criando assim uma maneira visual de criação dos modelos mentais sobre o fluxo de resolução para este tipo de algoritmo.

É importante salientar que estas três metáforas, bem como a forma com a qual elas foram especificadas, originaram-se de um processo de análise do problema. Foram elencados os principais aspectos e características durante a observação da execução destes algoritmos

recursivos. Em seguida, o processo de construção de nossa proposta consistiu em identificar os elementos visuais do mundo real, que fossem de mais simples compreensão, para representar e explicar as complexas características identificadas.

Portanto, concluímos que o esforço necessário para estender este modelo aos outros algoritmos recursivos, bem como aos outros conceitos computacionais, envolve um esforço de análise e extração das características apresentadas por eles, além da consequente identificação de elementos visuais do mundo real que possam representá-los com maior simplicidade.

5 Metodologia

Nesta seção estão contidos os procedimentos metodológicos empregados durante todo o processo de pesquisa realizado neste trabalho.

5.1 Quadro Metodológico

Segundo Gil (2000), as pesquisas científicas podem ser classificadas segundo dois critérios: quanto aos objetivos e quanto aos procedimentos técnicos. Tendo em vista os objetivos deste trabalho, optou-se por utilizar os tipos descritos na Tabela 3.

Tabela 3: Quadro metodológico.

Quanto aos objetivos	Quanto aos procedimentos técnicos
Exploratória: Compreender a psicologia da programação.	Levantamento bibliográfico: Identificar os trabalhos sobre o uso da psicologia cognitiva na programação.
Explicativa: Explicar as dificuldades no aprendizado de programação.	Estudo experimental: Avaliar o modelo criado para o ensino de conceitos de programação de computadores.

5.2 Etapas da Pesquisa

O processo de pesquisa consistiu nas etapas ilustradas na Figura 23. A primeira etapa refere-se a uma revisão bibliográfica preliminar sobre o ensino de programação, onde foi percebida a associação de algumas iniciativas com teorias relacionadas à psicologia cognitiva. Essa percepção motivou a pesquisa sobre o referido tema. Tais estudos formaram o alicerce para a proposta de um mapeamento sistemático acerca da psicologia cognitiva no ensino de conceitos de programação de computadores. Os resultados deste mapeamento serviram como fundamentação teórica durante a especificação do modelo lúdico. A etapa final do trabalho consistiu em avaliar e validar o modelo por meio da realização de um estudo experimental.

Figura 23: Etapas da pesquisa

5.3 Planejamento do Estudo Experimental

O estudo experimental teve como objetivo avaliar o modelo lúdico proposto. Para tanto, seu planejamento foi dividido em seis fases distintas, tomando como base as diretrizes propostas pelo framework DECIDE (Sharp et al., 2007), que norteou a especificação dos passos realizados durante todas as fases do experimento.

5.3.1 Determinar o Objetivo da Análise

O objetivo da análise foi obter informações sobre a eficácia de uma metodologia lúdica fundamentada na psicologia cognitiva para o ensino de conceitos de programação de computadores.

5.3.2 Explorar Perguntas a Serem Respondidas

- Os alunos enfrentaram dificuldades para entender o funcionamento da ferramenta?
- A ferramenta conseguiu construir os modelos mentais acerca do fluxo de execução dos algoritmos recursivos?
- Ao final do experimento foi possível observar uma melhora geral de desempenho nos testes que pudesse ser atribuída à metodologia proposta?
- Os alunos aprovaram a metodologia como ferramenta de apoio ao ensino de conceitos de programação de computadores?

5.3.3 EsColher Método de Avaliação

A execução do estudo experimental fez uso de dois tipos de avaliações: (i) avaliação quantitativa, realizada por meio do *método pretest-posttest* (Dimitrov e Rumrill, 2003); e (ii) avaliação qualitativa, realizada por meio de questionário formatado em função da escala de Likert (1932).

O método *pretest-posttest* foi utilizado para construir o processo ao qual foram submetidos os participantes do experimento. A Figura 24 ilustra o processo realizado durante o experimento.

Figura 24: Metodologia pretest-posttest.

O pré-teste consistiu em um questionário contendo três questões algorítmicas textuais, referentes aos três algoritmos abordados. O objetivo foi o de averiguar a capacidade dos alunos de identificar o funcionamento do fluxo algorítmico recursivo por meio da metodologia textual tradicional. Em seguida foi aplicada a metodologia lúdica construída, que consistiu na execução dos mesmos três algoritmos, porém não em forma textual, mas em modelos lúdicos visuais. Nesta etapa os alunos tiveram a oportunidade de executar os algoritmos de forma visual por meio das ferramentas construídas. O objetivo foi fornecer aos alunos uma melhor explicação acerca do funcionamento do fluxo recursivo através de uma explicação lúdica. A terceira etapa do experimento (pós-teste) consistiu na reaplicação dos questionários textuais submetidos na primeira etapa (pré-teste), mas desta vez com os valores das questões trocadas.

É importante observar que este método de experimento possui o objetivo de verificar o índice de retenção do conhecimento que os estudantes obtiveram por meio da comparação dos índices de acerto obtidos no pré-teste e pós-testes. Estes índices foram devidamente documentados e utilizados para fundamentar os nossos resultados quantitativos (Seção 6.1).

Com o objetivo de avaliar a opinião dos participantes sobre os diversos aspectos referentes ao experimento *pretest-posttest* (Dimitrov e Rumrill, 2003), foi lhes submetido, como última etapa do processo investigativo, um questionário qualitativo (Análise qualitativa), com o intuito de obter um indicador de qualidade para a metodologia. Esta análise consistiu em um questionário contendo sete questões criadas com base na escala de Likert (1932) e uma (oitava questão) discursiva, que teve o objetivo de avaliar a opinião aberta dos estudantes. As informações obtidas nesta análise foram, também, devidamente documentadas e utilizadas para fundamentar nossos resultados qualitativos (Seção 6.2).

5.3.4 Identificar e Administrar as Questões Práticas

O estudo experimental (Figura 25) foi realizado no laboratório de informática da Universidade Salvador, com a participação de três turmas da disciplina de Estrutura de Dados, dos cursos de Engenharia da Computação, Elétrica e Mecatrônica, contemplando 38 participantes. Os alunos foram divididos em três turmas, de acordo com o turno em que a disciplina foi ministrada. É importante salientar que, de acordo com a matriz curricular dos cursos, o conceito de algoritmos recursivos é abordado pela primeira na disciplina de Estrutura de Dados. Caracterizando, desta forma, uma aprendizagem inicial por parte dos alunos. O laboratório foi previamente preparado com as ferramentas lúdicas desenvolvidas, com o objetivo de validar a abordagem.

Figura 25: Estudo experimental

O início do processo consistiu em uma aula expositiva sobre o conceito de recursividade, com o objetivo de promover um nivelamento de conhecimento entre os participantes do experimento. Foi apresentado o conceito matemático de recursividade e a sua aplicação na resolução computacional de problemas do mundo real por meio dos algoritmos recursivos.

Após a aula deu-se início ao estudo experimental, começando com a submissão do pré-teste aos alunos. Após a aplicação do pré-teste, os alunos foram convidados à participar do experimento lúdico, com as ferramentas criadas, sendo sua execução mediada pelos autores deste trabalho. Em seguida, os alunos foram submetidos ao pós-teste, finalizando a etapa quantitativa.

Por último, foi submetido aos alunos o questionário qualitativo, finalizando o estudo experimental.

5.3.5 *Decidir como Lidar com as Questões Éticas*

O projeto desta pesquisa foi submetido ao Comitê de Ética, identificado na Plataforma Brasil por meio do identificador CAAE: 54856416.3.0000.5033, sendo devidamente aprovado. O experimento foi conduzido preservando o anonimato dos participantes. Foi especificado no Termo de Consentimento Livre e Esclarecido que as informações pessoais dos participantes não seriam divulgadas. Além disso, todos os voluntários possuíam mais de dezoito anos de idade e gozavam de plena capacidade física e mental.

5.3.6 *Estabelecer Forma de Avaliar, Interpretar e Apresentar os Resultados*

As informações geradas com o estudo experimental abrangeram dados quantitativos e subjetivos que, após avaliados e interpretados, geraram gráficos e resultados que serão apresentados e discutidos na próxima seção.

6 Resultados

Para a análise dos resultados quantitativa e qualitativa foram gerados os gráficos apresentados na Figura 26, na Tabela 4 e na Tabela 5.

6.1 Análise Quantitativa

O gráfico ilustrado na Figura 26, mostra as médias dos resultados para todos os 38 alunos participantes do experimento. O contraste apresentado nos índices de acerto entre o pré-teste e o pós-teste sugere o real ganho de desempenho provido pela metodologia.

Figura 26: Resultados da análise quantitativa

A primeira questão (referente ao Triângulo de Sierpinski) apresentou o menor índice de acertos, tendo 7 alunos acertado o pré-teste e 30 o pós-teste. A segunda questão (referente à Torre de Hanoi) apresentou o índice mediano de acertos, tendo 16 alunos acertado a questão no pré-teste e 35 no pós-teste. A terceira questão (referente ao Cálculo do Fatorial) apresentou o maior índice de acertos, 32 alunos acertaram a questão no pré-teste e 37 no pós-teste.

Os resultados apresentados na Tabela 4 mostram o ganho de desempenho promovido pela intervenção proposta pelo estudo experimental. É possível observar que, para todas as questões, ocorreu variações positivas, totalizando, entre as três questões, uma média de 41,23% de ganho de desempenho por parte dos alunos.

Tabela 4: Dados sobre o ganho de desempenho.

Questão	Desempenho Pré-teste	Desempenho Pós-teste	Ganho de desempenho
1ª	18,42%	78,95%	60,53%
2ª	42,11%	92,11%	50,00%
3ª	84,21%	97,37%	13,16%
Média	48,24%	89,47%	41,23%

6.2 Análise Qualitativa

Na análise qualitativa foram submetidas sete perguntas baseadas na escala de Likert (1932), com o objetivo de obter a opinião dos alunos com relação aos aspectos da metodologia, e uma pergunta discursiva (oitava questão), com o objetivo de obter a opinião pessoal dos alunos sobre a proposta:

1. *Dificuldade enfrentada para responder ao pré-teste.*
2. *Dificuldade enfrentada para responder ao pós-teste.*
3. *Dificuldade enfrentada para entender o funcionamento da metodologia proposta.*
4. *Eficácia da metodologia para a compreensão do algoritmo de Sierpinski.*
5. *Eficácia da metodologia para a compreensão do algoritmo de Hanoi.*
6. *Eficácia da metodologia para a compreensão do algoritmo do Cálculo do Fatorial.*
7. *Relevância de uma abordagem que utiliza metáforas lúdicas, no lugar da textual.*
8. *Escreva com suas palavras sobre a proposta lúdica para o ensino de conceitos de programação e o que você mudaria ou adicionaria para melhorá-la.*

A Tabela 5 apresenta os resultados obtidos para cada questão submetida na análise qualitativa. De maneira geral, os resultados demonstram que houve real aceitação da metodologia por parte dos alunos.

Tabela 5: Resultados da análise qualitativa baseada na escala de Likert (1932).

Questão	1-muito difícil	2-difícil	3-razoável	4-fácil	5-muito fácil
1ª	5,3%	47,4%	39,5%	7,9%	0,0%
2ª	0,0%	0,0%	10,5%	57,9%	31,6%
3ª	0,0%	2,6%	23,7%	36,8%	36,8%
Questão	1-muito eficaz	2-pouco eficaz	3-razoável	4-eficaz	5-muito eficaz
4ª	0,0%	2,6%	7,9%	55,3%	34,2%
5ª	0,0%	5,3%	7,9%	47,4%	39,5%
6ª	0,0%	0,0%	0,0%	39,5%	60,5%
Questão	1-nada relevante	2-pouco relevante	3-razoável	4-relevante	5-muito relevante
7ª	0%	0%	2,6%	23,7%	73,7%

No que diz respeito à oitava questão (questão discursiva) foi possível reafirmar a aceitação da metodologia. Os alunos observaram que a metodologia lúdica:

- Torna mais eficiente a compreensão do problema.
- É uma inovação metodológica.
- Torna concreto um conceito abstrato.
- Aumenta o interesse pela programação.
- Tem grande potencial como metodologia complementar a ser implementada nas universidades.
- Deve ser utilizada em conjunto com a abordagem baseada em código.

7 Conclusões e Trabalhos Futuros

No que diz respeito aos pontos positivos do trabalho, os resultados expressos nos gráficos quantitativos do estudo experimental sugerem que houve um real ganho de desempenho para o entendimento dos algoritmos recursivos por parte dos alunos, e que este ganho de desempenho sugere a eficácia da intervenção experimental proposta por este trabalho. Os resultados expressos nos gráficos qualitativos sugerem que houve aceitação da proposta pelos alunos. Portanto, com base nas análises pode-se responder às perguntas definidas para o estudo experimental (Seção 5.3.2):

- Os alunos enfrentaram dificuldades para entender o funcionamento da ferramenta? Resposta – Não. Conforme ilustra a terceira questão da análise qualitativa (Tabela 5), os alunos avaliaram a ferramenta como fácil e muito fácil.
- A ferramenta conseguiu construir os modelos mentais acerca do fluxo de execução dos algoritmos recursivos? Resposta: Sim. Conforme ilustra a quarta, quinta e sexta questões da análise qualitativa (Tabela 5), houve um melhor esclarecimento acerca da lógica de funcionamento dos algoritmos apresentados.
- Ao final do experimento foi possível observar uma melhora geral de desempenho nos testes que pudesse ser atribuída à metodologia proposta? Resposta: Sim. Conforme ilustra a Tabela 4, houve melhora significativa de desempenho dos alunos entre o pré-teste e o pós-teste.
- Os alunos aprovaram a metodologia como ferramenta de apoio ao ensino de conceitos de programação de computadores? Resposta: Sim. Conforme os próprios comentários dos alunos na questão discursiva, houve grande aceitação da metodologia e expectativa

de que seja utilizada como ferramenta de apoio ao ensino de conceitos de programação de computadores nas universidades.

Com relação aos pontos negativos do modelo, as respostas apresentadas para a questão discursiva indicaram que os alunos, apesar de terem observado vantagens no aprendizado por meio de metáforas lúdicas, sentiram a falta da visualização do código fonte. Neste sentido, chegou-se à conclusão de que a visualização simultânea do código é importante para uma melhor compreensão do algoritmo, pois permite que o aluno observe e relacione as metáforas com as etapas algorítmicas descritas no código textual.

No que diz respeito ao objetivo geral de propor uma metodologia lúdica para o ensino de conceitos de programação de computadores, conclui-se que este trabalho cumpriu o seu papel, realizando todas as etapas necessárias para o alcance do objetivo. Quanto à problemática sobre como mitigar os elevados índices de desistência, ocasionadas pelas dificuldades com as matérias introdutórias de programação, conclui-se que este trabalho contribui de forma significativa, mesmo que ainda inicial. Entende-se que a implementação de metodologias lúdicas baseadas em metáforas, aliadas à abordagem textual utilizada atualmente nas universidades, apresenta-se como uma proposta promissora.

As perspectivas para a continuidade desta pesquisa foram idealizadas de acordo com as seguintes etapas: (i) incluir a Perspectiva CODE, na qual o aluno poderá visualizar o código fonte gerado pelo algoritmo; (ii) implementar mais algoritmos recursivos, com o intuito de avaliar a visualização de outros problemas; e (iii) extensão da metodologia aos outros conceitos computacionais, tais como: estruturas de repetição, arrays, gerenciamento de memória, entre outros descritos na bibliografia científica.

No que se refere à metodologia e aos processos de avaliação, pretende-se realizar mais experimentos com alunos universitários, incluindo novos participantes, grupos de controles e utilizar novas técnicas de análise estatística, como, por exemplo, o método Analysis of Variance (ANOVA).

Referências

- Amaral L. R., Silva G. B., Pantaleão E. (2015). Plataforma Robocode como Ferramenta Lúdica de Ensino de Programação de Computadores - Extensão Universitária em Escolas Públicas de Minas Gerais. In: *Anais do XXVI Simpósio brasileiro de informática na educação - SBIE*, Maceió, Brasil, páginas 200-208. doi:[10.5753/cbie.sbie.2015.200](https://doi.org/10.5753/cbie.sbie.2015.200)
- Alaoutinen S., Smolander K. (2010). Student self-assessment in a programming course using bloom's revised taxonomy. In: *Anais do 15th ACM annual conference on innovation and technology in computer science education ITiCSE'10*, Ankara, Turkey, páginas 155-159. . doi:[10.1145/1822090.1822135](https://doi.org/10.1145/1822090.1822135)
- Ambrosio A. P., Costa F. M., Almeida L., Franco A., Macedo J. (2011). Identifying cognitive abilities to improve CS1 outcome. In: *Frontiers in education conference (FIE)*, Rapid City, SD, USA, páginas F3G-1-F3G-7. doi:10.1109/FIE.2011.6142824 [[GS Search](#)]
- Caldeira J., Vilela A. P. (2016). Um Mapeamento Sistemático para auxiliar na escolha de plataformas EAD para o ensino-aprendizagem de Algoritmos e Programação de Computadores. In: *Anais do XXVII Simpósio brasileiro de informática na educação - SBIE*, Uberlândia, Brasil, páginas 52-59. doi:[10.5753/cbie.sbie.2016.52](https://doi.org/10.5753/cbie.sbie.2016.52)
- Caspersen M. E., Bennedsen J. (2007). Instructional design of a programming course. In: *International computing education research ICER*, Atlanta, GA, USA, páginas 111-122. doi:[10.1145/1288580.1288595](https://doi.org/10.1145/1288580.1288595)

- Covington R., Benegas L. (2005). A cognitive based approach for teaching programming to computer science and engineering students. In: ASEE PEER annual conference, Portland, OR, USA, páginas 10.17.1-10.17.25. doi:[10.1145/1288580.1288595](https://doi.org/10.1145/1288580.1288595)
- Dimitrov D. M., Rumrill P. D. (2003). Pretest-posttest designs and measurement of change. Work (Reading, Mass.), 20(2):159-165. [[GS Search](#)]
- Eagle M., Barnes T. (2008). Wu's Castle: Teaching arrays and loops in a game. In: 13th annual conference on Innovation and technology in computer science education - ITiCSE, Madrid, Spain, páginas 245-249. doi:[10.1145/1597849.1384337](https://doi.org/10.1145/1597849.1384337)
- Francisco R. E., Pereira Junior C. X., Ambrósio A. P. (2016). Juiz On line no ensino de Programação Introdutória - Uma Revisão Sistemática da Literatura. In: Anais do XXVII Simpósio brasileiro de informática na educação - SBIE, Uberlândia, Brasil, páginas 11-20. doi:[10.5753/cbie.sbie.2016.11](https://doi.org/10.5753/cbie.sbie.2016.11)
- Galdino C. B. T., Neto S. R. S., Costa E. B. (2015). KidCoder: Uma Proposta de Ensino de Programação de forma Lúdica. In: Anais do XXVI Simpósio brasileiro de informática na educação - SBIE, Maceió, Brasil, páginas 687-691. doi:[10.5753/cbie.sbie.2015.687](https://doi.org/10.5753/cbie.sbie.2015.687)
- Galvão L., Fernandes D., Gadelha B. (2016). Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. In: Anais do XXVII Simpósio brasileiro de informática na educação - SBIE, Uberlândia, Brasil, páginas 140-149. doi:[10.5753/cbie.sbie.2016.140](https://doi.org/10.5753/cbie.sbie.2016.140)
- Gil A. C. (2000). Como Elaborar Projetos de Pesquisa. Atlas S.A., São Paulo.
- Guimarães A., Lages N. (2012). Algoritmos e estruturas de dados. Bookman, Porto Alegre.
- Hernandez C. C., Silva L., Segura R. A., Schimiguel J., Ledon M. F., Bezerra L. N. et al. (2010). Teaching Programming Principles through a Game Engine. CLEI electronic journal, 13(2):1-8. [[GS Search](#)]
- Inception. (2010). Christopher Nolan. United Kingdom: Warner Bros. Pictures. 148 min.
- Laureano M. (2008). Estrutura de dados com algoritmos e C. Brasport, Rio de Janeiro.
- Likert R. (1932). A Technique for the Measurement of Attitudes. Archives of Psychology, 22(140):1-55. [[GS Search](#)]
- Mattos F., Ferreira V., Anacleto J. (2016). O Ensino de Programação com Scratch e seu Impacto na Opção Profissional para Meninas. In: Anais do XXVII Simpósio brasileiro de informática na educação - SBIE, Uberlândia, Brasil, páginas 300-309. doi:[10.5753/cbie.sbie.2016.300](https://doi.org/10.5753/cbie.sbie.2016.300)
- Odifreddi P. (1999). Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers. Elsevier, Amsterdam. [[GS Search](#)]
- Oliveira M., Souza M., Reblin L., Oliveira E. (2016). Reconhecimento Automático de Representações de Rúbricas em Agrupamentos de Soluções de Exercícios de Programação. In: Anais do XXVII Simpósio brasileiro de informática na educação - SBIE, Uberlândia, Brasil, páginas 1106-1015. doi:[10.5753/cbie.sbie.2016.1106](https://doi.org/10.5753/cbie.sbie.2016.1106)
- Ramos V., Freita M., Galimbert M., Mariani A. C., Wazlawick R. (2015). A Comparação da realidade mundial do ensino de programação para iniciantes com a realidade nacional: Revisão sistemática da literatura em eventos brasileiros. In: Anais do Simpósio brasileiro de informática na educação - SBIE, Maceió, Brasil, páginas 318-327. doi:[10.5753/cbie.sbie.2015.318](https://doi.org/10.5753/cbie.sbie.2015.318)

- Sharp H., Rogers Y., Preece J. (2007). Interaction design: beyond human-computer interaction. John Wiley & Sons, New York. [[GS Search](#)]
- Stewart T. C., Eliasmith C. (2011). Neural Cognitive Modelling: A Biologically Constrained Spiking Neuron Model of the Tower of Hanoi Task The Tower of Hanoi. In: 33rd Annual meeting of the cognitive science society - Cog-Sci, Boston, MA, USA, páginas 656-661. [[GS Search](#)]
- Wilcocks D., Sanders I. (1994). Animating recursion as an aid to instruction. Computers & Education, 23(3):221-226. [[GS Search](#)]
- Zanetti A. P., Borges M. A. F., Ricarte I. L. M. (2016). Pensamento Computacional no Ensino de Programação: Uma Revisão Sistemática da Literatura Brasileira. In: Anais do XXVII Simpósio brasileiro de informática na educação - SBIE, Uberlândia, Brasil, páginas 21-30. doi:[10.5753/cbie.sbie.2016.21](https://doi.org/10.5753/cbie.sbie.2016.21)
- Zavala A. B. P. (2007). O estudo de funções exponenciais e logarítmicas motivado pela geometria fractal. Monografia de Especialização para Professores de Matemática, UFPR. Obtido em <https://docs.ufpr.br/~ewkaras/especializa/zavala.pdf>