UNIFACS UNIVERSIDADE SALVADOR
MESTRADO EM SISTEMAS E COMPUTAÇÃO

ALMIR DAVID VALENTE SANTIAGO

METODOLOGIA LÚDICA APLICADA NO ENSINO DE CONCEITOS DE PROGRAMAÇÃO DE COMPUTADORES

Salvador
2016

ALMIR DAVID VALENTE SANTIAGO

METODOLOGIA LÚDICA APLICADA NO ENSINO DE CONCEITOS DE PROGRAMAÇÃO DE COMPUTADORES

Dissertação apresentada ao Programa de Pós-Graduação em Sistemas e Computação da UNIFACS Universidade Salvador, Laureate International Universities como requisito parcial para obtenção do título de Mestre.

Orientador: Prof. Dr. Artur Henrique Kronbauer.

Salvador
2016

FICHA CATALOGRÁFICA
Elaborada pelo Sistema de Bibliotecas da UNIFACS Universidade Salvador, Laureate International Universities)

Santiago, Almir David Valente
Metodologia lúdica aplicada no ensino de conceitos de programação de computadores./ Almir David Valente Santiago.- Salvador: UNIFACS, 2016.

170 f. : il.

Dissertação apresentada ao Curso de Mestrado em Sistemas e Computação, Universidade Salvador – UNIFACS, como requisito parcial para obtenção do grau de Mestre.

Orientador: Prof. Dr. Artur Henrique Kronbauer.

1. Ensino - Programação de computadores. 2. Metodologias Lúdicas. 3. Psicologia Cognitiva. I. Kronbauer, Artur Henrique, orient. II. Título.

CDD: 374.26

ALMIR DAVID VALENTE SANTIAGO

METODOLOGIA LÚDICA APLICADA NO ENSINO DE CONCEITOS DE PROGRAMAÇÃO DE COMPUTADORES

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre em Sistemas e Computação, UNIFACS Universidade Salvador, Laureate International Universities, pela seguinte banca examinadora:

Artur Henrique Kronbauer – Orientador _____
Doutor em Ciência da Computação, na área de Interação Humano-Computador pela Universidade Federal da Bahia (UFBA)
UNIFACS Universidade Salvador, Laureate International Universities

Paulo Nazareno Maia Sampaio _____
Doutor em Informática e Telecomunicações pela Université Toulouse III Paul Sabatier, UPS, França
UNIFACS Universidade Salvador, Laureate International Universities

Monica de Souza Massa _____
Doutora em Educação pela Universidade Federal da Bahia (UFBA)
Universidade do Estado da Bahia - UNEB

Salvador, 30 de maio de 2016.

Dedico este trabalho à minha tia Izabel Cristina, meu exemplo, à minha noiva Cemary Correia, minha companheira, aos meus pais e a todo o pensamento positivo existente no universo.

AGRADECIMENTOS

Agradeço a Deus por me conceder a serenidade, o equilíbrio e a inteligência suficientes para me permitir realizar este trabalho.

Agradeço à minha Tia, e segunda mãe, Izabel Cristina Guimarães, pelo exemplo, dedicação e amor dedicados a mim durante tantos anos.

À minha noiva e futura esposa, Cemary Correia de Souza, por todo amor, incentivo, e companheirismo durante esta jornada.

À minha querida avó, Pedrina Vivas dos Santos, pelo amor e dedicação durante quase todos os meus anos de vida.

Aos meus pais, por me concederem a vida e pelo incentivo, apesar das circunstâncias.

Aos meus tios e familiares, os quais sempre me incentivaram e torceram por mim.

Agradeço ao meu orientador, Prof. Dr. Artur Henrique Kronbauer, pelos ensinamentos, dedicação, boa vontade e por ter acreditado na proposta do trabalho.

À Josiane Melo, pelo conhecimento, boa vontade e grande ajuda nos procedimentos administrativos.

Aos colegas, pelas constantes trocas de conhecimentos, os quais foram fundamentais para a minha formação acadêmica.

À Universidade Salvador, por ter me concedido a oportunidade de realizar uma parte de um grande objetivo de vida.

E a todos que contribuíram para a realização deste trabalho.

“Nenhuma complexidade resiste ao tempo e à repetição”.

Autor deste trabalho.

RESUMO

Esta dissertação relata o processo de pesquisa e construção de uma metodologia lúdica para o ensino de conceitos de programação de computadores. A proposta, oriunda de uma revisão bibliográfica na área da psicologia da programação, está fortemente fundamentada em trabalhos científicos que propõem a substituição da abordagem textual por metáforas lúdicas, sob a prerrogativa de que as metodologias instrucionais baseadas puramente em código textual, utilizadas atualmente, têm sido as mesmas há mais de quatro décadas, sendo insuficientes para construir nos estudantes universitários iniciantes os corretos modelos mentais acerca dos complexos conceitos de programação de computadores. Para tanto, foi especificado um modelo lúdico para o ensino do conceito de algoritmos recursivos. O modelo foi fundamentado à luz dos resultados obtidos no processo de pesquisa bibliográfica e realizado por meio da implementação de três algoritmos recursivos. O processo de validação do modelo aconteceu por meio de uma pesquisa experimental, realizado em sala de aula com 38 alunos iniciantes da Universidade Salvador. Os métodos *pretest-posttest* e *Likert* foram utilizados com o intuito de avaliar, quantitativa e qualitativamente, a eficácia da proposta apresentada. Os resultados quantitativos mostraram que os participantes obtiveram significativa melhora de desempenho e os resultados qualitativos mostraram que houve boa aceitação da metodologia por parte dos alunos. Os resultados sugerem que a metodologia lúdica proposta por esta dissertação possui grande potencial como ferramenta instrucional a ser aplicada como metodologia de ensino para alunos iniciantes nas Universidades.

Palavras-chave: Ensino de Programação. Metodologias Lúdicas. Psicologia Cognitiva. Psicologia da Programação. Recursividade. Modelos Mentais.

ABSTRACT

This dissertation describes the process of research and construction of a ludic methodology for teaching computer programming concepts. The proposal, coming from a literature review in the area of programming psychology, is strongly based on scientific studies that propose the replacement of textual approach (sintax-driven) by ludic metaphors (schema-driven) under the prerogative of the instructional methodologies based purely on textual code, currently used, have been the same for more than four decades, being insufficient to build in the university students the correct mental models about the complex concepts of computer programming. Thus, it was specified a ludic model for teaching the concept of recursive algorithms. The model was based on the results obtained in the literature search process and carried out through the implementation of three recursive algorithms. The model validation process took place through a case study in the classroom with 38 beginner students of the University Salvador. The pretest-posttest and Likert methods were used in order to evaluate quantitatively and qualitatively, the effectiveness of the proposal. The quantitative results showed that participants had significant improvement of performance and qualitative results showed good acceptance of the methodology by the students. The results suggest that the ludic methodology proposed by this dissertation has great potential as an instructional tool to be applied as a teaching methodology for beginning students in universities.

Keywords: Programming teaching. Ludic Methodologies. Psychology of Programming. Recursion. Sintaxe-Driven. Schema-Driven. Mental Models.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

CLT	*Cognitive Load Theory*
D	Diferença de desempenho entre o pré-teste e o pós-teste
EL	*Extraneous Load*
GL	*Germane Load*
IL	*Intrinsic Load*
LTM	*Long-Term Memory*
SR	*Sensory register*
STM	*Short-Term Memory*
T	Tratamento experimental
VD	Variável Dependente
VI	Variável Independente

SUMÁRIO

1 INTRODUÇÃO

O ensino de programação de computadores nas universidades é um assunto que tem sido alvo de grandes discussões em diversos trabalhos científicos, tais como, Cooper et al. (2012), Ambrósio et al. (2011), Hernandez et al. (2010) e Eagle et al. (2008). Estes trabalhos afirmam que a programação é um assunto complexo e envolve uma grande quantidade de conhecimentos. Eles abordam as dificuldades enfrentadas nas universidades para o ensino da matéria e relatam grande preocupação com os elevados índices de desistência dos cursos de computação decorrentes destas dificuldades. Algumas iniciativas, tal como Hernandez et al. (2010), discutem os principais fatores envolvidos, outras buscam, por meio de metodologias lúdicas, propor formas criativas de mitigar essas dificuldades [Eagle et al., 2008]. Existem ainda iniciativas que buscam, em modelos e metodologias pedagógicas, formas de explicar as dificuldades sob o ponto de vista da psicologia (ALAOUTINEN ; SMOLANDER, 2010).

Independente da abordagem utilizada, todos os estudos advogam pela melhoria do ensino da programação e reconhecem que ainda existem muitas oportunidades para contribuições nesta área. Neste sentido, esta dissertação objetiva, por meio da pesquisa bibliográfica na área da psicologia da programação, formar a base de conhecimentos necessários para a definição de uma abordagem lúdica para o ensino de conceitos de programação de computadores, voltada aos alunos iniciantes dos cursos de computação e engenharias.

1.1 CONTEXTO

A proposta apresentada nesta dissertação está inserida em duas grandes áreas: (I) o **ensino/aprendizagem de programação de computadores** e (II) o **estudo da psicologia cognitiva**. A primeira trata de uma competência complexa e que envolve diversos domínios do conhecimento. A segunda trata de estudos dos processos cognitivos da mente humana.

1.2 PROBLEMA

Os estudos da bibliografia científica sobre o ensino/aprendizagem de programação de computadores apresentaram evidências de que: (i) uma metodologia de ensino de programação de computadores baseada puramente em sintaxe é perniciosa para o processo de aprendizado dos alunos universitários (COVINGTON; BENEGAS, 2005); (ii) os alunos são desestimulados devido às dificuldades com as matérias de programação, pois não conseguem criar os corretos modelos mentais acerca do conhecimento que está sendo aprendido (RENUMOL et al., 2010); (iii) há necessidade de novas propostas de metodologias instrucionais, as quais não sejam baseadas puramente em código textual (EAGLE et al., 2008).

Ambrósio et al. (2011) afirmam que as matérias introdutórias de programação são responsáveis pelas altas taxas de desistência dos alunos, e Hernandez et al. (2010) ratificam esta ideia, afirmando que as desistências ocorrem já nos primeiros semestres dos cursos. Eagle et al. (2008) afirmam que a quantidade de alunos que desistem dos cursos de computação após as primeiras matérias de programação chega a 40%. Caspersen and Bennedsen (2007) afirmam que os 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.

Quanto às principais causas para o problema, Renumol et al. (2010) afirmam que a grande quantidade de processos cognitivos envolvidos na tarefa da programação a torna complexa. Alaoutinen e Smolander (2010) relatam os altos níveis de dificuldade impostos pelos conceitos da programação introdutória. Cooper et al. (2012) afirmam que a abordagem do ensino de programação tem sido a mesma nos últimos 40 anos.

Com relação às possíveis soluções, Covington and Benegas (2005) propõem uma mudança da abordagem do ensino de programação de computadores baseada em sintaxe (syntax-driven) para uma abordagem baseada no desenvolvimento de modelos mentais sobre padrões de programação (schema-driven) e como eles podem ser utilizados para resolver problemas do mundo real.

No que diz respeito aos motivos para a escolha da psicologia cognitiva como área de pesquisa, estudos como Khairuddin e Hashim (2008), Jesus e Raabe (2009), Sorva (2012) e Alaoutinen e Smolander (2010) demonstraram relação com o tema, por meio da abordagem da Taxonomia de Bloom no ensino de programação. Outros estudos mais antigos, como Hannafin et al. (1988), Hannafin e Rieber (1989) e Abd-El-Hafiz e Basili (1996) apresentaram a associação da psicologia cognitiva com os processos de ensino/aprendizagem.

A partir da identificação da problemática em questão, da percepção de que as metodologias instrucionais estão relacionadas com a psicologia cognitiva, e da percepção de que o aprendizado da programação é importante para os processos da civilização moderna (STROUSTRUP, 2012), foi possível formular o problema a ser investigado por esta dissertação: **Como mitigar os elevados índices de desistência dos alunos iniciantes em computação, devido às dificuldades enfrentadas por eles com as matérias introdutórias de programação de computadores?**

1.4 OBJETIVO

O objetivo geral deste trabalho é **propor uma metodologia lúdica, baseada na psicologia cognitiva, para facilitar o aprendizado do conceito de programação por parte dos alunos iniciantes dos cursos de computação e engenharias**. Para alcançar este objetivo foram definidos os seguintes objetivos específicos: (i) realizar um mapeamento sistemático para identificar os mais recentes trabalhos que relatam uso da psicologia cognitiva no ensino/aprendizado de programação de computadores, com o intuito de entender o estado da arte do tema “psicologia da programação”; (ii) usar os resultados do mapeamento sistemático como fundamentação científica para a definição do modelo lúdico; (iii) implementar três algoritmos recursivos para validar o modelo proposto, e; (iv) realizar uma pesquisa experimental com alunos iniciantes na área de programação, com o intuito de validar o modelo lúdico.

1.5 JUSTIFICATIVA

Como já foi discutido, as dificuldades identificadas no estudo da programação e a percepção da sua associação com a psicologia, (KHAIRUDDIN ; HASHIM, 2008), (JESUS ; RAABE, 2009), (SORVA, 2012) (ALAOUTINEN ; SMOLANDER, 2010), foram os principais fatores que motivaram a proposta deste trabalho.

Na contramão da problemática discutida, Stroustrup (2012) demonstra em sua obra a importância do ensino da programação de computadores para o desenvolvimento da sociedade moderna, apontando diversos setores da indústria e do comércio que dependem do constante desenvolvimento de softwares para o seu funcionamento. Entende-se, portanto, que o estudo desse tema, no âmbito de prover melhores metodologias de ensino, é uma importante iniciativa.

A revisão bibliográfica sobre a psicologia cognitiva deixou clara a importância do estudo da mente humana para o maior entendimento dos processos do aprendizado. Neste contexto, destacam-se alguns trabalhos, tais como Miller (1956), que forneceu evidências científicas acerca da limitação da memória humana, e Atkinson e Shiffrin (1968), que descrevem a mente como um mecanismo de processamento de informações. Outros trabalhos importantes investigados foram Bennet e Murdock (1962) e Glanzer e Cunitz (1966), que relatam experimentos científicos de memorização cujos resultados ratificam as evidências científicas sobre a limitação da mente apresentadas por Miller.

Neste sentido, o contato com tais trabalhos deixou clara a importância do estudo da psicologia cognitiva e como ela pode fornecer melhores explicações sobre as questões relacionadas aos fatores cognitivos durante o processo de aprendizado de programação.

1.6 ESTRUTURA DA DISSERTAÇÃO

Com o intuito de descrever as etapas que compreendem o desenvolvimento dessa pesquisa, definimos a estrutura do trabalho da seguinte forma:

a) Capítulo 01 – Contextualiza e descreve os motivos para a realização deste trabalho.
b) Capítulo 02 – Discute aspectos dos processos de ensino/aprendizagem da programação de computadores.
c) Capítulo 03 – Discute a definição da psicologia cognitiva e a sua importância na busca do entendimento da mente humana.
d) Capítulo 04 – Apresenta e especifica o processo de desenvolvimento do modelo lúdico proposto para o ensino de conceitos de programação.
e) Capítulo 05 – Discute os aspectos metodológicos e a pesquisa experimental realizada.
f) Capítulo 06 – Discorre sobre os resultados da pesquisa experimental.
g) Capítulo 07 – Apresenta as conclusões e relata as possibilidades de trabalhos futuros.

2 O ENSINO DE PROGRAMAÇÃO DE COMPUTADORES

O objetivo desta etapa é contextualizar o papel da programação de computadores e a importância do seu aprendizado. Para isso, será discutido o objetivo das linguagens de programação e algumas ferramentas lúdicas que auxiliam o ensino de conceitos computacionais, bem como as principais dificuldades enfrentadas em seu processo de ensino/aprendizagem.

2.1 INTRODUÇÃO

De acordo com o arquiteto e desenvolvedor da linguagem de programação ***C++*** Bjarne Stroustrup o funcionamento de diversos serviços e produtos da vida humana dependem de software, os quais englobam um conjunto inimaginável de habilidades (STROUSTRUP, 2012). Neste contexto, a programação torna-se tarefa fundamental. Programar é imprescindível para o processo de desenvolvimento de software, os quais são as partes fundamentais para o funcionamento dos computadores que controlam quase todos os setores da sociedade.

> Nossa civilização depende de software. Melhorar o software e encontrar novos usos para o software são duas das maneiras com as quais os indivíduos podem ajudar a melhorar a vida de muitos. A programação tem um papel fundamental nisso. (STROUSTRUP, 2012, p.19).

O autor ainda afirma que a programação sozinha não tem utilidade. Que a programação não deve ser tarefa solitária e isolada. Ela é parte de muitas outras disciplinas técnicas importantes. Ela é uma ferramenta com a qual é possível expressar soluções para problemas práticos, de forma que eles possam ser testados e melhorados. Programar é a arte de modelar computacionalmente problemas da realidade, utilizando conceitos de diversos outros domínios do conhecimento.

2.2 AS LINGUAGENS COMO DESCRITORAS DE CONHECIMENTOS

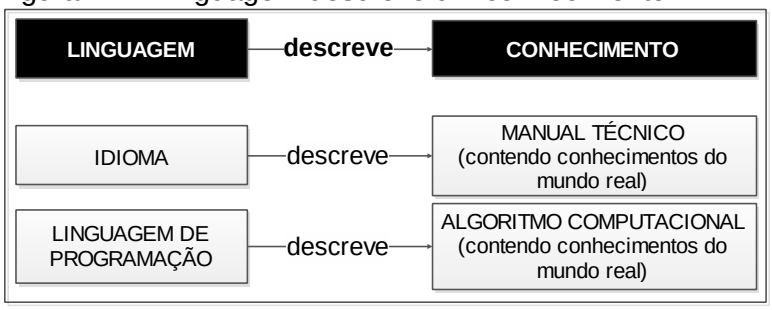
As afirmações de Stroustrup (2012) relatam um dos grandes desafios identificados no ensino da programação de computadores: fazer com que o aluno consiga modelar computacionalmente problemas do mundo real. Neste contexto se inserem

as linguagens de programação. Elas permitem descrever, em linguagem computacional, conjuntos de conhecimentos humanos, conhecimentos estes expressados em forma de algoritmos.

> Vejamos um exemplo: da mesma forma que um manual técnico de instrução veículo faz uso das palavras e das regras de um determinado idioma para descrever e explicar um conjunto de conhecimentos e procedimentos específicos de engenharia, as linguagens de programação utilizam um alfabeto próprio, com sua própria sintaxe e regras, para modelar e descrever computacionalmente conjuntos de de conhecimentos específicos do mundo real. Esta ideia é ilustrada na

Figura .

Figura 1 – A linguagem descreve um conhecimento



Fonte: Autor desta dissertação (2016).

Vejamos um exemplo de como a linguagem descreve um conhecimento através da descrição algorítmica do conceito matemático denominado *Teorema de Pitágoras* (comumente utilizado no desenvolvimento de jogos digitais para calcular distâncias entre objetos na tela do computador). Este conceito pode ser modelado por meio do algoritmo na linguagem ***C++*** descrito na Figura .

Figura 2 – Descrição do teorema de Pitágoras

```
float cateto_x, cateto_y, hypotenusa;
cateto_x = triangulo.get_cateto_x;
cateto_y = triangulo.get_cateto_y;
hypotenuse = std::sqrt(std::pow(cateto_x, 2) + std::pow(cateto_y, 2) );
```

(a)

$$hipotenusa = \sqrt{(catetoX^{2} + catetoY^{2})}$$

(b)

Fonte: Autor desta dissertação (2016).

A princípio, para um leigo em programação, o fragmento de código da Figura a pode parecer complicado, mas é a forma com a qual o programador descreve computacionalmente um conceito simples, o qual pode ser demonstrado em uma linguagem mais convencional e mais simples de ser entendida (Figura b). Os conceitos apresentados em ambas expressões da imagem representam o mesmo conhecimento do mundo real, porém expressados em linguagens distintas. Portanto, entende-se que a linguagem representa apenas a maneira como o conhecimento é apresentado.

O exemplo apresentado na Figura teve o objetivo de mostrar que a linguagem de programação, independente das complexidades envolvidas, apenas descreve um determinado conhecimento do mundo real, o teorema de Pitágoras. Isso significa que o entendimento da complexidade sintática da linguagem é apenas o passo inicial para se aprender a programar. É preciso aprender a usar o conhecimento em programação para descrever outros conhecimentos do mundo real. Stroustrup deixa bem clara essa ideia quando afirma:

> [...] a programação é mais bem vista como uma parte de algo maior. [...] Vemos a programação como uma tecnologia que capacita para as áreas de Computação e Informação de Ciência e Engenharia, bem como para Física, Biologia, Medicina, História, Literatura e outros campos acadêmicos e de pesquisa. (STROUSTRUP, 2012, p.24-25).

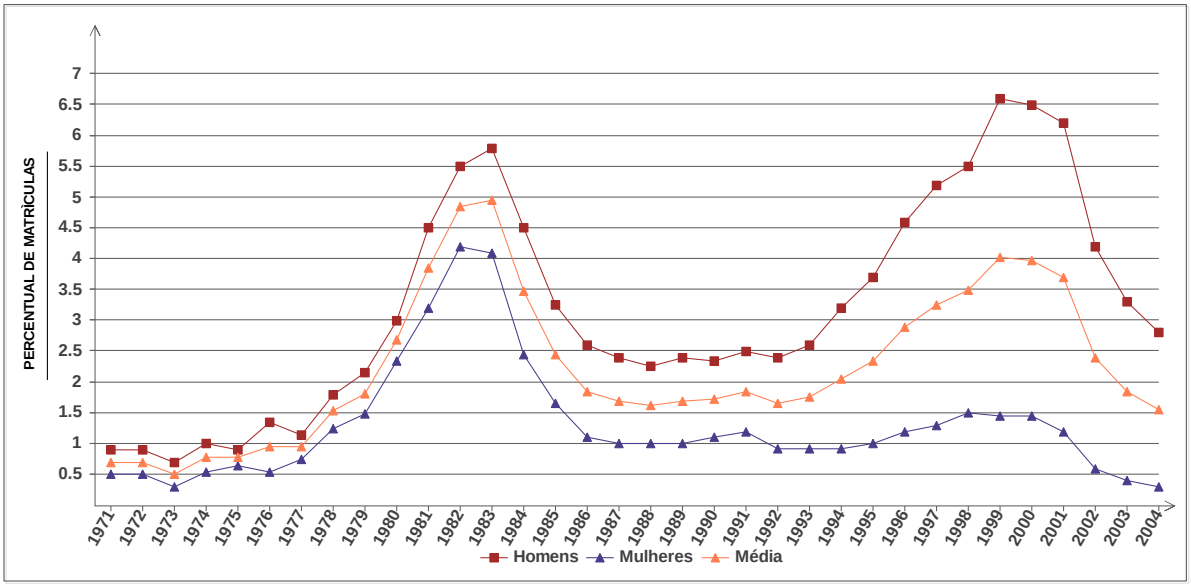
Portanto, entende-se que a programação é apenas uma ferramenta que pode ser utilizada em diversos domínios da ciência, para resolver os mais variados tipos de problemas.

2.3 OS ALTOS ÍNDICES DE DESISTÊNCIA NAS UNIVERSIDADES

Hernandez et al. (2010) explicam que as expectativas dos jovens iniciantes para com os cursos de computação está fora da realidade, pois esses cursos requerem, por exemplo, sólidas bases de conhecimentos em matemática. Os autores afirmam que as matérias de programação são planejadas para que os discentes desenvolvam habilidades cognitivas que não foram desenvolvidas pelas grades curriculares dos cursos de ensino fundamental e médio.

Os autores ainda apresentam um gráfico evolutivo contendo os números de matrículas dos cursos de computação na Universidade da Califórnia de Los Angeles (UCLA), ilustrada na Figura . Nela é possível perceber duas crescentes fases de aumento da procura por esses cursos. A primeira fase é registrada entre 1982 e 1983, e a segunda no início dos anos 2000. Os autores afirmam que a primeira fora motivada pelo advento dos microcomputadores, a partir do início dos anos 1980, e a segunda pelo surgimento da internet. Entretanto, independente das ondas motivadoras, é possível perceber uma forte tendência de queda que se mantém nos demais períodos.

Figura 3 – Número de inscrições em cursos de computação da UCLA



Fonte: Hernandez et al. (2010).

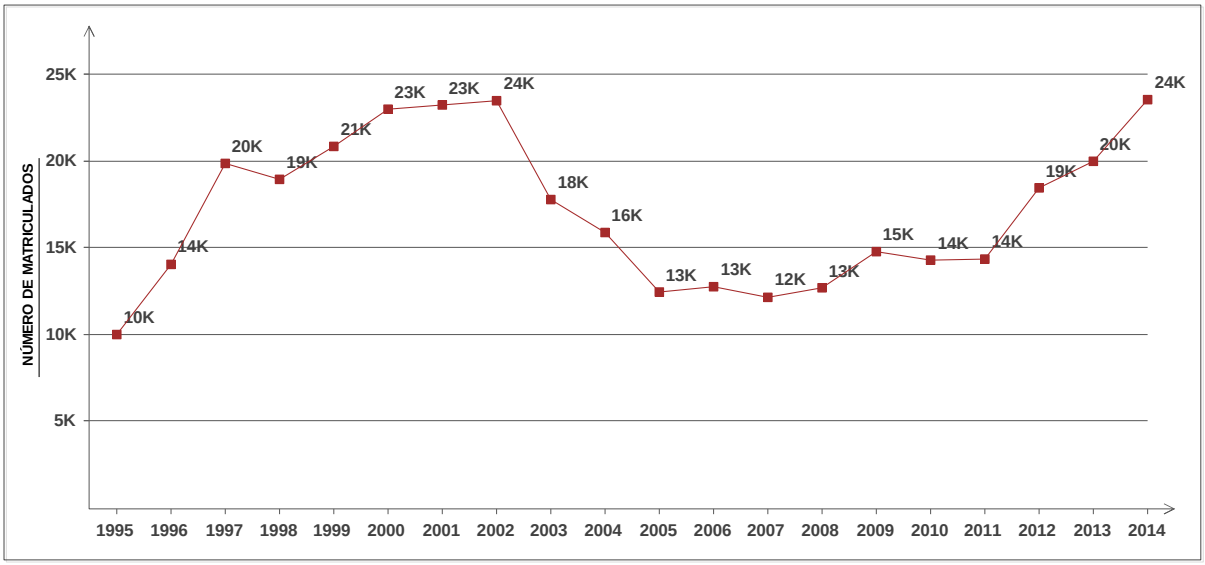
Hernandez et al. (2010) atribuem essas tendências de queda às dificuldades para o aprendizado de programação e ao fato de que tal aprendizado é de fundamental importância para o currículo de um curso de computação, além de ser pré-requisito para outras disciplinas. Eles ressaltam que a falta de conhecimentos sólidos em áreas como a matemática e o alto grau de esforço para desenvolver o raciocínio lógico tornam as expectativas do aluno frustradas, já que lhes são requisitadas habilidades cognitivas que não lhes foram desenvolvidas em suas formações de nível fundamental e médio.

Alaoutinen e Smolander (2010) ratificam este fato, afirmando que, juntamente com matérias como matemática, a programação é a maior causadora de problemas para estudantes de graduação dos cursos de computação. Os resultados dos testes realizados pelos pesquisadores indicam **ponteiros** e **gerenciamento dinâmico de memória** como os assuntos mais difíceis, segundo os próprios alunos.

Eagle et al. (2008) afirmam que a quantidade de alunos que desistem dos cursos de computação após as primeiras matérias de programação chega a 40%, e que desde o ano 2000, o número de matrículas desses cursos tem diminuído. Esta afirmação também pode ser confirmada pelo gráfico apresentado por Hernandez et al (2010) (Figura).

Outro estudo relevante e que apresenta dados sobre a procura por cursos de computação é o relatório Taulbee Survey (Computing Resource Association, 2015), uma das principais fontes de informações sobre dados estatísticos das áreas de Ciências e Engenharia da Computação na América do Norte. A Figura mostra o gráfico evolucionário do número de novos alunos graduandos das universidades norte americanas e confirma a veracidade do aumento de matrículas citado por Hernandez et al. (2010) no início dos anos 2000, confirmando a forte tendência por estes cursos na referida época.

Figura 4 – Número de novos estudantes de graduação em cursos de computação



Fonte: Taulbe e Survey (2015).

Outro dado que chama atenção no gráfico da Figura refere-se à tendência de crescimento que vem se apresentado desde 2008. Pode-se levantar a hipótese de que teria alguma relação com o advento e popularização dos dispositivos móveis? Contudo, é importante perceber que o gráfico sugere que os cursos de computação estão enfrentando uma nova fase de grande procura, o que abre espaço para questionamentos sobre a possibilidade de esta apenas ser uma terceira onda, similar às registradas nas décadas de 1980 e 2000. Será que o assunto “programação de computadores” representará novamente o grande pivô de uma nova tendência de queda pela procura desses cursos? É muito difícil de responder tal questão, porém, fica clara a grande responsabilidade com o tema, bem como a grande validade da pesquisa científica que busque melhores metodologias instrucionais para o ensino de programação, principalmente em um século onde, segundo Stroustrup (2012), todos os serviços da vida humana dependem de software.

2.4 METODOLOGIAS LÚDICAS NO ENSINO DE PROGRAMAÇÃO

Uma boa iniciativa identificada na bibliografia consultada para a problemática em questão são as metodologias lúdicas. Elas são defendidas como formas alternativas e mais eficientes de motivar os alunos a desenvolverem o pensamento computacional. Neste sentido, diversos autores as propõem para o ensino de conceitos de programação, tais como, *Carnegie Mellon University* (2016), Hernandez et al. (2010) e Eagle et al. (2008).

O projeto Alice é uma iniciativa da *Carnegie Mellon University* (2016) que propôs a criação de um software educacional voltado ao ensino de programação de computadores para iniciantes (Figura). O software fornece um ambiente interativo que auxilia os iniciantes, por meio de ferramentas e materiais, no processo de prototipação de ambientes tridimensionais, estimulando o desenvolvimento do pensamento computacional.

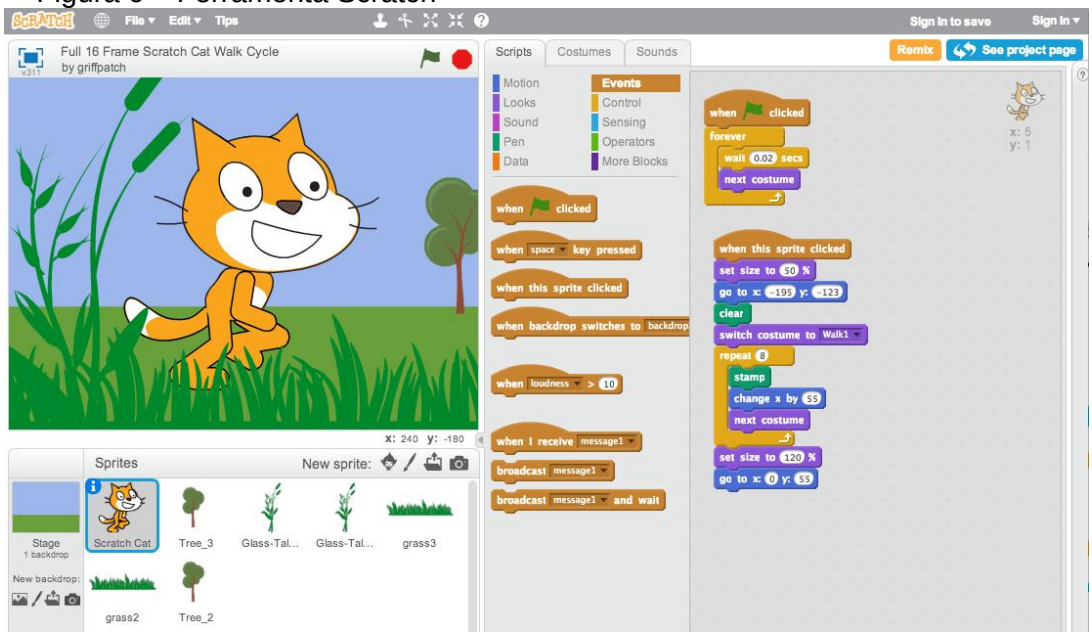
Figura 5 – Ferramenta Alice 3

Fonte: Carnegie Mellon University (2016).

Por meio de comandos de "arrastar e soltar" elementos gráficos, os alunos podem criar programas. Os comandos fornecidos pela ferramenta correspondem aos padrões de programação das linguagens orientadas a objetos, tais como, *C++*, *Java* e *C#*. A plataforma objetiva que, por meio da observação de animações, os alunos compreendam a relação existente entre os constructos de programação e o comportamento dos objetos gráficos. Atualmente a ferramenta encontra-se na versão 3 e possui grande quantidade de material de referência voltados, tanto para professores, como para alunos. A iniciativa já originou diversas publicações em eventos e revistas científicas, além de uma tese de doutorado.

O Scratch é uma linguagem de programação baseada em componentes gráficos voltada a jovens entre 8 e 16 anos de idade (Figura). Desenvolvida por *Lifelong Kindergarten* (2016), grupo pertencente ao MIT Media Lab, a linguagem é integrada e distribuída em uma plataforma online, a qual conta com uma grande comunidade que cria, ensina e distribui projetos construídos na plataforma.

Figura 6 – Ferramenta Scratch

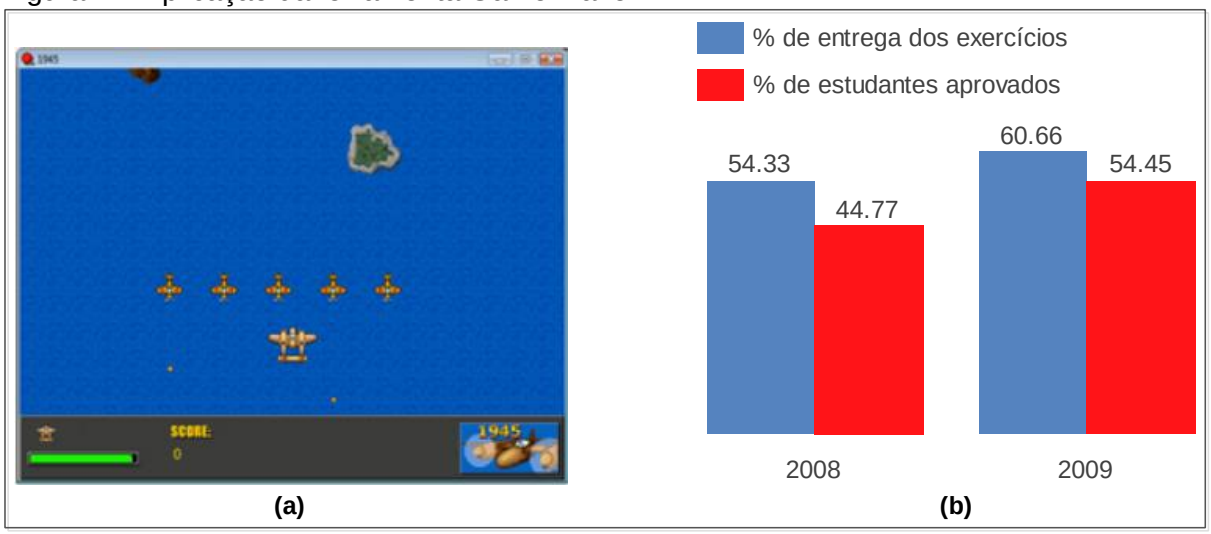


Fonte: Carnegie Mellon University (2016).

O Scratch permite que crianças aprendam de maneira intuitiva, criando histórias, animações e jogos, por meio de mecanismos de “arrastar e soltar” os componentes gráficos. Desde o ano de 2007, a linguagem vem sendo adotada por educadores, no âmbito de fornecer melhores formas de ensino programação e provês o pensamento computacional.

Hernandez et al. (2010) propuseram ensinar conceitos fundamentais de programação por meio de um curso introdutório da ferramenta *Game Maker*. O objetivo foi utilizar a construção de jogos para introduzir aos alunos conceitos básicos de programação. A estratégia utilizada pelos autores foi a de ensinar a manipular estruturas lógicas de forma visual, sem a necessidade de utilizar linguagens e, aos poucos, fazer a transição para a programação textual. O cenário se baseia em um jogo com aeronaves, onde é possível perceber de forma visual as mudanças ocorridas na interface gráfica de acordo com a manipulação visual dos valores das variáveis. A Figura a ilustra a interface da aplicação utilizada pelos autores.

Figura 7 – Aplicação da ferramenta Game Maker



Fonte: Hernandez et al. (2010).

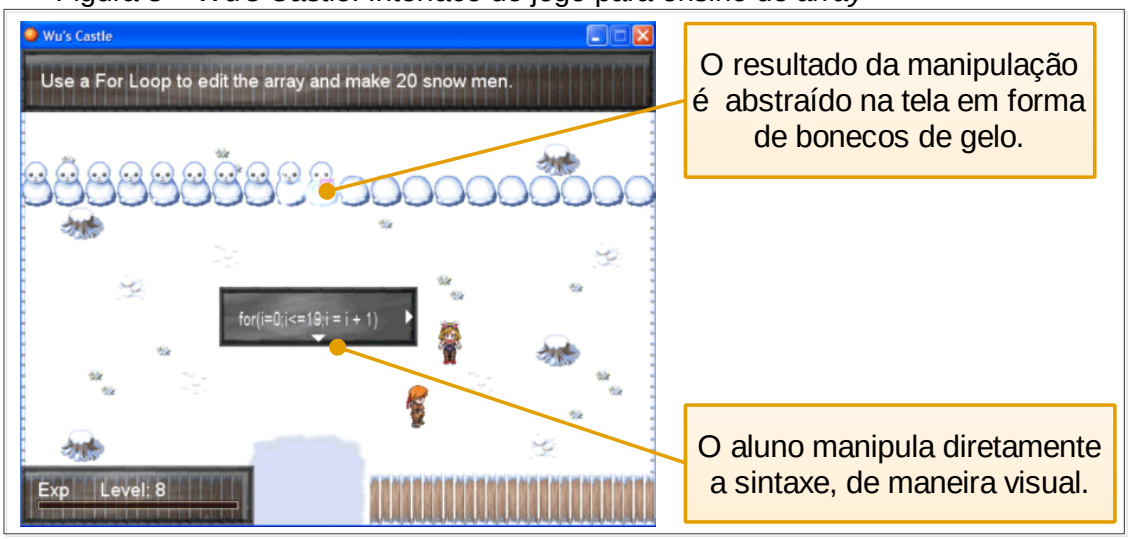
Em suas conclusões (Figura b) os autores sugerem uma melhora nos índices de aprovação dos estudantes entre os anos de 2008 e 2009, período entre o qual foi introduzida a metodologia. Eles relatam também que o experimento permitiu que os alunos pudessem começar a construir seus conhecimentos acerca de conceitos de programação sem a necessidade de formalizações ou especificações de linguagens, e que o uso de elementos lúdicos, como a criação de jogos, abre possibilidades interessantes para o ensino.

Eagle et al. (2008) propuseram um jogo denominado *Wu's Castle,* voltado para o ensino de conceitos computacionais de ***"loops"*** e ***"arrays"*** em ***C++***. Os autores afirmam que estes dois conceitos possuem elevado índice de dificuldade. No jogo, é proposta a junção do código textual com a abstração visual, onde o aluno tem a possibilidade de realizar a construção interativa do algoritmo por meio da manipulação de um código de programação desenhado na tela.

Para o ensino do conceito de ***"arrays"***, foi proposta uma interface (Figura) onde o aluno deve manipular valores de uma estrutura de repetição ***"for"*** visualmente, de maneira que o resultado desta manipulação é representado como um conjunto de bonecos de gelo. O jogador manipula esses parâmetros com o intuito de definir as posições a serem visitadas, sendo possível visualizar o resultado na tela. Segundo o autor, esta implementação permite que o aprendiz consiga visualizar em tempo real

o funcionamento de um ***"array"*** por meio das alterações nos parâmetros textuais do código, proporcionando um entendimento mais concreto desta abstração.

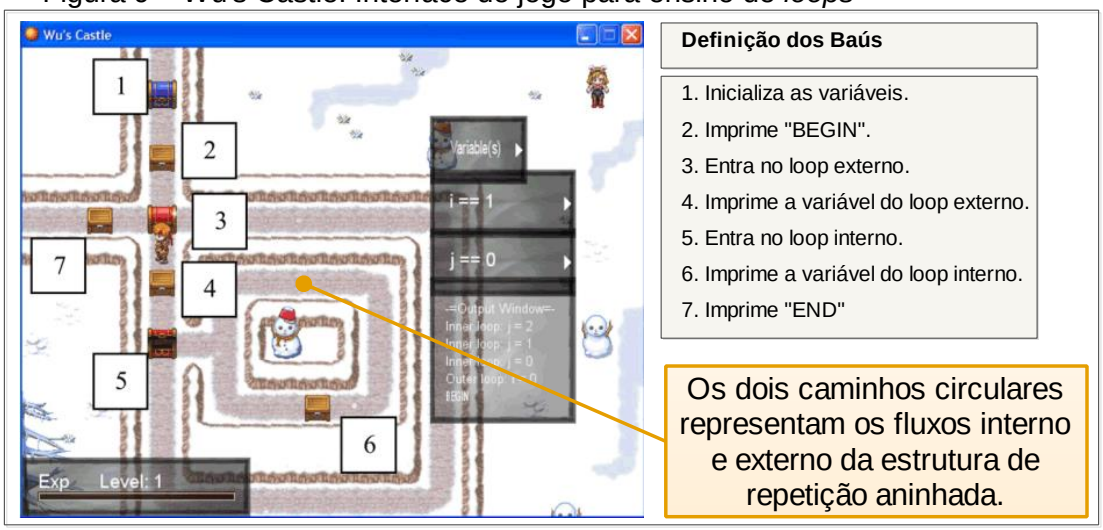
Figura 8 – Wu's Castle: Interface do jogo para ensino de *array*



Fonte: Eagle et al. (2008).

Como forma de introduzir o conceito de estrutura de repetição aninhada (***loop***) o autor utilizou a abstração ilustrada na Figura , que representa a execução de uma estrutura ***"for"*** aninhada. Esta abstração contém dois caminhos circulares que representam os dois níveis de repetição da estrutura. Durante o fluxo de execução, o jogador deve movimentar o personagem em direção a baús, os quais representam as etapas algorítmicas que o direciona para os caminhos circulares.

Figura 9 – Wu's Castle: Interface do jogo para ensino de *loops*

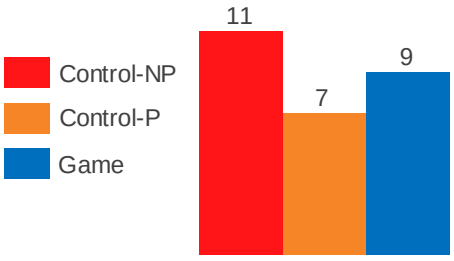


Fonte: Eagle et al. (2008).

Segundo os autores, essa metáfora foi desenvolvida para ajudar os alunos a visualizarem o fluxo de execução de uma estrutura de repetição aninhada por meio da observação da trajetória de um personagem, sob seu controle, executando o teste de mesa. O intuito foi o de fornecer ao aluno um melhor entendimento da relação existente entre código e imagem. Os valores das variáreis de controle das repetições foram exibidos constantemente na tela, para que eles acompanhassem estas mudanças durante a execução do jogo.

Os autores investigaram se o uso de jogos voltados para o ensino de conceitos de programação se apresentava como uma iniciativa motivadora para que os alunos aprendessem fora da sala de aula. Para tanto, foi realizada uma pesquisa experimental como forma de validação da ferramenta proposta. A pesquisa fez uso da metodologia *pretest-posttest*, na qual foram definidos dois grupos de controle (*Control-NP* e *Control-P*) e um grupo experimental (*Game*), ilustrados na Figura a.

Figura 10 – Wu's Castle: Resultados do experimento



(a)

	Q1	Q2	Q3	Q4	Q5	Tot.	%
cP	0.72	0.33	0.17	0.17	0.21	1.60	32
cNP	0.47	0.36	0.18	0.09	0.27	1.38	27.6
cP+cNP	0.59	0.35	0.17	0.13	0.24	1.49	29.8
Game	0.81	0.85	0.25	0.40	0.48	2.79	55.8

(b)

Fonte: Eagle et al. (2008).

A Figura b mostra os resultados do experimento para os três grupos, onde é possível observar claramente a pouca diferença entre os resultados dos grupos de controle (**32% e 27,9%**). Porém, quando se observa os resultados do grupo experimental, que foi submetido à metodologia proposta pelos autores, fica claro o ganho significativo de desempenho (**55,8%**), sugerindo a validade da metodologia.

Eagle et al. (2008) afirmam que os resultados obtidos foram muito satisfatórios. Os estudantes que jogaram *Wu's Castle* alcançaram ganhos significativos de aprendizado quando comparados ao grupo de controle. Os resultados sugerem a eficácia no uso da metodologia lúdica baseada em jogos para o ensino de programação. Eles concluem o trabalho afirmando que o jogo implementa boas

práticas diferenciadas, tais como, frequentes respostas imediatas e visualização interativa do código, e que o trabalho deles deve encorajar os docentes a considerarem seu uso em sala de aula. Eles afirmam também que futuros jogos poderiam adotar tal metodologia para desenvolver outros conceitos de programação.

2.5 CONCLUSÕES

Stroustrup (2012) mostrou a importância do aprendizado da programação, afirmando que ela é a atividade fundamental para o processo de desenvolvimento de software, que por sua vez é a peça fundamental para o funcionamento de diversos setores da vida humana.

Ficou claro que a programação não está apenas relacionada com a computação, pelo fato dela ser apenas uma ferramenta para descrever conhecimentos do mundo real e solucionar os mais diversos tipos de problemas, de diversos domínios do conhecimento. Desta forma, conclui-se que aprender a programar não é apenas adquirir a habilidade de escrever algoritmos. O verdadeiro desafio é adquirir a habilidade de convergir este conhecimento com outros, os quais podem ser modelados computacionalmente por meio da programação, executados pelos softwares e tornados úteis à vida humana.

Foram apresentados dados estatísticos que indicam as dificuldades enfrentadas pelos alunos iniciantes nos cursos de computação e ficou claro que a multidisciplinaridade de conhecimentos necessários ao aprendizado da programação é o principal fator que desmotiva os alunos. Portanto, conclui-se que as dificuldades enfrentadas pelos alunos se deve ao fato da programação de computadores ser um domínio do conhecimento que está fortemente associado com outros conhecimentos, sendo apresentada de maneira muito distante do contexto de formação educacional dos estudantes universitários iniciantes.

A análise das propostas e os resultados dos trabalhos apresentados neste capítulo mostraram como as metodologias lúdicas podem ser relevantes para o ensino. Foram apresentadas duas metodologias lúdicas para o ensino de conceitos de

programação, cujos resultados as sugerem como sendo boas alternativas para mitigar as dificuldades com o aprendizado inicial. Foi possível observar que qualquer conceito que requeira grande capacidade de abstração pode sim ser facilitado por meio dessas abordagens, as quais permitem explorar formas inteligentes de explicar conceitos abstratos.

Desta forma, conclui-se que o uso de metodologias lúdicas é uma excelente forma de facilitar a aquisição do conhecimento. Não é possível afirmar, a partir desta análise, que o uso de jogos seja a solução para os problemas instrucionais com o ensino de programação, mas pode-se afirmar que ela é o veículo que, aliando a metodologias bem fundamentadas, pode proporcionar melhorias significativas no ensino de programação.

3 O ESTUDO DA PSICOLOGIA COGNITIVA

Este capítulo traz uma discussão sobre a psicologia cognitiva. O que é, como surgiu e qual a sua importância. A existência deste capítulo se deve: (i) à necessidade de apresentar a teoria de Miller (1956), que relata a limitação da mente humana em armazenar grandes quantidades de informações; e (ii) à necessidade de formar a base de conhecimentos necessários para o entendimento dos assuntos que serão tratados nos capítulos posteriores e nas fundamentações do modelo proposto por esta dissertação.

Os termos e as teorias que justificam a proposta deste trabalho são melhor explicados neste capítulo. Termos como "*estímulos sensoriais"* e "*memória de curto prazo"* são aqui abordados em concordância com a bibliografia científica. Portanto, a existência deste capítulo se fundamenta na necessidade de introduzir alguns conceitos, para que o leitor possa melhor compreender a fundamentação do modelo proposto por esta dissertação.

3.1 DEFINIÇÃO DA PSICOLOGIA COGNITIVA

A psicologia cognitiva estuda os processos mentais que interferem no comportamento do ser humano e de suas ações. Na década de 1950, ela representou uma mudança de paradigma para a investigação do comportamento humano, que até então não considerava os aspectos internos da mente.

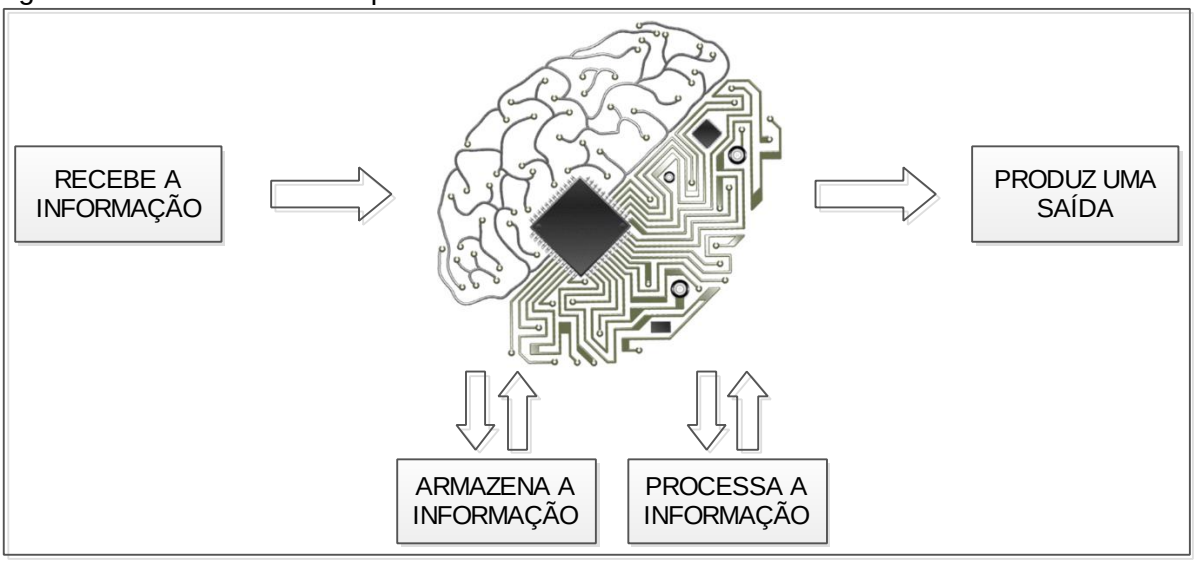
A origem do seu nome como área de pesquisa veio da publicação do livro de Neisser (1967), denominado *Cognitive psychology*, no qual o autor reuniu a investigação sobre a percepção, o reconhecimento de padrões, a atenção, a capacidade de resolução de problemas e a memória. Collin (2012) confirma este fato, dizendo que, apesar da sua origem estar fundamentada em trabalhos de diversos autores, tais como *George Armitage Miller* e *Jerome Bruner*, o termo *Psicologia Cognitiva* foi cunhado através da publicação do livro de Neisser.
Segundo Frensch (2001) a psicologia cognitiva é um vasto campo da ciência que abriga diversos sub tópicos, mas que de modo geral se concentra em explicar a

estrutura e as operações mentais da inteligência e suas manifestações comportamentais.

McLeod (2007a) afirma que a psicologia cognitiva se refere ao estudo dos processos mentais e o seu papel no pensamento, sentidos e comportamento humano. Ele relata que esta ciência está interessada nas variáveis que intermediam as entradas (estímulos sensoriais) e as saídas (respostas). Além disso, o autor relata que para entender como as pessoas agem é preciso compreender os processos internos de suas mentes. Ele afirma que a abordagem cognitiva revolucionou a psicologia entre as décadas de 1950 e 1960, sendo os principais fatores que contribuíram para isso:

a) A insatisfação com a abordagem “behaviorista” (WATSON, 1913), que considerava apenas os aspectos externos do ser.
b) O advento da metáfora do computador, que propunha um modelo para análise dos processos humanos internos.
c) O desenvolvimento de melhores métodos experimentais.

Figura 11 – Metáfora do computador



Fonte: Adaptado de McLeod (2007a).

A mudança de paradigma proposta pela psicologia cognitiva está fortemente relacionada com o advento das tecnologias da informação e da inteligência artificial, áreas estas que forneceram um modelo comparativo para que se pudesse melhor entender o funcionamento do cérebro. Neste contexto, se insere a Metáfora do

Computador (Figura), que foi utilizada pela psicologia cognitiva como modelo para a investigação da mente.

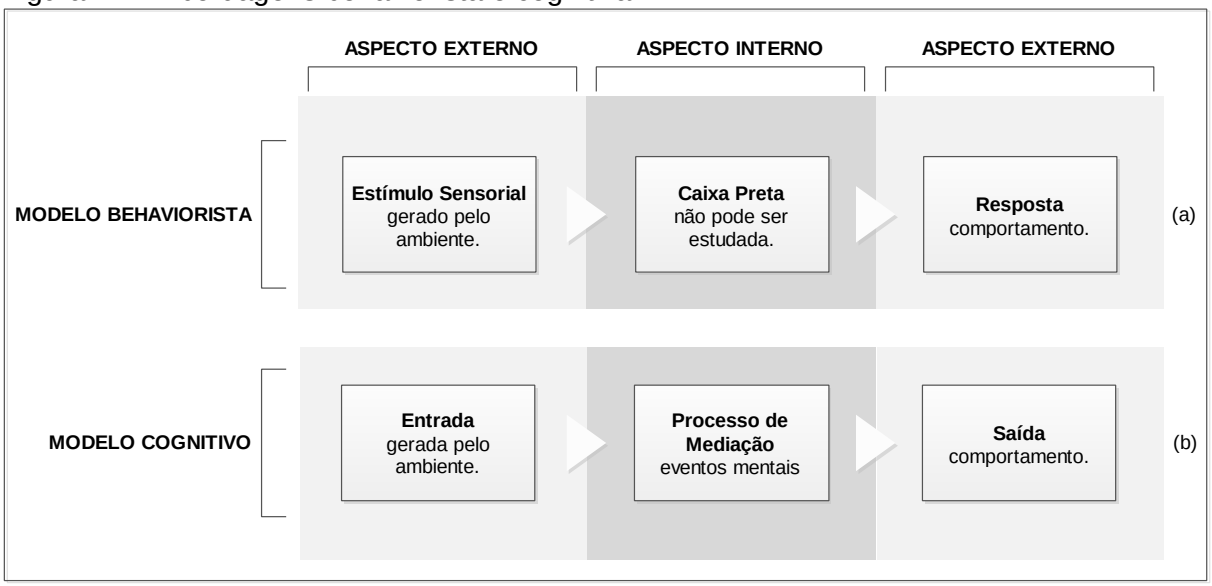
Frensch (2001) afirma que o sistema cognitivo humano pode ser melhor entendido quando se faz uma analogia à um sistema de processamento de informação. Segundo o autor, Neisser (1967) usou a metáfora do computador para descrever o processo cognitivo, sendo esta analogia ainda a mais aceita para fundamentar as pesquisas na área.

Segundo McLeod (2007a) o advento do computador deu aos cientistas a terminologia e a metáfora de que precisavam para tentar entender as complexidades desses processos intermediários da cognição humana, fato este que era ignorado pelos “behavioristas”.

Collin (2012) ratifica esta ideia, afirmando que na primeira metade do século XX a psicologia foi dominada por duas correntes de pensamento: o *Behaviorismo* e a *Psicanálise*. Correntes estas que desconsideravam os processos internos da mente, tais como, percepção, consciência e memória. Além disso, o autor salienta que o desenvolvimento da tecnologia dos computadores e o advento da inteligência artificial foi o que motivou a denominada *Revolução Cognitiva*.

A Figura ilustra a diferença entre as abordagens behaviorista e a cognitiva. O modelo behaviorista estuda apenas os estímulos e suas respectivas repostas, pois considera que o comportamento interno não pode ser estudado, devido ao fato de que não é possível saber o que ocorre dentro da mente humana (Figura a). Em contrapartida, o modelo cognitivo acredita que os aspectos internos da mente, pode ser estudado por meio de experimentos científicos (Figura b).

Figura 12 – Abordagens behaviorista e cognitiva



Fonte: Adaptado de McLeod (2007a).

Diante das abordagens descritas, entende-se que a psicologia cognitiva representou uma mudança de paradigma na investigação científica do pensamento, que passava a considerar não somente a exteriorização da expressão humana, mas também buscava entender a estrutura e os mecanismos internos da mente. Sendo assim, a metáfora do computador permitiu aos pesquisadores terem outro ponto de vista para a observação dos experimentos científicos.

Quando o cérebro humano passou a ser pensado como uma arquitetura análoga a um sistema computacional, os pesquisadores puderam estabelecer relações de causa e efeito entre o comportamento interno e externo do ser, e criar teorias acerca do funcionamento e limitações da memória.

3.2 POR QUE O ESTUDO DA PSICOLOGIA COGNITIVA?

A associação do ensino de programação aos assuntos da psicologia é um forte aspecto desta dissertação. Muitos dos trabalhos sobre programação, investigados durante o levantamento bibliográfico preliminar, demonstraram estar relacionados às metodologias pedagógicas oriundas da psicologia cognitiva. O exemplo mais evidente deste fato são as abordagens que relatam a Taxonomia de Bloom, a qual define níveis cognitivos hierarquizados para o aprendizado.

Jesus e Raabe (2009) discutem a interpretação desta taxonomia e seu uso no contexto de programação introdutória. Khairuddin e Hashim (2008) abordam o seu uso como mecanismo de aperfeiçoamento de avaliações de engenharia de software. Alaoutinen e Smolander (2010) apresentam uma ferramenta de auto avaliação, baseada na Taxonomia de Bloom, para motivação do aprendizado de programação. Sorva (2012) afirma que a Taxonomia de Bloom permite caracterizar as atividades de programação em termos de complexidade cognitiva. Associações com o tema também foram encontradas em outros estudos mais antigos, os quais investigaram os fatores psicológicos envolvidos no processo de ensino/aprendizagem.

Hannafin et al. (1988) relatam como as atividades práticas tendem a aumentar a memorabilidade da informação, através da exposição dos alunos às informações adicionais, as quais criam mais conexões cognitivas e são de suma importância no processo de reconstrução da informação principal.

Hannafin e Rieber (1989) discutem princípios psicológicos para modelos de Instrução Baseada em Computador (CBI). O artigo analisa o impacto do uso dessas metodologias para o processo de aprendizado. Os autores fazem referências à teoria *dual-coding* (PAIVIO, 1979), afirmando que essa teoria é o suporte mais relevante no estudo das imagens no processo cognitivo humano. Fazem referência também ao modelo de “multe armazenamento da memória” (ATKINSON ; SHIFFRIN, 1968) afirmando que o processo cognitivo de recordação envolve a transferência de informação da memória de curto prazo (*short-term*) para a memória de longo prazo (*long-term*).

Outra afirmação feita por Hannafin e Rieber (1989) e que motiva o estudo da psicologia cognitiva, é que os projetos instrucionais baseados em computador têm tido fracasso, pois, ao invés de se discutir a melhor forma para o uso das tecnologias com o intuito de atender às demandas do aprendizado, têm se utilizado as capacidades tecnológicas vigentes para ditar as diretrizes do ensino. Os autores concluem que o principal objetivo da instrução é proporcionar atividades que geram informações contextuais que tornem o processo de aprendizagem mais profundo, apoiando o processo de recordação das informações através da criação de um

“plano de retorno” mental que ajude a memória de longo prazo (*long-term memory*) a encontrar a informação desejada.

Abd-El-Hafiz e Basili (1996) apresentam uma abordagem para análise e classificação de estruturas de repetição (**`loops`**) baseada em lógica de primeira ordem (*first-order logic*). Os autores afirmam que o processo de compreensão de código de programação está associado às unidades de conhecimentos denominadas *Plans*, que em programação representam soluções estereotipadas para determinados problemas, sendo esta definição inspirada em estudos cognitivos.

Burkhard (2004) aborda o tema “*knowledge visualization*”, que estuda as vantagens da representação visual para a transferência de conhecimento. O autor utilizou tanto fundamentações da psicologia Gestalt (KOFFKA, 1935), para fazer afirmações sobre a capacidade do cérebro de reconhecer padrões em imagens, como as afirmações de Miller (1956), para relatar sobre como a capacidade cognitiva humana é aumentada quando habilidades visuais são utilizadas. O autor ainda salienta a necessidade de se criar ferramentas de visualização direcionadas para cada área do conhecimento, considerando-se suas características específicas.

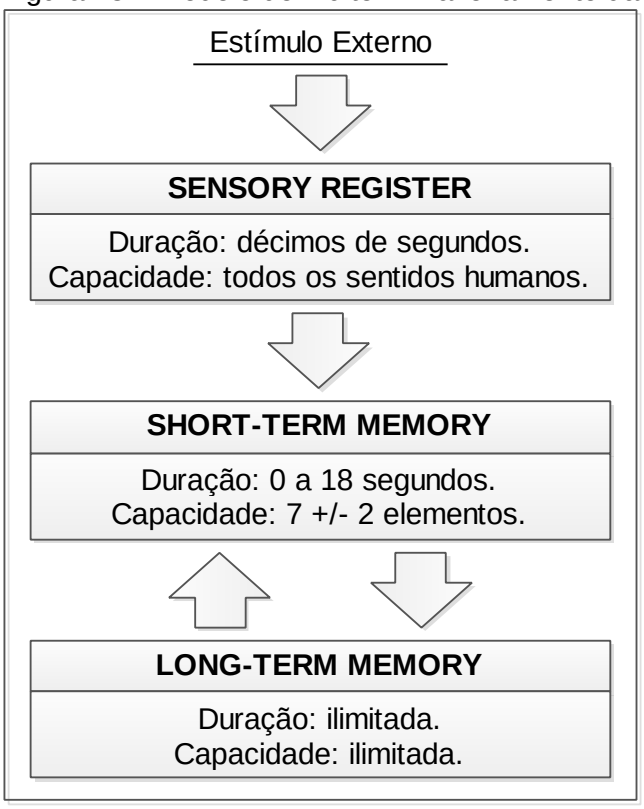
Ficou clara a relação destes trabalhos com assuntos da psicologia. Eles demonstraram a relevância e importância de se considerar os fatores cognitivos para o estudo de um modelo voltado ao ensino de programação de computadores, pois entende-se que por meio dessa abordagem é possível melhor compreender os aspectos inerentes ao processo de ensino/aprendizagem.

3.3 O MODELO DE MULTE ARMAZENAMENTO DA MEMÓRIA

Antes de abordarmos quaisquer teorias da psicologia, faz-se necessário introduzir o conceito do Modelo de Multe Armazenamento da Memória (*Multi Store Model of Memory*), proposto por Atkinson e Shiffrin (1968). Tal modelo descreve a memória humana como um mecanismo de processamento de informações que navegam em um sistema formado por uma série de blocos de armazenamento (Figura). O

modelo descreve três tipos de memórias da mente: a *sensory register* – SR, a *short-term memory* - STM e a *long-term memory* - LTM.

Figura 13 – Modelo de Multe Armazenamento da Memória



Fonte: Adaptado de Atkinson e Shiffrin (1968).

A SR, ou registrador sensorial, é o primeiro tipo de memória. Ela é responsável por reter de forma imediata o estímulo externo captado pelos sentidos humanos. Segundo McLeod (2007b) este tipo de memória retém a informação por frações de segundos. O autor também relata que existe um tipo de memória sensorial para cada sentido.

O segundo tipo é a STM, também denominada *short-term store*, *working memory* ou simplesmente *memória de curto prazo*. Ela é responsável por armazenar de forma temporária as informações recém decoradas, advindas do SR, e as informações recordadas, advindas da LTM. É memória STM que ficam retidas as informações utilizadas durante os processos cognitivos do pensamento. Ela possui a capacidade de armazenar simultaneamente entre cinco e nove elementos, segundo a teoria de Miller (1956), por um curto período que vai de zero a dezoito segundos.

A memória LTM, também denominada *long-term store*, é o tipo de memória permanente, responsável por armazenar de forma fixa as informações aprendidas pelo indivíduo e por fornecer à memória STM as informações necessárias aos processos cognitivos. Ex.: lembrar de um fato que aconteceu, ou de uma habilidade aprendida há anos.

Visto que a memória LTM é a responsável por armazenar o conhecimento aprendido, entende-se que o aprendizado consiste na transferência da informação da STM para a LTM. Neste sentido, Atkinson e Shiffrin (1968) afirmam que tal transferência depende de dois fatores: (I) a quantidade de tempo que a informação permanece na memória STM; e (II) os processos cognitivos controlados pelo próprio ser. Fica claro que o *Modelo de Multe Armazenamento da Memória* representa a abstração utilizada pela psicologia cognitiva para compreender o sistema humano de armazenamento de informações. Serão relatadas nas próximas sessões teorias que fazem uso deste modelo para fundamentar e conduzir seus experimentos científicos.

3.4 A TEORIA DO NÚMERO MÁGICO

Em seu trabalho denominado "*The magical number seven, plus or minus two: some limits on our capacity for processing information.*" Miller (1956) forneceu evidências científicas acerca da limitação de armazenamento de informações da memória STM. Sua teoria sugere que a maior parte dos adultos possuem a capacidade de armazenar uma média de sete, mais ou menos dois, elementos de forma simultânea neste tipo de memória (Figura). Collin (2012) ratifica a importância da teoria de Miller (1956), afirmando que o trabalho dele representou um marco para a psicologia cognitiva e o estudo da memória.

Figura 14 – Capacidade da memória de curto prazo

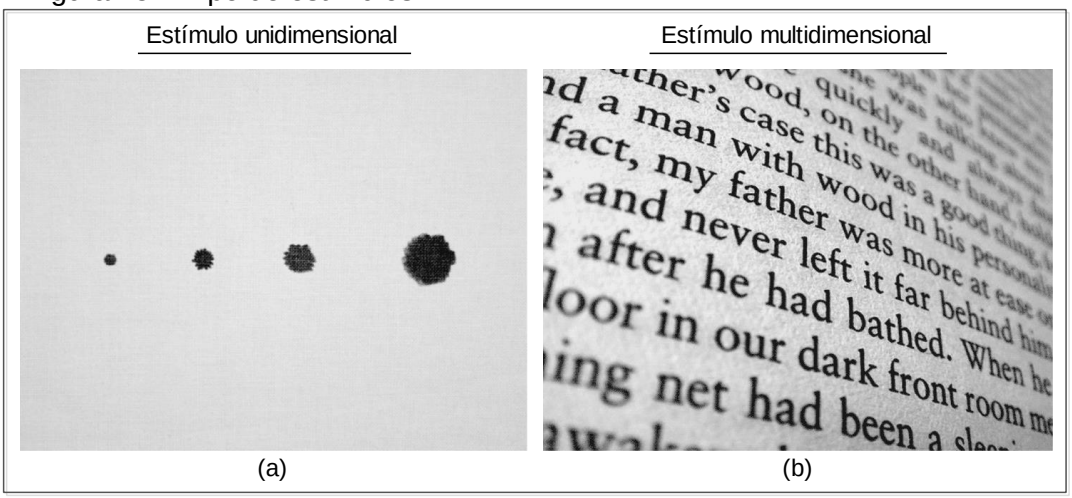
Fonte: Autor desta dissertação (2016).

Miller (1956), enxergava a mente humana como um sistema de comunicação. Um ambiente de recepção de informações sensoriais, no qual a quantidade de informação transferida para o cérebro é diretamente proporcional à quantidade de informação que entra no sistema. Essa relação se mantinha até que um determinado limiar de entrada de informações fosse atingido, limiar esse denominado *capacidade do canal*.

O autor dividia as informações que entravam no sistema em duas categorias:

1. Estímulos *unidimensionais* – informações que se diferem umas das outras em apenas um aspecto. Ex: notas musicais, pontos na tela (Figura a).
2. Estímulos *multidimensionais* – informações que se diferem umas das outras em vários aspectos. Ex: palavras, (Figura b).

Figura 15 – Tipo de estímulos



Fonte: Autor desta dissertação (2016).

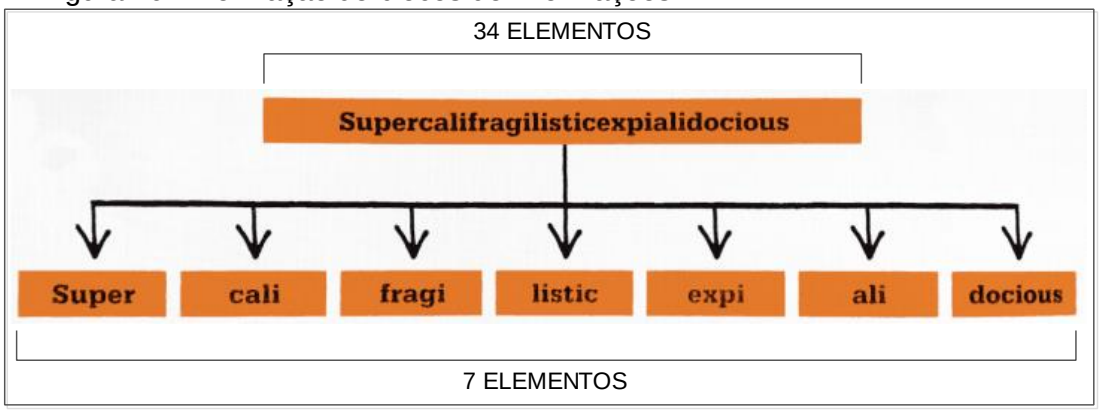
Em seu artigo, Miller (1956), faz referência ao trabalho de Pollack (1952) que realizou um experimento científico onde os participantes atribuíam números à um grupo de notas musicais tocadas. O objetivo era identificar a precisão com a qual os seres humanos conseguem distinguir estímulos unidimensionais distintos. Miller afirma que os resultados de Pollack (1952), demonstraram que, ao passo que a quantidade de notas tocadas ultrapassava sete, aumentava a dificuldade dos participantes em atribuir números às notas.

A partir de então, Miller (1956), buscou investigar se este limiar de sete elementos se mantinha para os estímulos multidimensionais e surpreendentemente constatou, em estudos posteriores de Pollack, que este limiar de sete elementos sofria apenas um ligeiro declínio com o aumento das variáveis, ou seja, o limite de sete elementos tende a ser mantido independentemente da complexidade da informação. Miller (1956), passou então a considerar os estímulos multidimensionais como grandes pedaços de informações inter-relacionadas, porém tratados pela memória STM como um único elemento. O autor acreditava que este era o mecanismo que a memória utilizava para lidar com grandes quantidades de informações.

A Figura mostra um exemplo do conceito proposto por Miller, retirado de Collin (2012), onde é possível perceber um agrupamento de letras que representam trinta e quatro elementos de informações unidimensionais (letras), mas que quando agrupadas em conjuntos resultam em sete elementos de informações

multidimensionais (palavras), as quais, segundo a teoria de Miller (1956), cabem perfeitamente na memória humana de curto prazo.

Figura 16 – Formação de blocos de informações



Fonte: Collin (2012).

> Segundo a teoria da formação de blocos de Miller, ao construir ou quebrar longas sequências de números ou letras e agrupá-los em blocos memorizáveis, aumentamos a quantidade de informação que podemos armazenar na memória 'de trabalho'. (COLLIN, 2012, p. 172).

Fica óbvio que a proposta da formação de blocos, como mecanismo utilizado pela mente para lidar com grandes quantidades de informações, foi outra grande contribuição de Miller para a psicologia cognitiva. Sua teoria, além de fornecer evidência sobre a real capacidade de armazenamento da mente humana de curto prazo (em torno de sete elementos), serviu de fundamentação para diversas pesquisas acerca do estudo da memória.

3.5 CONCLUSÕES

Durante a discussão deste capítulo ficou clara a relação de diversos estudos científicos que relacionam processos instrucionais com conceitos da psicologia cognitiva. Isso mostrou o quanto este tema é importante para a área de ensino/aprendizagem.

Foi mostrado também que a psicologia cognitiva utiliza um sistema de informação como metáfora para estudar os aspectos internos do ser humano (Figura). Tal sistema descreve a mente como um conjunto de sistemas de armazenamento distintos (Figura), sobre ao quais acontecem todos os processos do pensamento e

do aprendizado. Neste contexto, o estudo de Miller (1956) mostrou que a memória humana é limitada no que diz respeito a trabalhar com grandes quantidades de informações. O autor demonstrou também que, apesar das limitações, o uso de técnicas como a formação de blocos (Figura) permite que a mente consiga gerenciar grandes volumes de dados de forma simultânea.

Este capítulo teve o objetivo de demonstrar que a investigação e entendimento do funcionamento da mente humana, por meio da psicologia cognitiva, é condição fundamental para que se possa propor melhores formas de ensinar. Entende-se que um modelo instrucional deve levar em consideração a limitação da mente, restringindo a exposição de grandes quantidades de informações, no âmbito de facilitar o processo de absorção.

4 MODELO LÚDICO PARA O ENSINO DE PROGRAMAÇÃO

Este capítulo discute o processo de criação do modelo lúdico para o ensino de conceitos de programação proposto por esta dissertação. Inicialmente são discutidos os motivos que levaram à escolha do conceito de recursividade para ser implementado pelo modelo. Depois é realizado um estudo deste conceito escolhido e são analisadas as complexidades impostas à sua compreensão. Por conseguinte, é apresentada a completa especificação do modelo lúdico, destacando suas características de usabilidade. Em seguida são apresentadas as fundamentações científicas do modelo. E, por fim, são especificados os detalhes da realização deste modelo, por meio da implementação de três algoritmos recursivos.

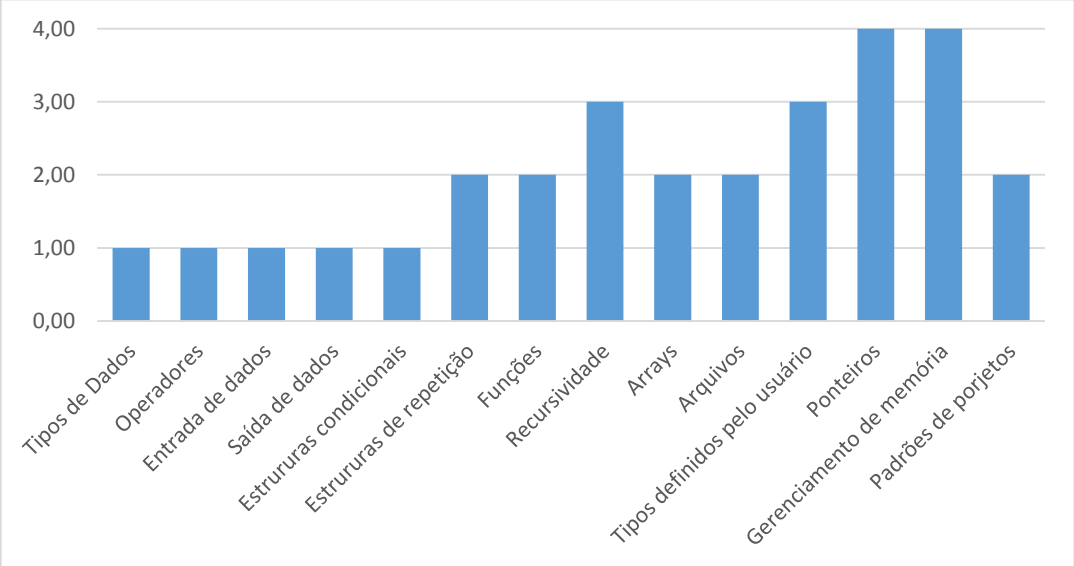
4.1 INTRODUÇÃO

Como demonstrou o mapeamento sistemático, a programação é complexa de ser aprendida e ensinada, não só devido à falta de maturidade do ensino e do aluno, mas também devido à grande quantidade de processos cognitivos envolvidos durante a tarefa. A programação é uma atividade que reúne uma série de outras complexas atividades, as quais, requerem um razoável tempo e esforço para serem desenvolvidas.

O estudo de Renumol et al. (2010) concluiu que os programadores mais eficientes possuem maior facilidade para leitura de código não pelo fato de dominarem o ambiente de desenvolvimento em que trabalham, mas por dominarem de forma mais profunda os processos de programação envolvidos na tarefa. Entende-se, com isso, que os programadores experientes possuem bons modelos mentais acerca dos padrões modelados pelas linguagens de programação. Portanto, buscou-se fornecer aos alunos uma forma lúdica de adquirir familiaridade com tais padrões, sem a necessidade de lidar com nenhum código textual, pois a proposta a ser apresentada por este trabalho está fortemente relacionada com a proposta de Covington e Benegas (2005), que defendem uma abordagem baseada na criação de *Schemas* relacionados aos processos de programação no lugar de uma abordagem textual.

Para contemplar este aspecto, foi utilizada uma metodologia lúdica baseada em jogos análoga à proposta de Eagle et al. (2008), que desenvolveu uma metáfora para a demonstração de conceitos de ***"loops"*** e ***"arrays"***. Neste contexto, foi preciso inicialmente definir qual conceito a ser abordado no modelo proposto, pois diversos são os conceitos de programação a serem aprendidos pelos alunos iniciantes dos cursos de computação e engenharias. Alaoutinen e Smolander (2010) relatam uma série deles, bem como seus respectivos graus de dificuldade, ilustrados na Figura .

Figura 17 – Grau de dificuldade dos conceitos de programação



Fonte: Adaptado de Alaoutinen e Smolander (2010).

A partir da análise dos conceitos computacionais, optou-se por abordar o conceito de recursividade. As principais prerrogativas para a escolha deste foram:

a) A não identificação na bibliografia científica de uma abordagem lúdica, ou quaisquer outras metodologias não baseadas em código, para o ensino deste conceito.
b) A influência de Wilcocks e Sanders (1994, p.221), que relatam a preocupação de diversos autores da bibliografia com relação à abordagem estática com a qual a recursividade é tratada nas universidades.
c) O auto grau de dificuldade, segundo Alaoutinen e Smolander (2010), para o seu aprendizado.

Portanto, entende-se que é válido fazer uso do conceito de recursividade para a definição de uma proposta lúdica para o ensino de conceitos de programação de computadores. Contudo, fez-se antes necessário um entendimento mais aprofundado sobre a recursividade, no âmbito de melhor entender a sua real complexidade.

4.2 O ALGORITMO RECURSIVO

Segundo Odifreddi (1999, p.3), o conceito de algoritmo recursivo é uma generalização da noção matemática de recursividade, a qual tem sua origem na Teoria dos Números. A recursividade pode ser melhor entendida como a repetição sucessiva de um mesmo procedimento. A Figura ilustra uma metáfora para a percepção do 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.

Figura 18 – Recursividade entre espelhos

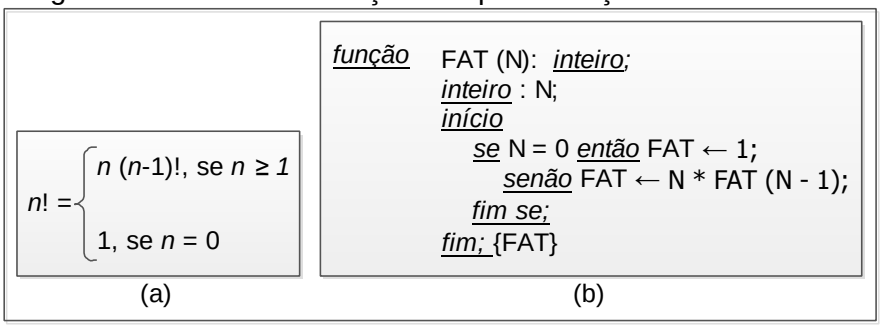
Fonte: Inception (2010).

Em programação este conceito é observado nas funções recursivas, que chamam a si mesmas, direta ou indiretamente, no mínimo uma vez (GUIMARÃES; LAGES, 2012, p. 141). A Figura demonstra o conceito de função recursiva aplicado na resolução do fatorial de um número inteiro. A Figura a ilustra a definição matemática

do conceito de fatorial e a Figura b ilustra uma implementação em PORTUGOL para a sua resolução computacional usando uma função recursiva.

Na implementação apresentada na Figura b, 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 19 – Fatorial: definição e implementação



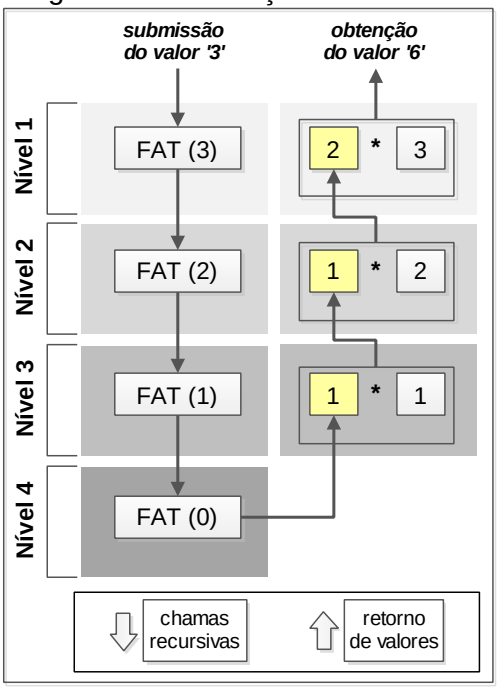
Fonte: Adaptado de Guimarães e Lages (2012).

Supondo que seja submetido inicialmente o valor ‘3’ à função recursiva ilustrada por Guimarães e Lages (2012), 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 ilustra o processo de resolução do fatorial de ‘3’ pelo algoritmo ilustrado na Figura b. Nela é possível perceber (lado esquerdo da Figura), 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 , é possível perceber o

mecanismo utilizado pela recursividade para achar o valor 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’.

Figura 20 – Execução do cálculo fatorial recursivo



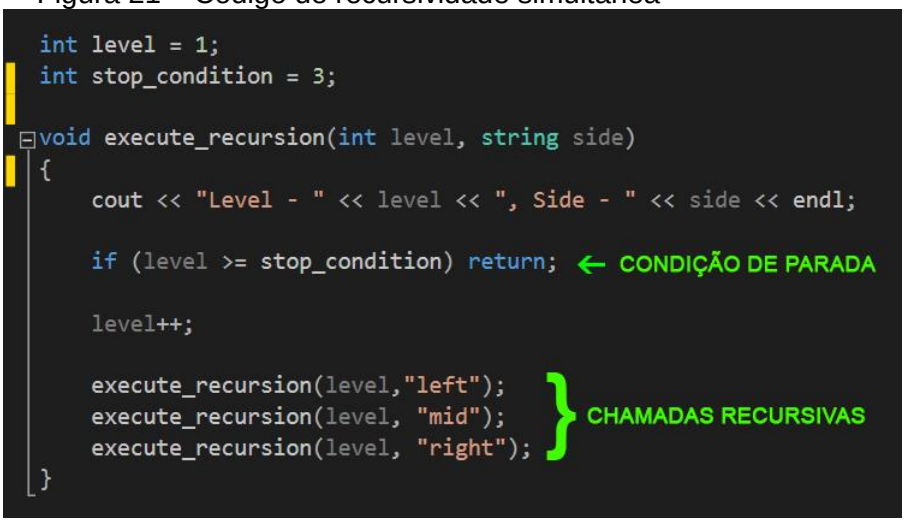
Fonte: Autor desta dissertação (2016).

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 afirmam 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.

É importante observar que, no exemplo do fatorial ilustrado na Figura , obtemos 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 a complexidade da estrutura hierárquica das chamadas recursivas é a quantidade de chamadas simultâneas, que é a quantidade de chamadas recursivas dentro de uma mesma chamada. A Figura apresenta um trecho de código escrito em ***C++*** que ilustra esse caso. 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.

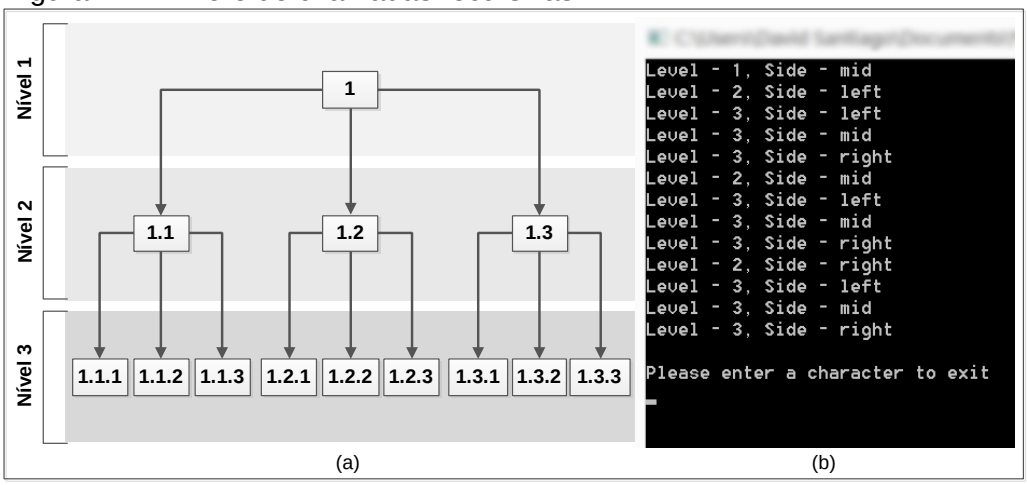
Figura 21 – Código de recursividade simultânea



Fonte: Autor desta dissertação (2016).

Diferente das chamadas recursivas simples, as chamadas recursivas simultâneas criam uma estrutura hierárquica em forma de árvore, que apresentam um comportamento de crescimento ao longo dos níveis de profundidade. A Figura a ilustra a hierarquia criada pelo algoritmo demonstrado na Figura . Nela é possível observar como o número de chamadas é multiplicado por três a cada nível da recursão. A Figura b ilustra a saída gerada pelo algoritmo.

Figura 22 – Árvore de chamadas recursivas



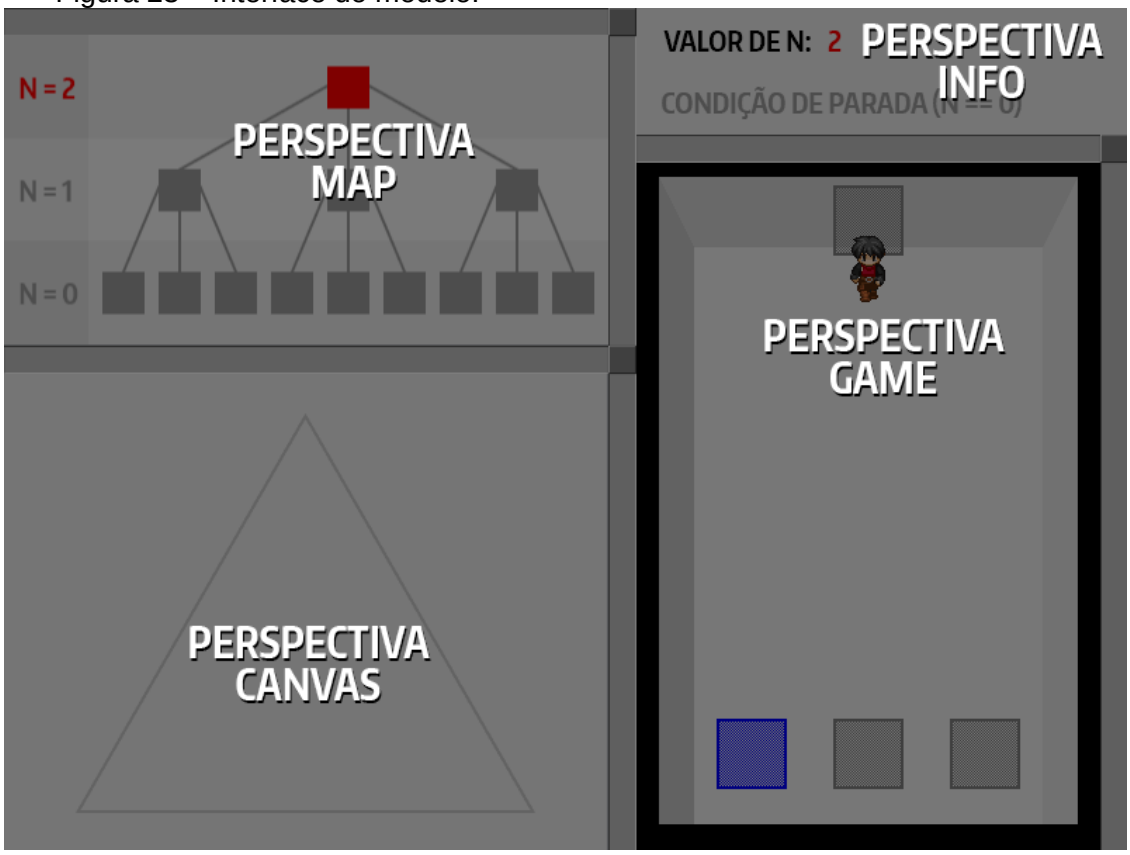
Fonte: Autor desta dissertação (2016).

Salienta-se que esta estrutura possui treze chamadas para três níveis de recursão. Se 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. Portanto, a partir da análise da estrutura de chamadas dos algoritmos recursivos, entende-se que o principal objetivo no ensino deste conceito é fazer com que o aluno se familiarize com a complexidade da estrutura de navegação hierárquica imposta pela sua execução.

4.3 CARACTERIZAÇÃO DO MODELO

Uma vez compreendida a complexidade da navegação hierárquica imposta pela recursividade, o modelo descrito por esta dissertação 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 encotra e a representação visual do problema modelado.

Figura 23 – Interface do modelo.



Fonte: Autor desta dissertação (2016).

A interface do modelo proposto (Figura) fornece quatro perspectivas distintas:

a) **Game** – painel de interação, onde o usuário controlará o fluxo de execução recursivo por meio da movimentação do personagem.
b) **Canvas** – representação gráfica do conceito modelado pelo algoritmo recursivo.
c) **Map** – apresenta um mapa conceitual de navegação hierárquica imposta pelas repetições.
d) **Info** – apresenta as informações textuais referentes ao nível da navegação hierárquica e os estados das variáveis impostas pelo algoritmo recursivo.

Estas perspectivas devem ser sincronizadas, fornecendo ao aluno a possibilidade de observar em tempo real a navegação (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:

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

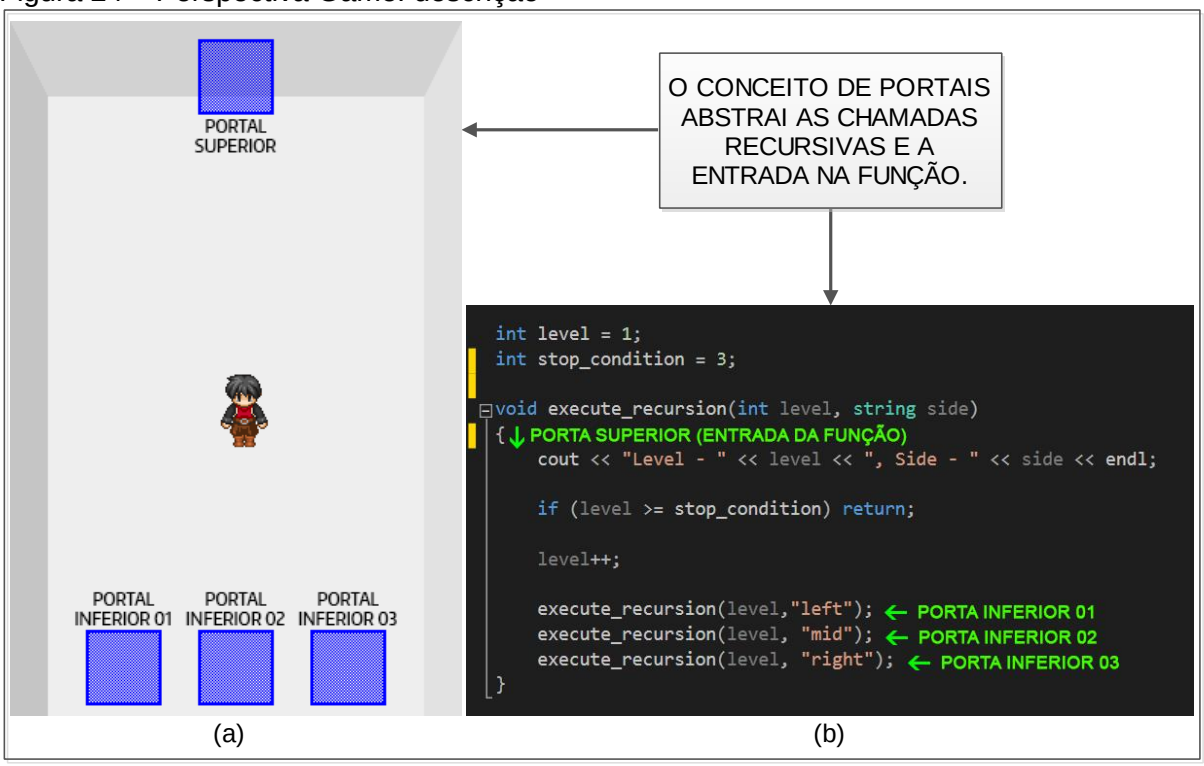
4.3.1 Perspectiva Game

A perspectiva *Game* representa a abstração do código de uma função recursiva. Ela tem o objetivo tem o objetivo de permitir que o aluno controle o personagem, que representa a forma lúdica com a forma lúdica com a qual ele interage com o fluxo de execução do algoritmo. Sua interface (interface (

Figura a), ilustra o cenário proposto pelo modelo. Este deve ser constituído por portais, sendo apenas um na parte superior e no mínimo um na parte inferior. A

Figura b indica a característica do código recursivo que foi ilustrada através do conceito de portais. O portal superior se refere à chegada de um novo fluxo recursivo, representado no código pela entrada na função. Os portais inferiores representam o acesso a novos níveis de execução recursiva, representado no código pelas novas chamadas.

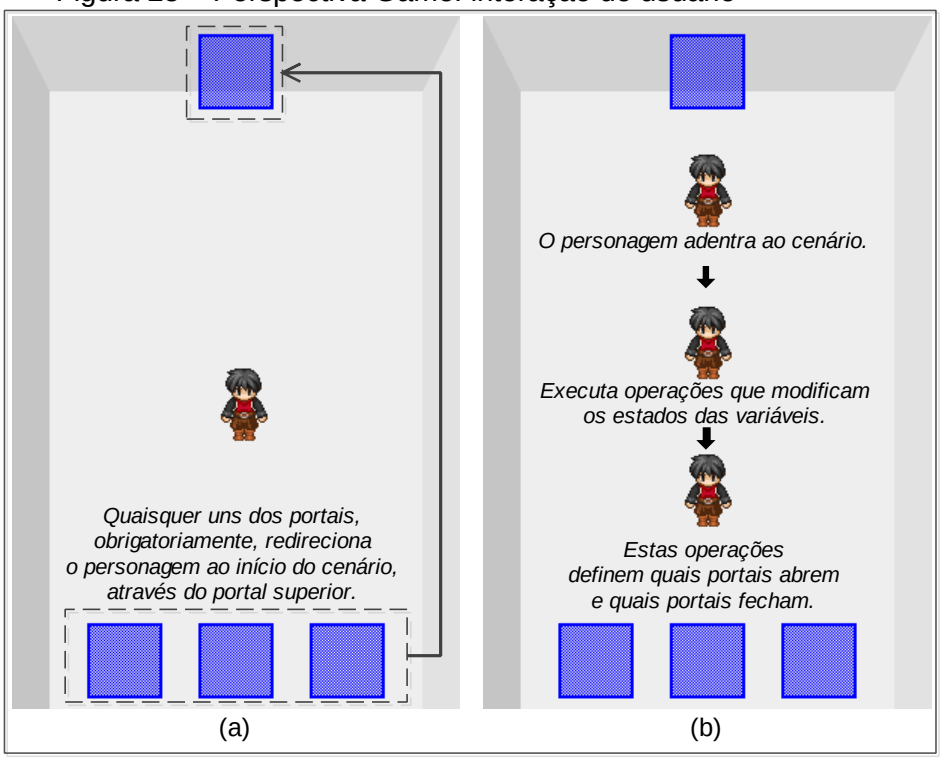
Figura 24 – Perspectiva Game: descrição



Fonte: Autor desta dissertação (2016).

A interação do jogador se resume a adentrar ao cenário, por meio do portal superior e, por meio de quaisquer portais da parte inferior, retornar ao mesmo cenário pelo portal superior, simulando exatamente o que acontece no fluxo de execução do algoritmo recursivo. A Figura a ilustra este processo. É importante salientar que em cada execução recursiva há alterações no estado do programa, principalmente nas variáveis que controlam a condição de parada das chamadas recursivas. Portanto, o termino do fluxo de repetição foi representado através do trancamento dos portais inferiores, que informarão ao aluno que o personagem não mais poderá adentrar em níveis recursivos mais profundos. A Figura b ilustra o processo básico que deverá ser realizado pelo usuário a cada execução recursiva.

Figura 25 – Perspectiva Game: interação do usuário



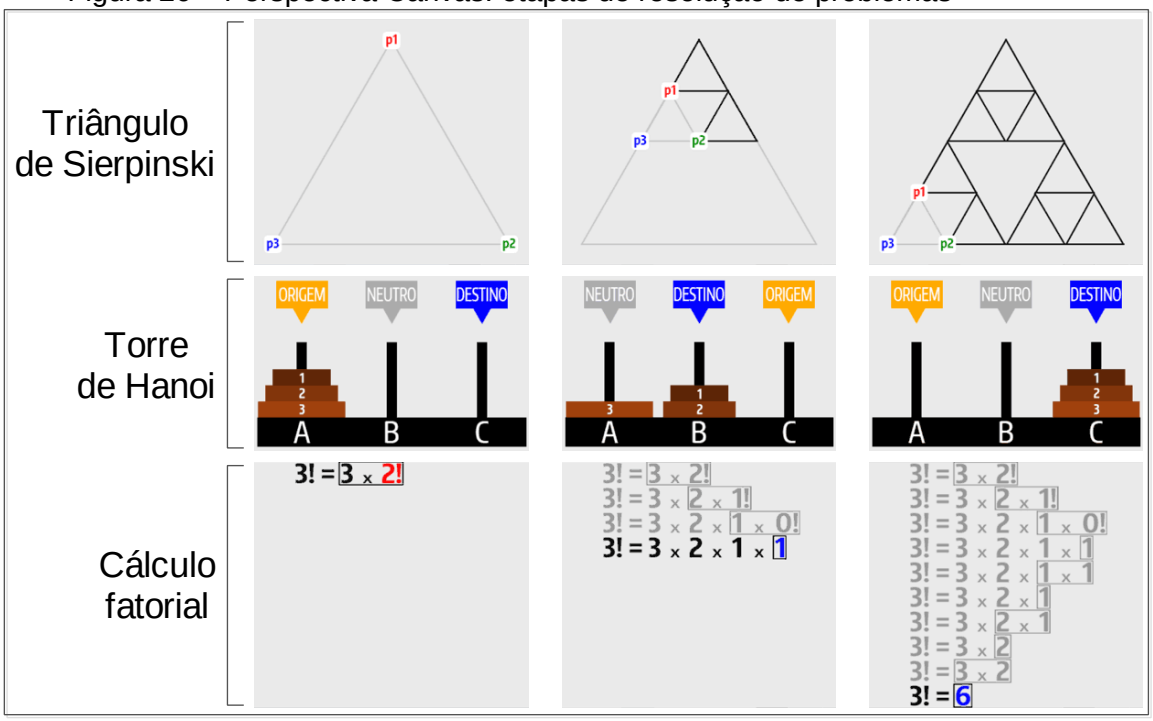
Fonte: Autor desta dissertação (2016).

4.3.2 Perspectiva Canvas

A perspectiva *Canvas* é a janela que 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 ilustra algumas saídas gráficas propostas para o algoritmo recursivo de construção do triângulo de Sierpinski, da Torre de Hanoi e do cálculo Fatorial.

Figura 26 – Perspectiva Canvas: etapas de resolução de problemas

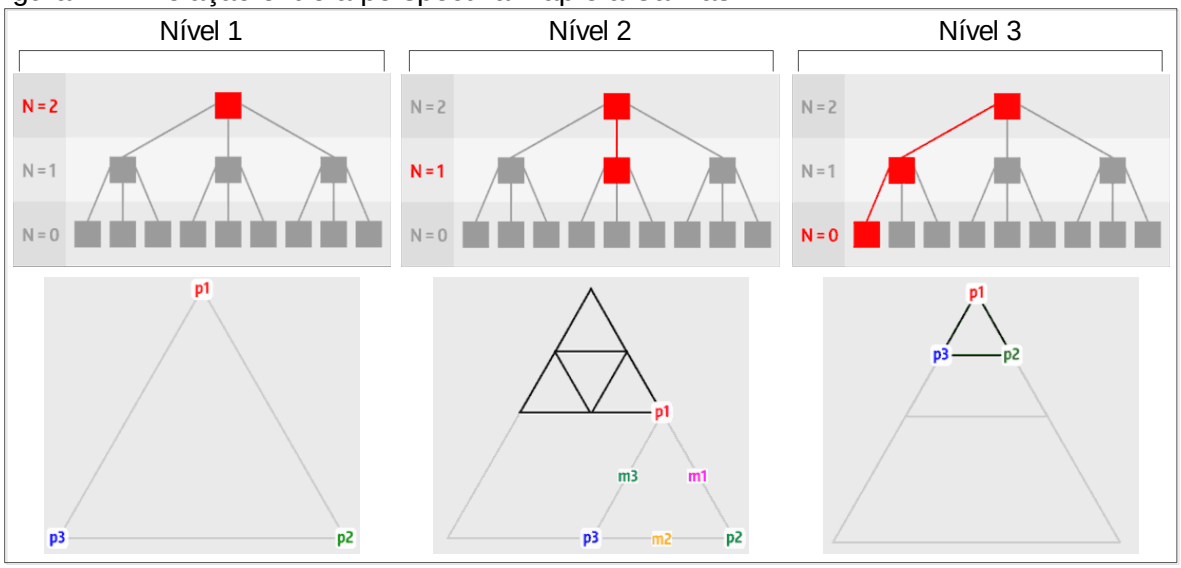


Fonte: Autor desta dissertação (2016).

4.3.3 Perspectiva Map

Esta perspectiva permite que o aluno se familiarize com a navegação hierárquica de chamadas imposta pelos algoritmos recursivos. A Figura ilustra como ela fornece ao aluno a capacidade de correlacionar o nível de navegação hierárquica em que ele se encontra (perspectiva *Map*) com o a saída produzida (perspectiva *Canvas*).

Figura 27 – Relação entre a perspectiva Map e a Canvas



Fonte: Autor desta dissertação (2016).

Este tipo de visualização é imprescindível para que o aluno consiga concluir todas as etapas do algoritmo. Ela evita que ele se perca durante o processo de execução, dando-o, a todo momento, a consciência de qual etapa recursiva se encontra.

4.3.4 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 algoritmicas relevantes, caso sejam necessárias.

4.4 FUNDAMENTAÇÃO CIENTÍFICA DO MODELO

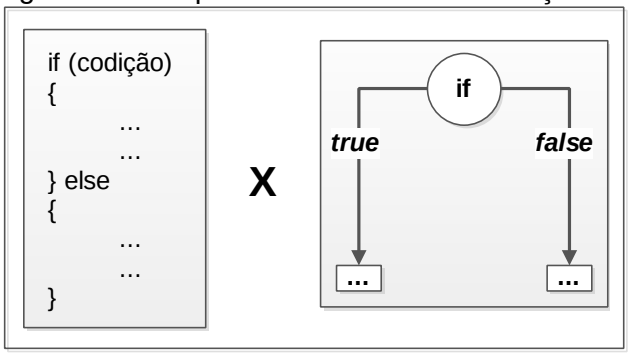
Com base nas afirmações identificadas no estudo da psicologia cognitiva, fundamentamos a construção do nosso modelo à luz de três critérios:

1. ***Critério de Apresentação*** – Justificamos a necessidade de se utilizar notações visuais no lugar de textual.
2. ***Critério de Partição*** – Justificamos a necessidade de gerenciar a quantidade de informações visuais apresentadas, devido às limitações da mente humana.
3. ***Critério de Repetição e Agrupamento*** – Justificamos a necessidade de se utilizar a repetição como mecanismo fundamental para a construção do conhecimento e o desenvolvimento das habilidades.

4.4.1 Critério de Apresentação

Este critério justifica a importância do uso de notações visuais para a demonstração de conceitos computacionais. Covington e Benegas (2005) afirmam que os alunos são mais propensos a desenvolverem *Schemas* a partir de apresentação, exemplos e exercícios do que a partir de explicações baseadas em códigos textuais. Os autores ainda afirmam que os conceitos de computação são melhor comunicados em forma de notações visuais, tais como diagramas ou modelos, do que em demonstração de código de programação. Eles relatam que desta forma consegue-se mostrar ao aluno iniciante que o modelo de um problema sugere um padrão de solução que pode ser aplicado a diversos outros problemas que possuem a mesma forma. Segundo os autores, esse é o conhecimento mais valioso que o aluno deve dominar, mais importante até do que detalhes de *if-else*. A Figura ilustra um exemplo de como eles sugerem o uso de notação visual para representar o conceito de estruturas condicionais.

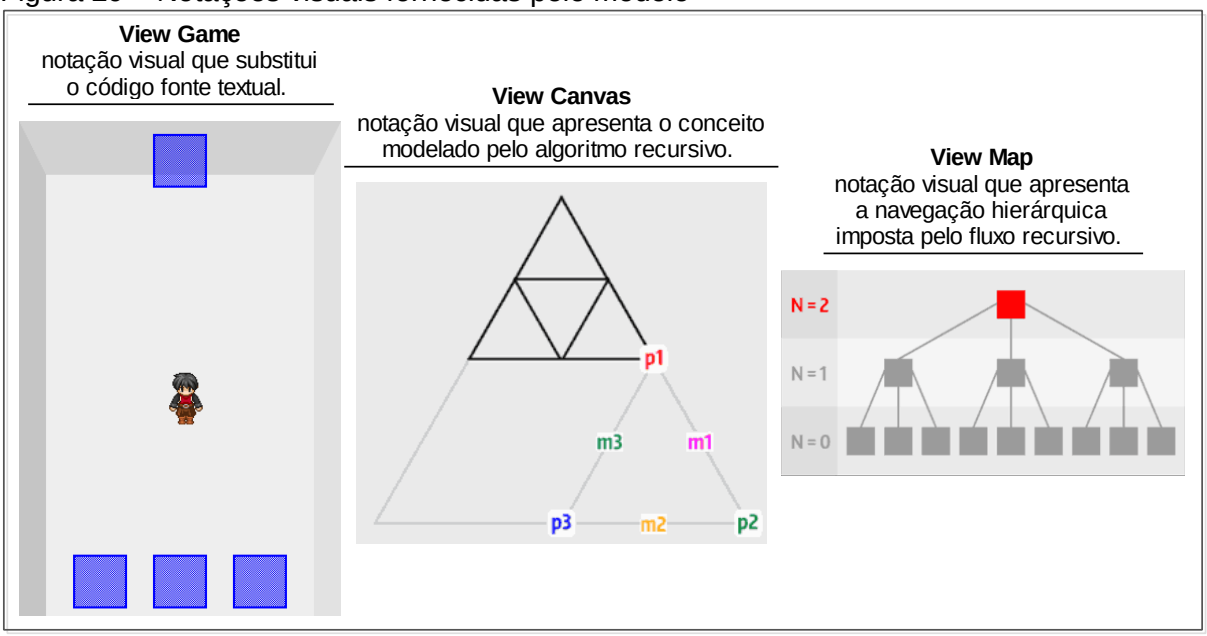
Figura 28 – Esquema de decisão em notação textual e visual



Fonte: Adaptado de Covington e Benegas (2005).

No modelo proposto nesta dissertação, as notações visuais são fornecidas por meio das perspectivas *Game*, *Canvas* e *Map*. Elas permitem que o aluno observe a abstração do fluxo de execução do conceito de recursividade de forma dinâmica, podendo acompanhar simultaneamente as várias informações a respeito da execução das etapas algorítmicas. Na Figura é possível observar as três notações visuais fornecidas para o aluno durante o fluxo de execução algorítmico.

Figura 29 – Notações visuais fornecidas pelo modelo



Fonte: Autor desta dissertação (2016).

Este mecanismo de apresentação visual baseado em jogos tem o objetivo de prover o **aprendizado declarativo** relatado por Shaffer et al. (2003) e fornece aos alunos os conhecimentos iniciais necessários para o desenvolvimento das habilidades. Entende-se que um modelo visual simples e explícito, que guarda similaridades com o modelo textual complexo, é a melhor maneira de prover o aprendizado declarativo e evitar o processo de Rote Learning (COVINGTON ; BENEGAS, 2005), no qual os alunos têm de decorar informações descontextualizadas, como é o caso do aprendizado por código de programação.

4.4.2 Critério de Partição

Este critério aborda o processo de divisão e classificação dos elementos gráficos utilizados na descrição do modelo, com o intuito de reduzir a quantidade de

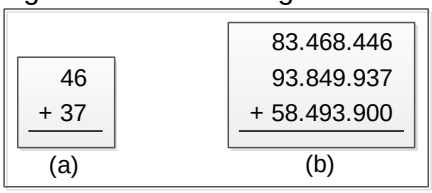
informações apresentadas e, consequentemente, a carga cognitiva despendida pelos alunos com os aspectos da apresentação do conteúdo instrucional.

O critério de partição está fortemente fundamentado nas evidências científicas identificadas no trabalho de Miller (1956), sobre a limitação de capacidade da mente humana em armazenar estímulos sensoriais e de como estes estímulos impõem esforços que impactam no desempenho do aprendizado (Figura), e nas fundamentações de Collin (2012) sobre o processo de formação de blocos de informações (Figura).

Segundo Morrison et al. (2014) o desempenho do estudante está diretamente relacionado à quantidade de carga cognitiva. Caspersen e Bennedsen (2007) afirmam que o aprendizado é maximizado quando há um balanceamento destas cargas.

Shaffer et al. (2003) afirmam que, se a capacidade da mente humana de sete elementos for excedida, então algumas ou todas as informações podem ser perdidas. Os autores propõem um exemplo para explicar como o excesso de informações (sobrecarga cognitiva) prejudica o processo de retenção das mesmas. O exemplo é ilustrado na Figura .

Figura 30 – Somas algébricas



Fonte: Shaffer et al. (2003).

Na Figura é possível observar duas operações algébricas idênticas de adição, a diferença está na quantidade de informações. A dificuldade imposta pela segunda operação (Figura b) se dá pelo fato de que a quantidade de estímulos propostos pelo problema excede a capacidade de sete elementos da memória de curto prazo. Neste sentido, os autores afirmam que o lápis e o papel funcionam como uma espécie de extensão da memória curto prazo, permitindo a recordação imediata dos estímulos, que não cabem na mente, durante o processo de resolução da operação.

Estas fundamentações, mostram à inevitável necessidade de gerenciar, no modelo proposto, a quantidade de informações simultâneas a serem apresentadas aos alunos. Neste sentido, buscou-se mitigar a quantidade de estímulos sensoriais, no âmbito de evitar sobrecarga cognitiva e a consequente perda de eficiência no processo de aprendizado. Contudo, no modelo proposto, não houve a delimitação restrita aos sete estímulos sensoriais. Entende-se que a quantidade de estímulos necessários para expressar um determinado conhecimento dificilmente ficará abaixo de sete. Portanto, buscou-se assumir uma responsabilidade minimalista durante a definição do modelo, de forma que o aluno seja exposto à quantidade mínima possível de estímulos necessários à aquisição do conhecimento, evitando assim a sobrecarga cognitiva de sua memória de curto prazo. A fundamentação desta ideia é melhor explicada na Seção 0.

Como pôde ser observado na Figura , o layout foi inspirado em formas geométricas simplificadas, conjuntos de caracteres e linhas em cores sólidas, no âmbito de representar somente os elementos necessários à ilustração do processo.

Sweller et al. (1998) afirmam que as informações relacionadas aos aspectos de apresentação do conteúdo instrucional impõem esforços cognitivos classificados como *Extraneous Load*, os quais devem ser reduzidos para evitar a sobrecarga de informações e a consequente ineficiência do processo de aprendizado. A Figura (Seção 0) ilustra como a redução desta carga propicia o “balanceamento ótimo” defendido pelos autores.

Caspersen e Bennedsen (2007) afirmam que o aprendizado é otimizado quando a carga cognitiva utiliza totalmente a capacidade da *working memory* com os elementos mais significativos para o processo de *Schema acquisition*. Em outras palavras, o aprendizado se faz de maneira mais otimizada quando o esforço mental é utilizado com o mínimo de detalhes possível para a aquisição do conhecimento. Detalhes estes identificados, no modelo proposto por esta dissertação, como os aspectos visuais.

Outra evidência científica que fundamenta esta ideia é exposta por Atkinson e Shiffrin (1968), que afirmam que a eficiência da transferência do conhecimento para a memória LTM depende da quantidade de tempo que o conhecimento permanece na memória STM. Com isso, entende-se que mitigar a quantidade de estímulos sensoriais é liberar espaço na memória STM para que uma maior quantidade de informações possa ser transferida para a memória de longo prazo, permitindo assim uma melhor assimilação do conhecimento.

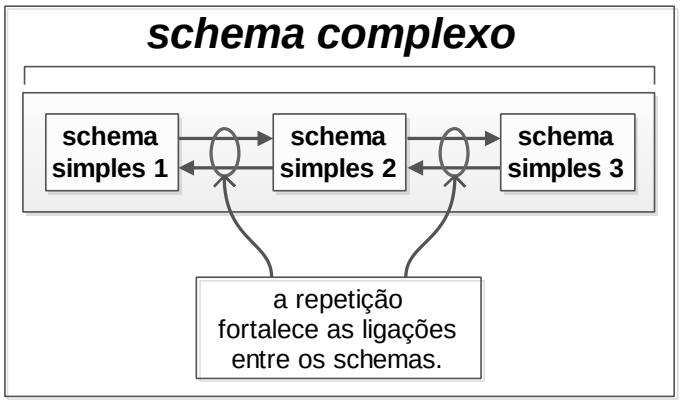
Portanto, o critério de partição buscou mitigar a quantidade das cargas cognitivas na definição dos aspectos de apresentação como forma de favorecer a assimilação do conhecimento.

4.4.3 Critério de Repetição e Agrupamento

Ficou claro que “possuir o conhecimento”, segundo a psicologia cognitiva, significa ter, armazenado na memória LTM, um conjunto de informações relacionadas umas com as outras, formando um agrupamento denominado *Schema*. O conceito de *Schema* é explicado na Seção 0.

O objetivo das metodologias fundamentadas na CLT é desenvolver no aluno um conjunto de *Schemas* de conhecimentos básicos, aos quais todas as novas informações possam ser relacionadas. Neste sentido, Sweller et al. (1998) afirmam que o desenvolvimento de habilidades se resume na construção de *Schemas* mais complexos a partir do agrupamento de *Schemas* mais simples. Este agrupamento nada mais é do que a fortificação dos relacionamentos entre os diversos *Schemas* do conhecimento, e que acontecem através do processo de repetição. A Figura ilustra este conceito.

Figura 31 – Construção de schemas complexos

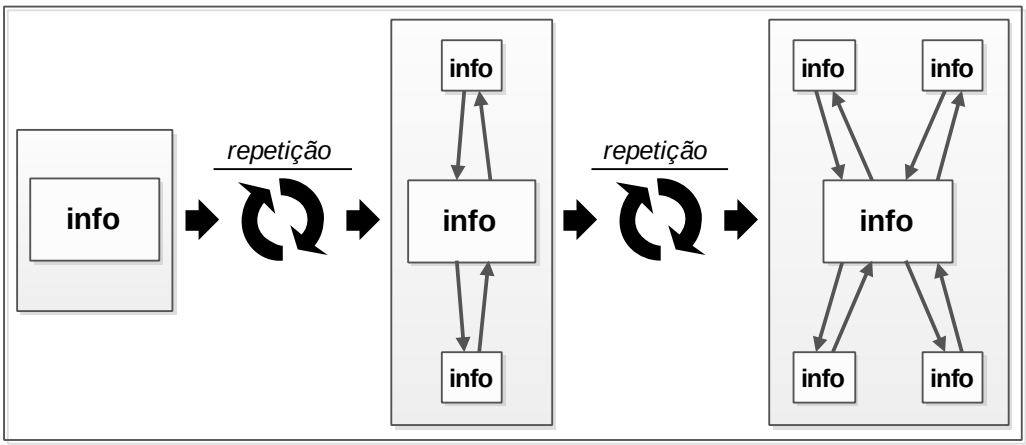


Fonte: Autor desta dissertação (2016).

O mecanismo de repetição tem o objetivo de prover o **aprendizado procedural** relatado por Shaffer et al. (2003, p.4) e define a atividade prática como mecanismo fundamental para o melhoramento dos *Schemas* acerca do conhecimento a ser aprendido. O autor deixa claro que a repetição torna o processo mental de recordação do conhecimento cada vez mais automático, pois as estruturas dos modelos mentais (*Schemas*) dos alunos que repetem vão se tornando mais ricas e organizadas.

A definição do aprendizado procedural guarda fortes semelhanças com o processo de aprendizado ótimo (*meaningful learning*) defendidos por Mayer (1981) e *Caspersen e Bennedsen (2007)* e descrito na Figura. Portanto, entende-se que o aprendizado ótimo é a consequência do aprendizado procedural. Desta forma, é possível afirmar que **a repetição permite o agrupamento** das informações ao longo do tempo. A Figura ilustra este processo.

Figura 32 – Repetição e agrupamento

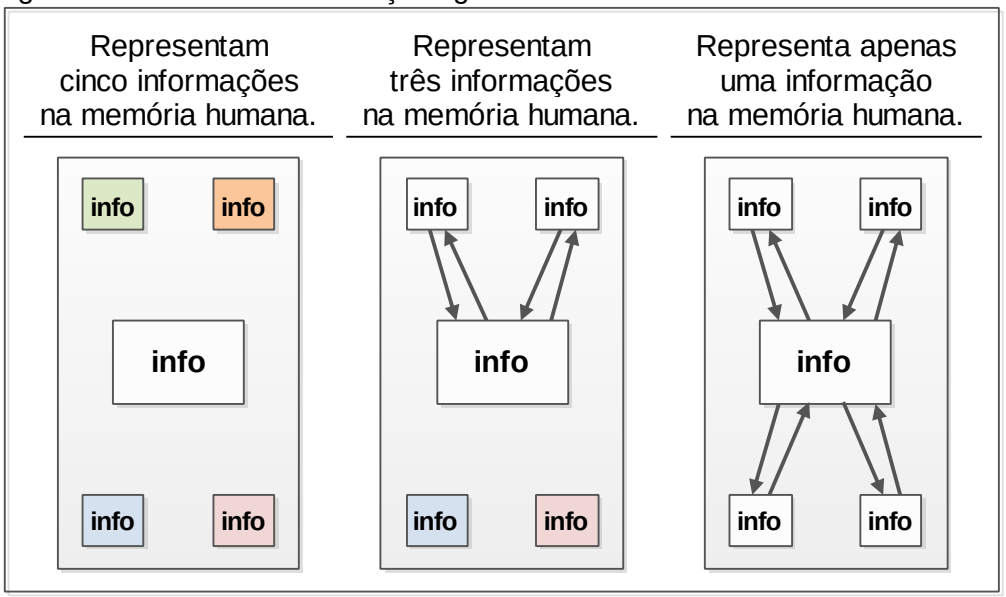


Fonte: Autor desta dissertação (2016).

Na Figura , é possível perceber que a partir do momento que se faz presente a repetição, por meio do aprendizado procedural, criam-se novos relacionamentos entres os entres os *Schemas*, gerando conjuntos únicos de informações cada vez maiores. A consequência deste processo é a redução da carga cognitiva, pois, a partir de então, o o *Schema* resultante carregará uma grande quantidade de informações, as quais ocuparão ocuparão menor espaço na limitada memória LTM. A

Figura ilustra como o processo de repetição diminui a carga cognitiva. Nela é possível perceber como a criação e o fortalecimento das relações entre os *Schemas* cognitivos de informação criam blocos únicos, os quais passam a consumir menor esforço cognitivo na memória de curto prazo.

Figura 33 – Schemas vs esforço cognitivo

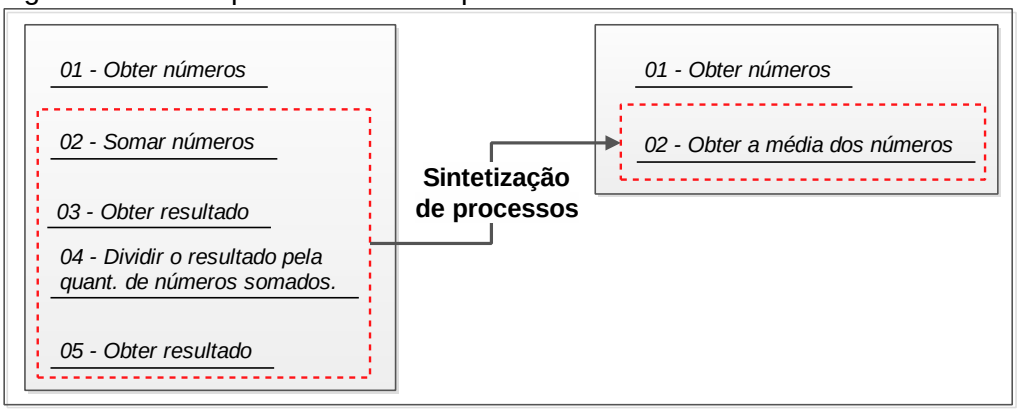


Fonte: Autor desta dissertação (2016).

Segundo Shaffer et al. (2003), é por meio desse processo de agregação de pequenos conjuntos de informações em *Schemas* de maior nível que as habilidades são desenvolvidas.

Sweller et al (1998) afirmam que, com a prática suficiente, o processo de recordação acontece de forma inconsciente. Um exemplo prático é apresentado na Figura , que ilustra como um conjunto de processos referentes à operação de média aritmética pode ser reduzido a um único processo bem definido e bem conceituado.

Figura 34 – Exemplo de schema superior



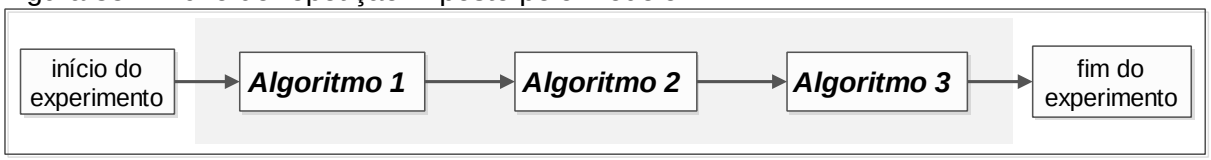
Fonte: Adaptado de Shaffer et al. (2003).

O processo de repetição durante o desenvolvimento escolar permite que o aluno sintetize estes processos em um *Schema* superior. O resultado disso, é que ele não precisa recordar todos os processos envolvidos para a realização da média aritmética, todos os processos são automaticamente recordados, pois estão muito bem-conceituados e construídos em *Schemas* superiores.

Neste Contexto, entende-se que o processo de repetição consegue reduzir a carga cognitiva. Ela permite que as informações do conhecimento criem fortes relacionamentos, reduzindo o esforço cognitivo despendido durante o processo de recordação. Portanto, a implementação deste conceito no modelo proposto teve o objetivo de prover ao aluno a capacidade de relacionar e ligar conhecimentos sobre os detalhes de funcionamento do fluxo de execução algorítmica, a fim de construir *Schemas* superiores.

O aspecto de repetição foi implantado por meio da realização do modelo em três algoritmos distintos, e aplicados aos alunos por meio do experimento. O objetivo foi fazer com que o aluno repetisse a execução do jogo, realizando a execução algorítmica visual recursiva em mais de um algoritmo. A Figura ilustra o processo de execução proposto para impor o aprendizado procedural.

Figura 35 – Fluxo de repetição imposto pelo modelo



Fonte: Autor desta dissertação (2016).

Entende-se que a possibilidade de repetição do processo com três algoritmos distintos permite que o aluno possa construir melhores modelos mentais acerca do conceito de recursividade. Na próxima seção serão discutidos os aspectos de como estes algoritmos foram implementados a partir do modelo.

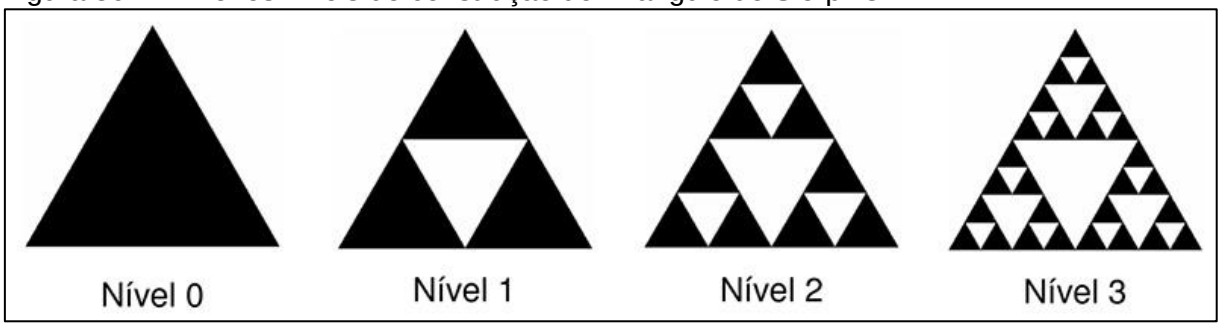
4.5 IMPLEMENTAÇÃO DO MODELO

Nesta seção estão contidas as descrições detalhadas dos aspectos de implementação dos três algoritmos utilizados para validar o modelo proposto por este trabalho. Os algoritmos do *triângulo de Sirpinski*, da *Torre de Hanoi* e do *cálculo fatorial*, foram implementados com o objetivo de tornar possível a avaliação e validação do modelo por meio de uma pesquisa experimental (Seção 0) realizado com alunos iniciantes.

4.5.1 Algoritmo do Triângulo de Sierpinski

Segundo Zavala (2007), o triângulo de Sierpinski, é uma figura geométrica fractal, estudada pelo matemático polonês *Waclav Sierpinski*, cuja construção é realizada por um processo iterativo de divisão de triângulos. A Figura ilustra algumas etapas da construção desse triângulo.

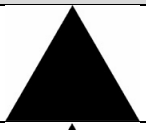
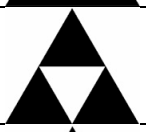
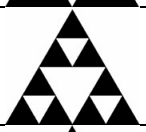
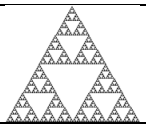
Figura 36 - Primeiros níveis de construção do Triângulo de Sierpinski



Fonte: Zavala (2007).

O Quadro apresenta a notação matemática para a construção do triângulo de Sierpinski.

Quadro 1 - Construção do Triângulo de Sierpinski

Nível x		Nº de triângulos *f(x)*	Comprimento de cada lado *g(x)*
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		$27 = 3^3$	$l \cdot \frac{1}{8} = l \cdot \left(\frac{1}{2}\right)^3$
...			
x		3^x	$l \cdot \left(\frac{1}{2}\right)^x$

Fonte: Zavala (2007).

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++*** (Figura). Neste contexto, o algoritmo apresentou-se bastante simples, possuindo, em cada etapa de sua execução, os seus objetivos bem definidos.

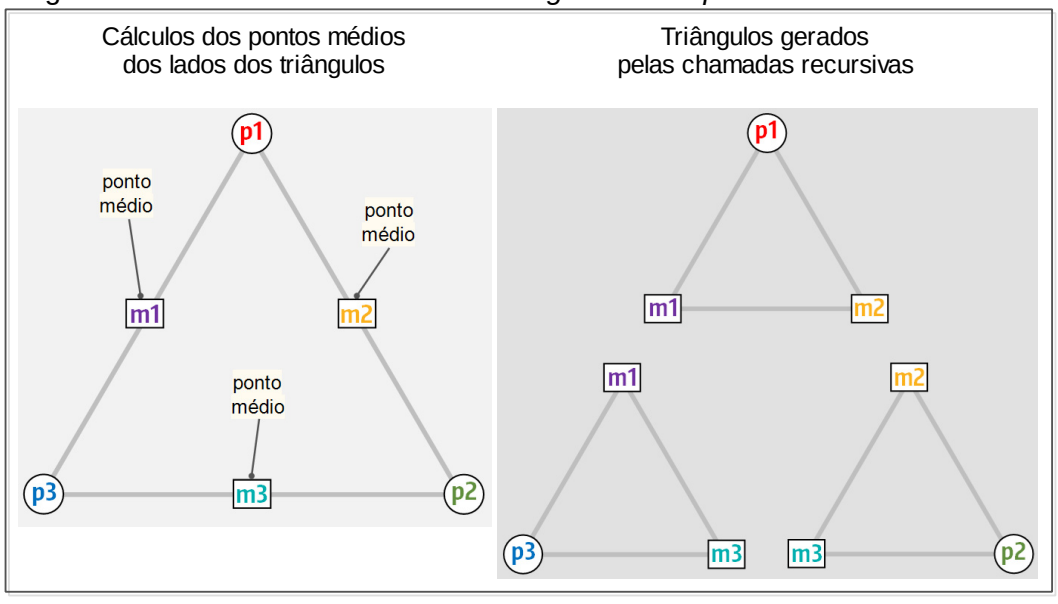
Figura 37 – Implementação recursiva do triângulo de Sierpinski em *C++*

```
1.  void sierpinski(int n, int x1, int y1, int x2, int y2, int x3, int y3)
2.  {
3.      if (n == 0)
4.      {
5.          drawLine(x1, y1, x2, y2);
6.          drawLine(x2, y2, x3, y3);
7.          drawLine(x3, y3, x1, y1);
8.      } else {
9.          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.
16.         sierpinski(n - 1, x1, y1, mx1, my1, mx3, my3);
17.         sierpinski(n - 1, mx1, my1, x2, y2, mx2, my2);
18.         sierpinski(n - 1, mx3, my3, mx2, my2, x3, y3);
19.     }
20. }
```

Fonte: Adaptado de Vandevenne (2007).

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

Figura 38 - *Subdivisão recursiva do triângulo de Sierpinski*

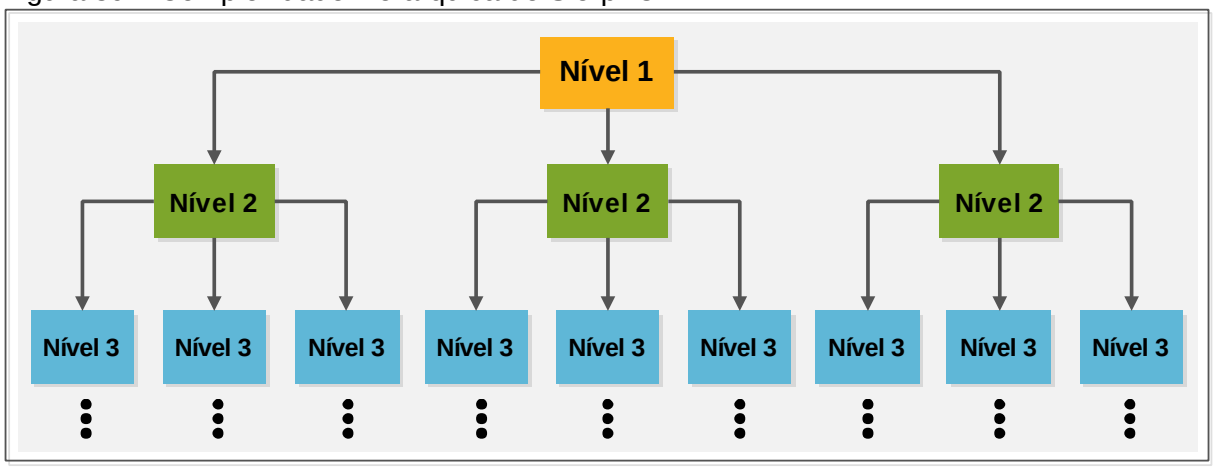


Fonte: Autor desta dissertação (2016).

A partir desta análise verificou-se a complexidade oriunda 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.

Este fato é melhor explicado na Figura , onde é possível observar que as chamadas simultâneas impõem a criação de uma estrutura hierárquica em forma de árvore. A cada etapa recursiva são geradas três novas etapas, fazendo com que a complexidade do problema cresça de forma exponencial. Este crescimento segue a fórmula 3^x, conforme descrito no Quadro .

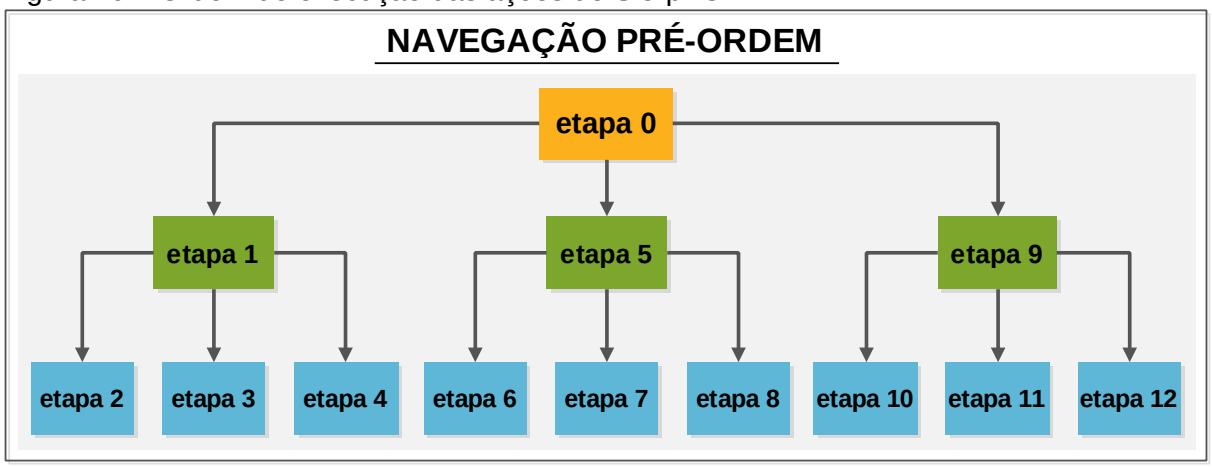
Figura 39 – Complexidade hierárquica de Sierpinski



Fonte: Autor desta dissertação (2016).

Considerando as ações deste algoritmo como sendo as operações de cálculo das médias dos lados dos triângulos (linhas 9 a 14 da Figura) e o desenho dos triângulos (linhas 5 a 7 da Figura), podemos classificar o algoritmo, segundo a ordem de execução das chamadas recursivas, como “*pré-ordem*”, pois, em cada etapa, essas ações são executadas antes das chamadas recursivas. A Figura ilustra este fato, onde é possível perceber a ordem de execução das ações.

Figura 40 – Ordem de execução das ações de Sierpinski



Fonte: Autor desta dissertação (2016).

Tanto o modelo conceitual dessa estrutura hierárquica, como a precedência existente entre as chamadas simultâneas, são características implícitas no que diz respeito ao fluxo de execução deste algoritmo. Desta forma, constata-se que a obtenção de uma imagem clara da relação de causa e efeito existente entre a quantidade chamadas recursivas no corpo da função e o aumento da complexidade de sua estrutura hierárquica de chamadas, bem como a percepção da existência de uma precedência, são de fundamental importância para o desenvolvimento das habilidades do aluno em compreender o conceito de recursividade.

Conclui-se, portanto, que a complexidade da implementação recursiva do algoritmo do Triângulo de Sierpinski reside na quantidade de chamadas recursivas, a qual impõe o aumento da complexidade de sua estrutura hierárquica de chamadas, e a ordem de precedência entre elas.

Desta forma, buscou-se implementar no modelo um mecanismo metafórico para ilustrar a execução do algoritmo, de forma que o aluno consiga perceber os seguintes aspectos:

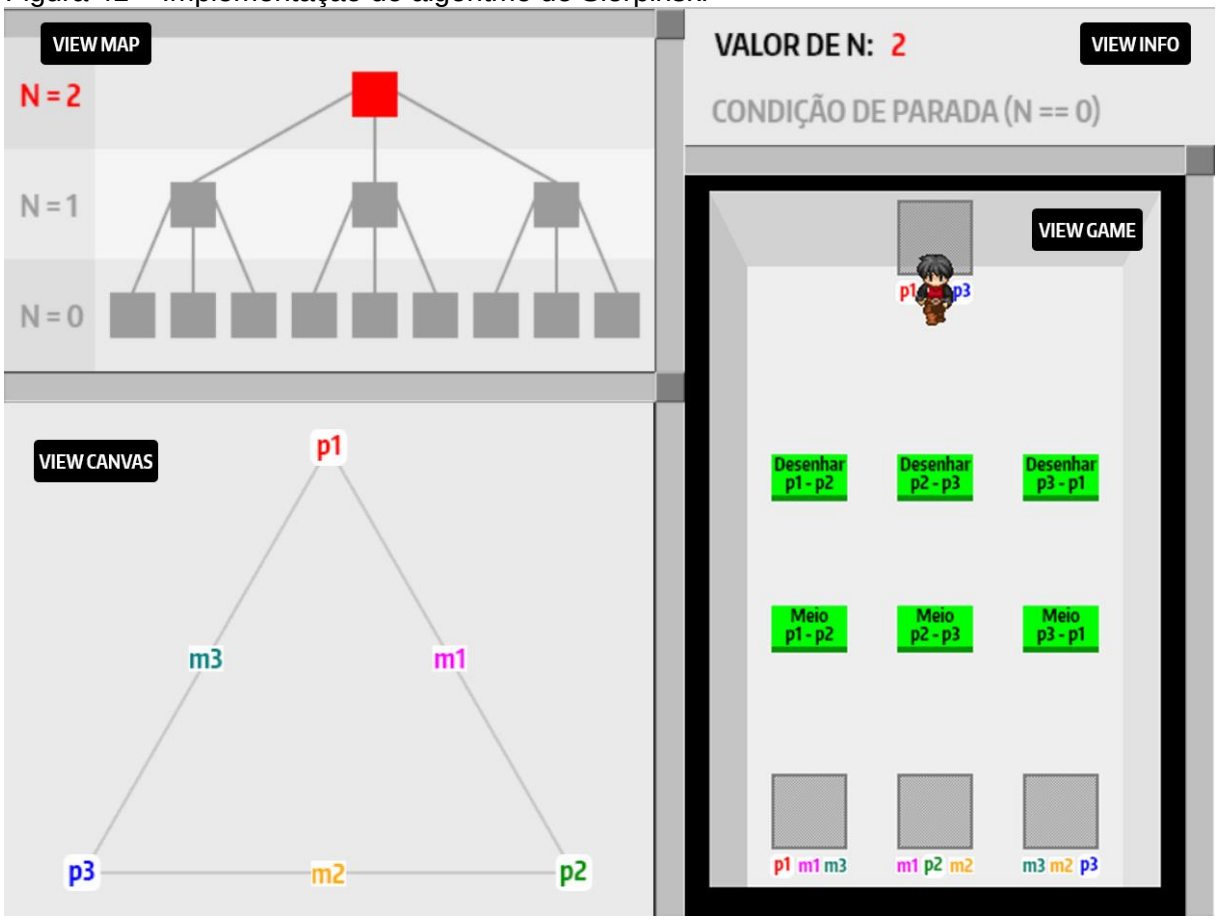
a) A existência de uma crescente estrutura hierárquica, resultante das chamadas recursivas simultâneas.
b) A precedência de execução existente entre as chamadas recursivas simultâneas.
c) A existência de uma condição de parada, e como ela limita a complexidade da estrutura.

Outros aspectos importantes são:

a) O cálculo dos pontos médios de cada lado do triângulo.
b) O mecanismo utilizado para desenhar os triângulos.

A Figura ilustra a proposta de Sierpinski implementada. Conforme descrito no modelo (Seção 0), a perspectiva *Map* apresenta, em tom vermelho, 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 o aspecto da existência de uma condição de parada para o algoritmo. A perspectiva *Game* apresenta a abstração lúdica proposta para o algoritmo. A porta superior abstrai a entrada na função recursiva, e as três portas inferiores abstraem as chamadas recursivas do algoritmo. A perspectiva *Canvas* apresenta o posicionamento dos pontos dos triangulo, em cada etapa iterativa, bem como o processo de desenho dos triângulos com base nas ações do aluno e na sua interação com os botões.

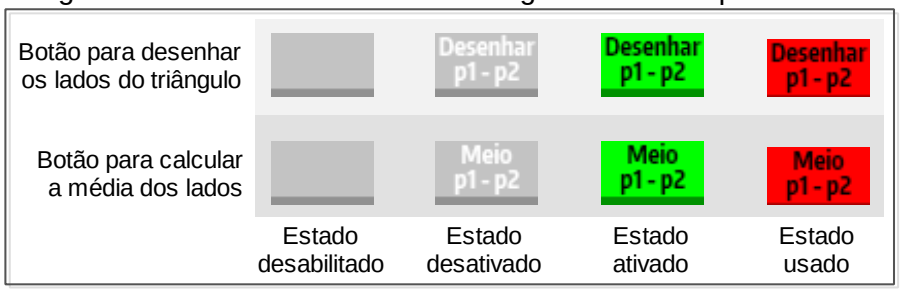
Figura 41 – Implementação do algoritmo de Sierpinski



Fonte: Autor desta dissertação (2016).

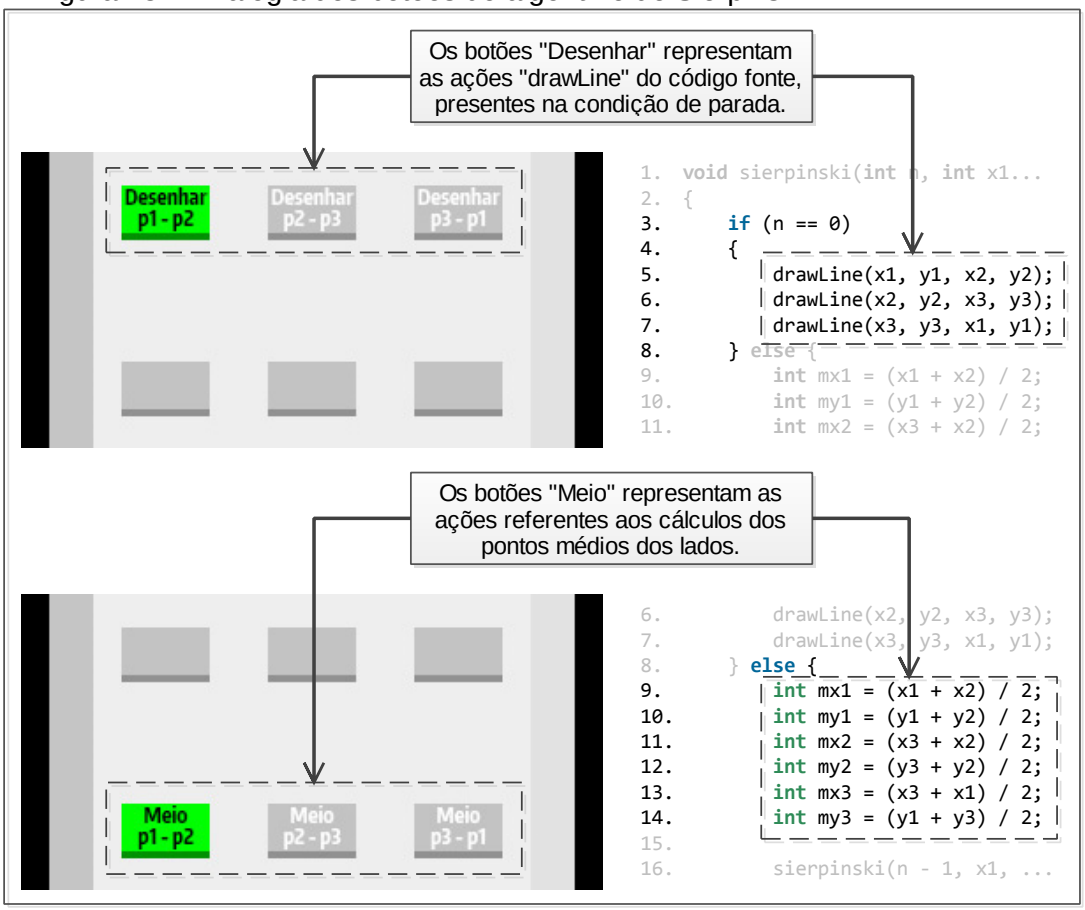
Para que o aluno possa perceber o mecanismo utilizado pelo algoritmo, tanto para calcular os pontos médios dos lados do triângulo como para desenhar os triângulos, foi implementado o uso de botões, os quais devem ser ativados pelo personagem, por meio de comando de teclado, para que os cálculos e os desenhos possam ser realizados. A Figura ilustra os possíveis estados dos botões. A Figura ilustra a relação existente entre as ações do código e suas abstrações lúdicas por meio de botões.

Figura 42 – Estados dos botões do algoritmo de Sierpinski



Fonte: Autor desta dissertação (2016).

Figura 43 – Analogia dos botões do algoritmo de Sierpinski

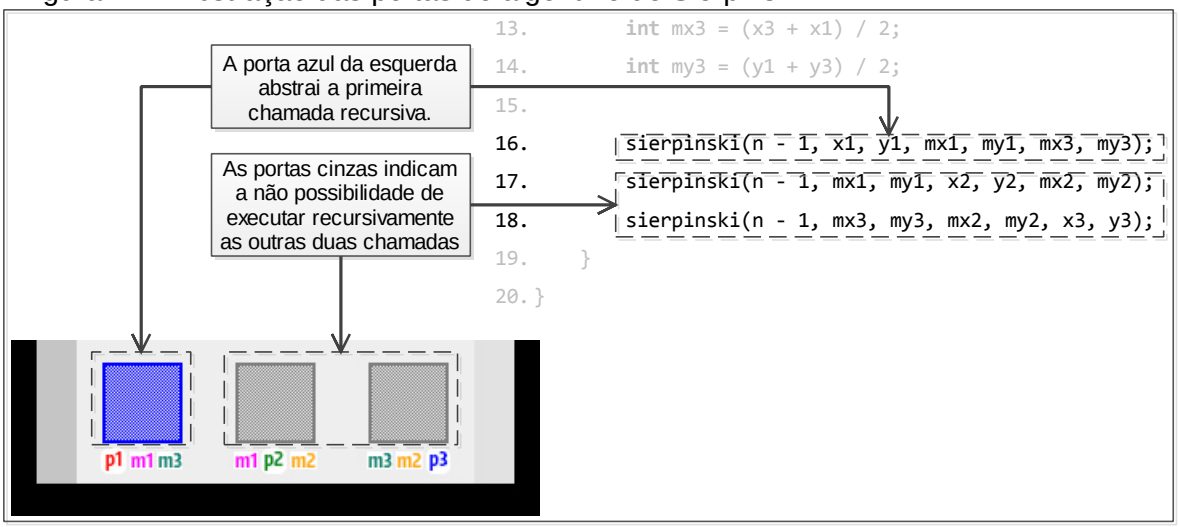


Fonte: Autor desta dissertação (2016).

O aspecto da precedência existente entre as chamadas recursivas é exibido na perspectiva *Game* perspectiva *Game* por meio do trancamento da segunda e terceira portas (

Figura), com o intuito de mostrar para o aluno a obrigatoriedade de se executar a primeira chamada recursiva antes das outras.

Figura 44 – Abstração das portas do algoritmo de Sierpinski

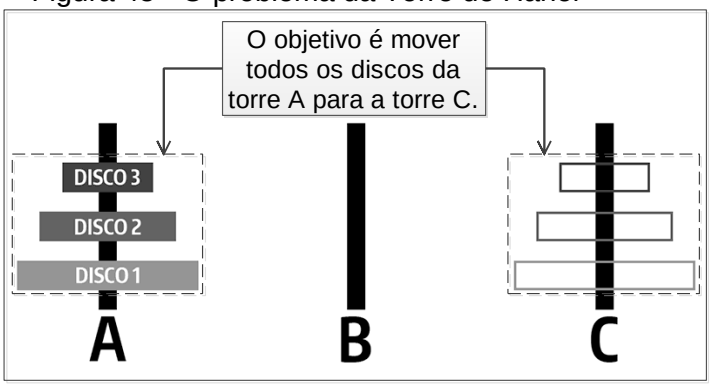


Fonte: Autor desta dissertação (2016).

4.5.2 Algoritmo da Torre de Hanoi

Segundo Stewart e Eliasmith (2011) a Torre de Hanoi é um problema histórico na área da ciência cognitiva. Ele consiste em três torres, nas quais são colocadas um número n de discos. 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. Neste procedimento a torre B funciona como um auxiliar. O objetivo é ilustrado na Figura .

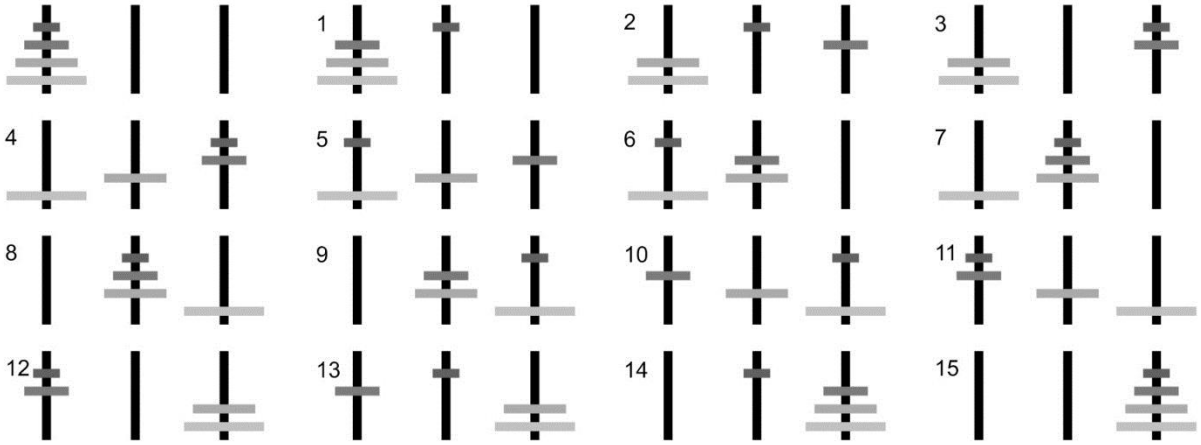
Figura 45 - O problema da Torre de Hanoi



Fonte: Autor desta dissertação (2016).

A Figura ilustra a resolução do problema para quatro discos em 15 etapas. É possível observar que em todas as etapas nenhum disco maior fica sobre um menor em uma mesma torre, e que a torre do meio (B) funciona como um auxiliar para o processo de resolução.

Figura 46 - Resolução da Torre de Hanoi com quatro discos em 15 etapas



Fonte: Stewart e Eliasmith (2011).

O algoritmo da resolução da Torre de Hanoi utilizado como referência para a implementação no modelo utiliza uma impressão no console como forma de analogia ao movimento dos discos (linhas 4 e 7 da Figura). A função (linha 1 da Figura) recebe quatro parâmetros:

- *n* – indica a quantidade de torres, e consequentemente o tamanho do problema.
- *origem* – indica a torre da qual o disco será retirado (A, B ou C).
- *neutro* – indica a torre que será ignorada (A, B ou C).
- *destino* – indica a torre que receberá o disco movido (A, B ou C).

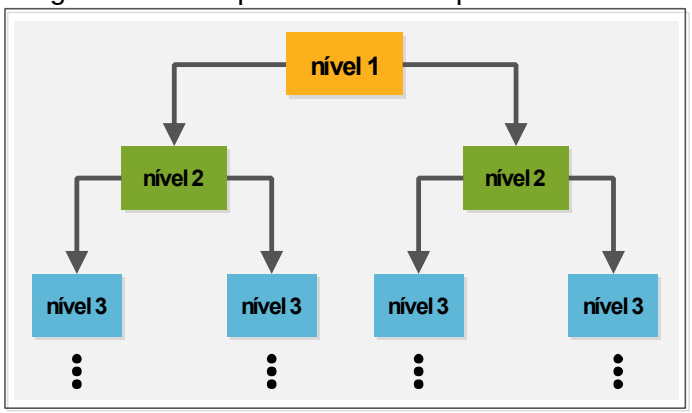
Figura 47 - Implementação recursiva da Torre de Hanoi na linguagem Java

```
1.  void Hanoi(int n, char origem, char neutro, char destino){
2.          if(n == 1)
3.          {
4.              System.out.println("mova disco "+n+" de "+origem+" para "+destino);
5.          } else {
6.              Hanoi(n-1, origem, destino, neutro);
7.              System.out.println("mova disco "+n+" de "+origem+" para "+destino);
8.              Hanoi(n-1, neutro, origem, destino);
9.          }
10.     }
```

Fonte: Adaptado de Burch (1999).

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 (Figura).

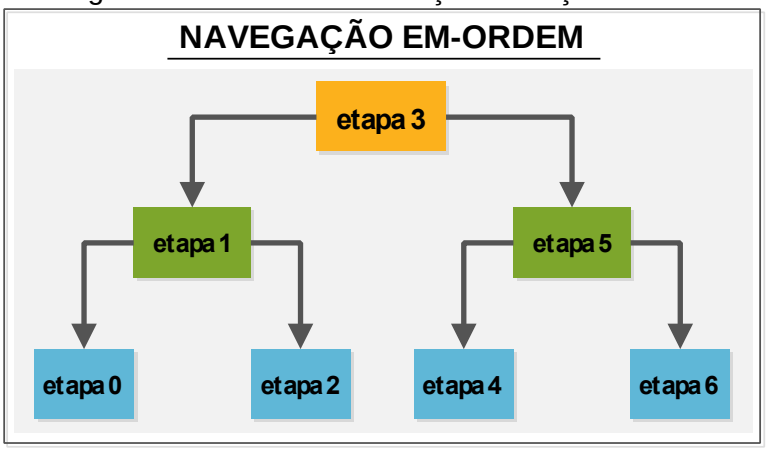
Figura 48 - Complexidade hierárquica de Hanoi



Fonte: Autor desta dissertação (2016).

Um outro aspecto deste algoritmo, e que também o difere da implementação de Sierpinski, diz respeito à ordem de execução das suas ações (movimento dos discos). O fato da ação de mover disco (linha 7 da Figura) estar entre duas chamadas recursivas (linhas 6 e 8 da Figura) implica em uma ordem de navegação na árvore conhecida como navegação “em-ordem”. Este fato é ilustrado na Figura .

Figura 49 - Ordem de execução das ações de Hanoi

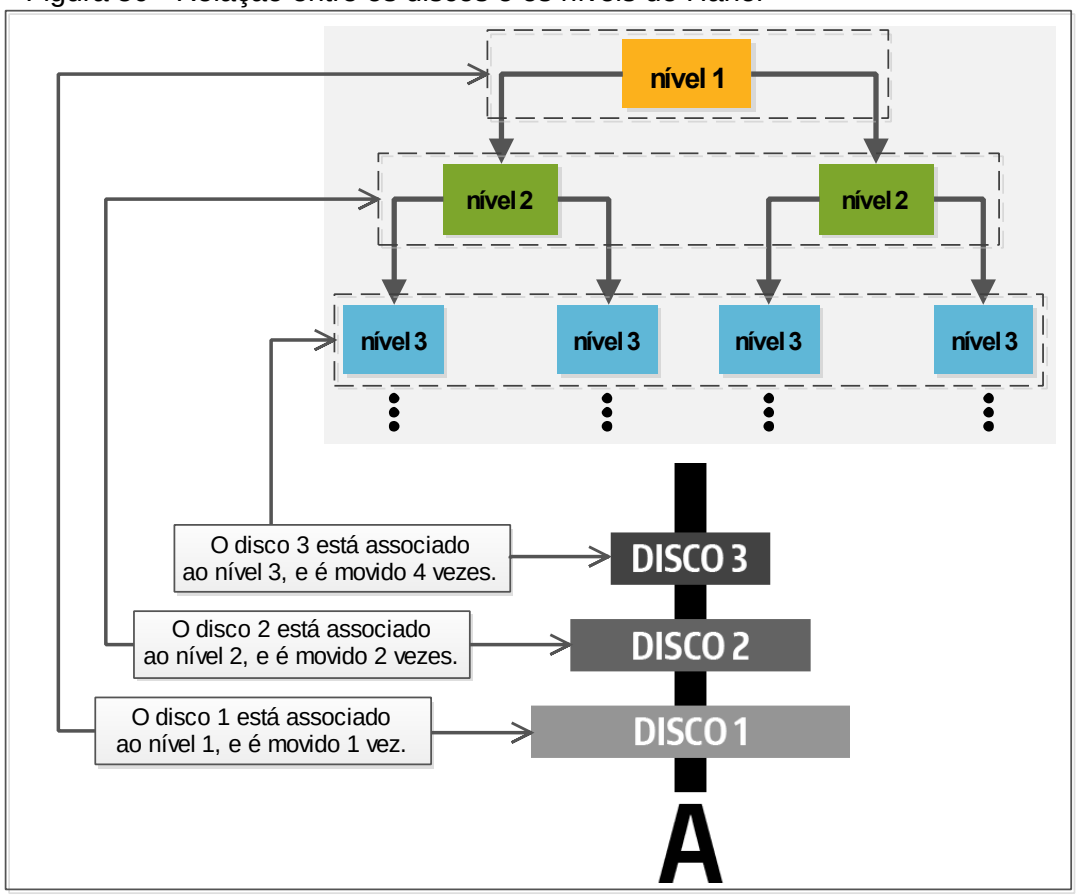


Fonte: Autor desta dissertação (2016).

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 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 50 - Relação entre os discos e os níveis de Hanoi

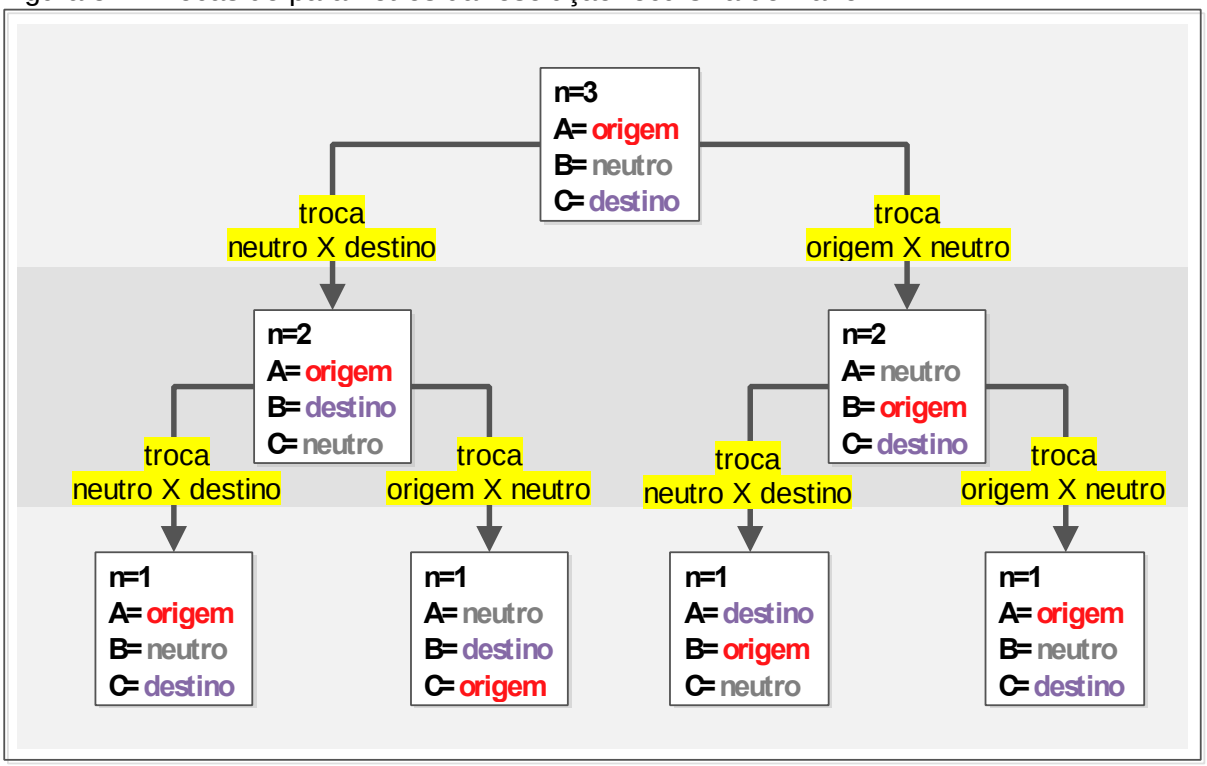


Fonte: Autor desta dissertação (2016).

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 6 e 8 da Figura), 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 mostra os estados de cada etapa da resolução do problema de Hanoi, considerando o tamanho do problema (n) igual ao valor ‘3’.

Figura 51 - Trocas de parâmetros da resolução recursiva de Hanoi



Fonte: Autor desta dissertação (2016).

Portanto, conclui-se que a maior complexidade inerente ao algoritmo recursivo para a resolução da Torre de Hanoi reside na troca dos parâmetros realizada nas chamadas recursivas.

Desta forma, buscou-se implementar no modelo, além dos mecanismos já implementados na resolução de Sierpinski, um mecanismo metafórico para ilustrar os seguintes aspectos:

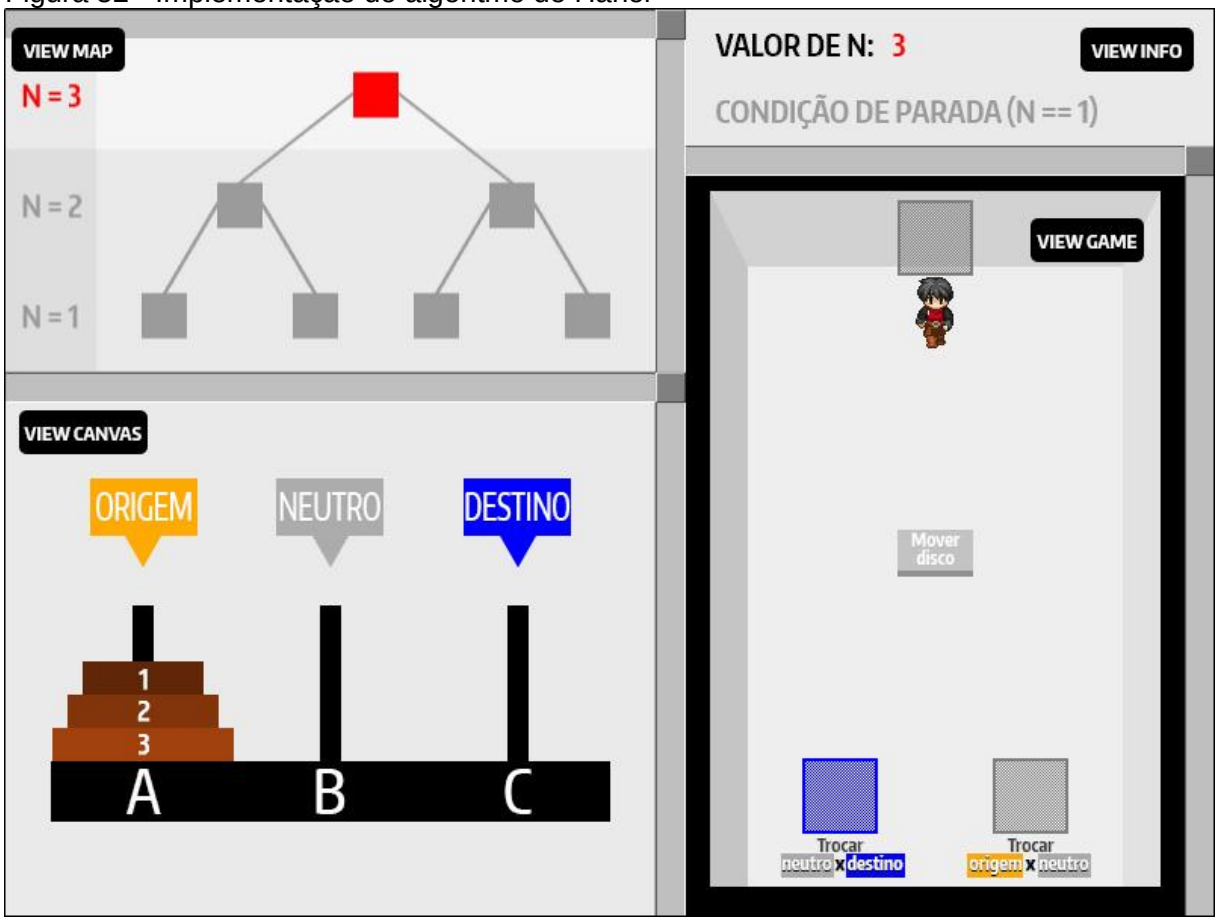
a) A existência de uma troca de parâmetros durante as chamadas recursivas.
b) A imposição de uma navegação hierárquica “*em-ordem*”.

Outro aspecto importante é:

a) A existência de uma relação entre os discos e os níveis da estrutura hierárquica.

Para a implementação da resolução da Torre de Hanoi foram considerados três discos, como sendo o tamanho do problema. A Figura ilustra a proposta implementada no modelo para este problema.

Figura 52 - Implementação do algoritmo de Hanoi

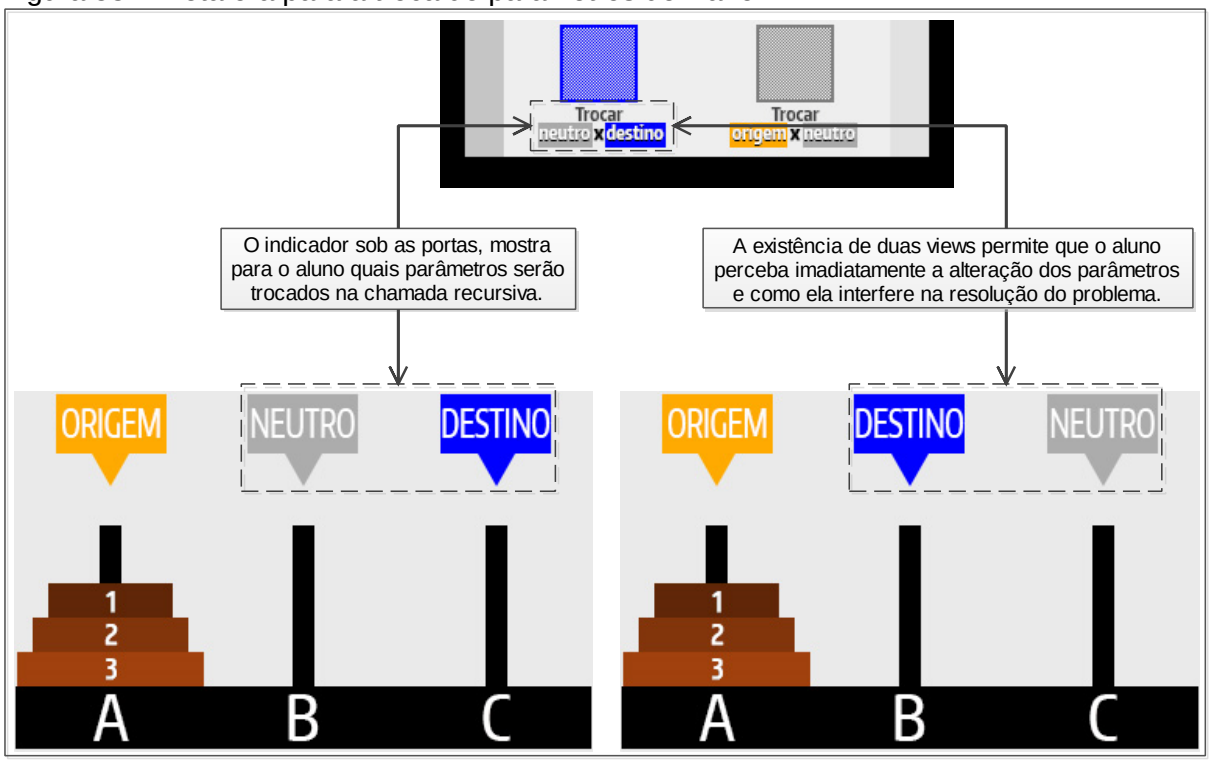


Fonte: Autor desta dissertação (2016).

Seguindo a mesma lógica de Sierpinski, a Perspectiva *Map* apresenta, em tom vermelho, 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. 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 6 e 8 da Figura), e um botão, representando a ação mover disco (linhas 4 e 7 da Figura).

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). 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. 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*.

Figura 53 – Metáfora para a troca de parâmetros de Hanoi

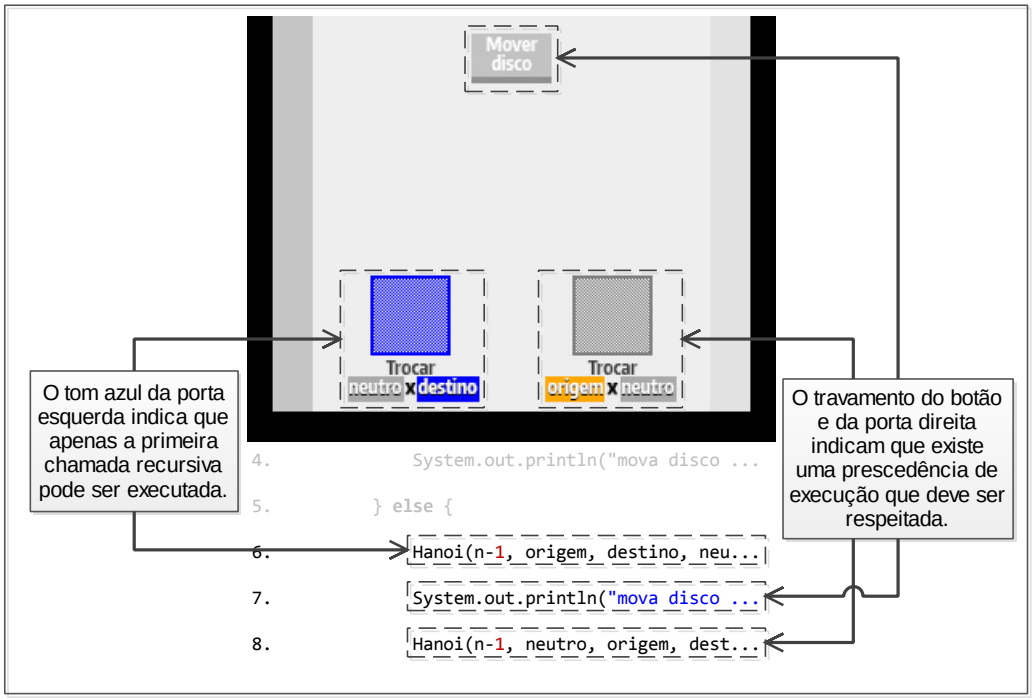


Fonte: Autor desta dissertação (2016).

Para ilustrar a navegação “*em-ordem*” imposta pelo algoritmo recursivo de Hanoi foi criada uma metáfora de travamento das portas e ações de movimentação dos discos. A discos. A

Figura ilustra esse fato, e mostra como a metáfora se relaciona com a regra imposta pelo código de programação para a execução “*em-ordem*”.

Figura 54 – Metáfora para navegação “*em-ordem*” de Hanoi

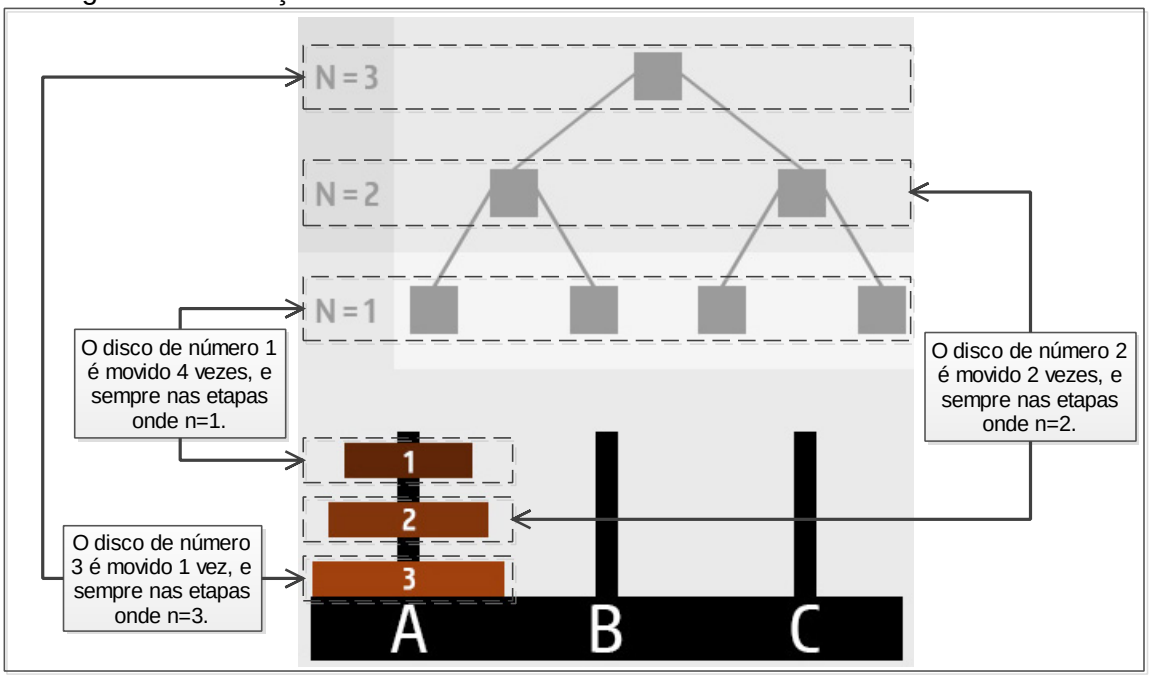


Fonte: Autor desta dissertação (2016).

Para ilustrar 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 respectivos níveis. A

Figura explica essa metáfora, onde é possível perceber como esta relação é ilustrada para o aluno.

Figura 55 – Relação entre os discos e os níveis de Hanoi

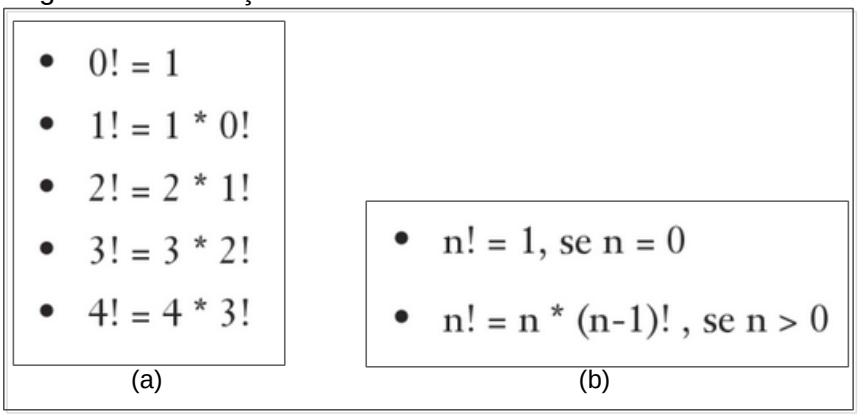


Fonte: Autor desta dissertação (2016).

4.5.3 Algoritmo do Cálculo Fatorial

Segundo Laureano (2008, p.61), o fatorial de um número N é o produto de todos os números inteiros entre 1 e N. A Figura a ilustra um exemplo dessa definição. A Figura b ilustra a representação matemática da resolução de um número fatorial.

Figura 56 - Definição de cálculo fatorial



Fonte: Laureano (2008).

Diferente da implementação de Sierpinski e de Hanoi, o algoritmo fatorial recursivo (Figura) 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.

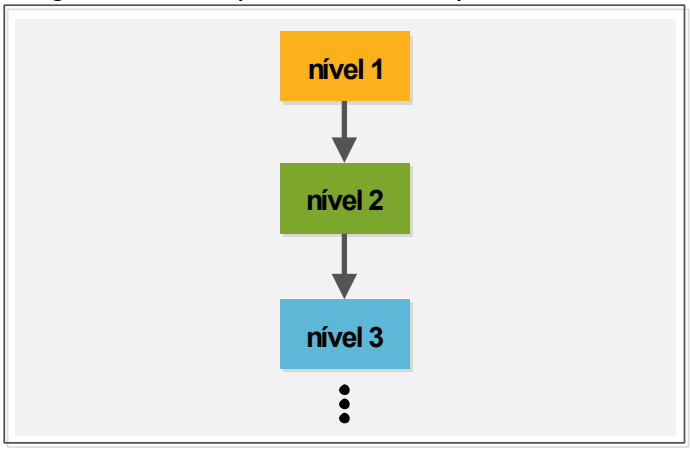
Figura 57 - Implementação recursiva do cálculo fatorial em *C++*

```
1.  int fatorial(int n)
2.  {
3.      if (n == 0)
4.          return 1;
5.      else
6.          return n * fatorial(n - 1);
7.  }
```

Fonte: Adaptado de Laureano (2008).

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. A Figura ilustra a simplicidade da estrutura hierárquica deste algoritmo recursivo.

Figura 58 - Complexidade hierárquica de Hanoi



Fonte: Autor desta dissertação (2016).

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 da Figura . Neste contexto, a chamada recursiva realiza o papel de um segundo operando da operação matemática. O Quadro 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.

Quadro 2 - Chamada recursiva como um operando algébrico

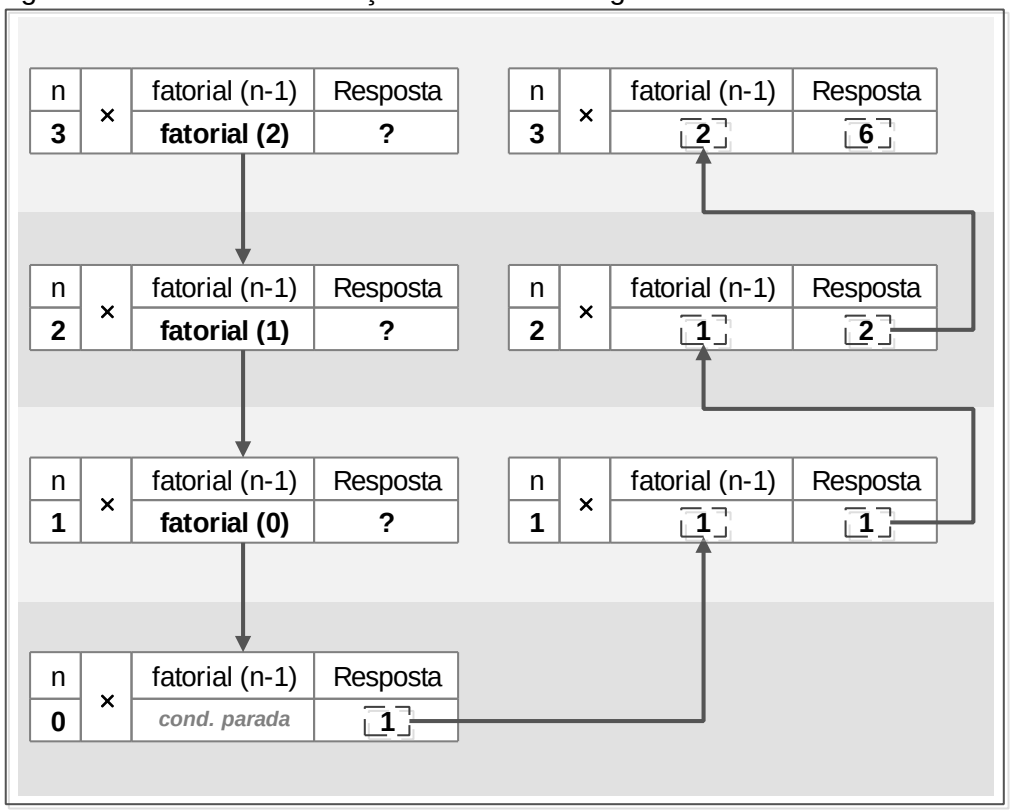
	operando 1	operador	operando 2	**resultado**
Exemplo de operação algébrica	*2*	×	*2*	*4*
Expressão do algoritmo recursivo	***n***	**×**	***Fatorial(n-1)***	***?***

Fonte: Autor desta dissertação (2016).

Este fato impede que o aluno consiga identificar o valor da expressão. Expressão esta que somente poderá ser calculada após todas as chamadas recursivas serem realizadas. A Figura 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 ser alcançada, e consequentemente a sua condição de parada, para que a partir de então o valor comece a ser retornado para as etapas anteriores, e o resultado possa ser encontrado.

Figura 59 - Ordem de execução recursiva do algoritmo fatorial



Fonte: Autor desta dissertação (2016).

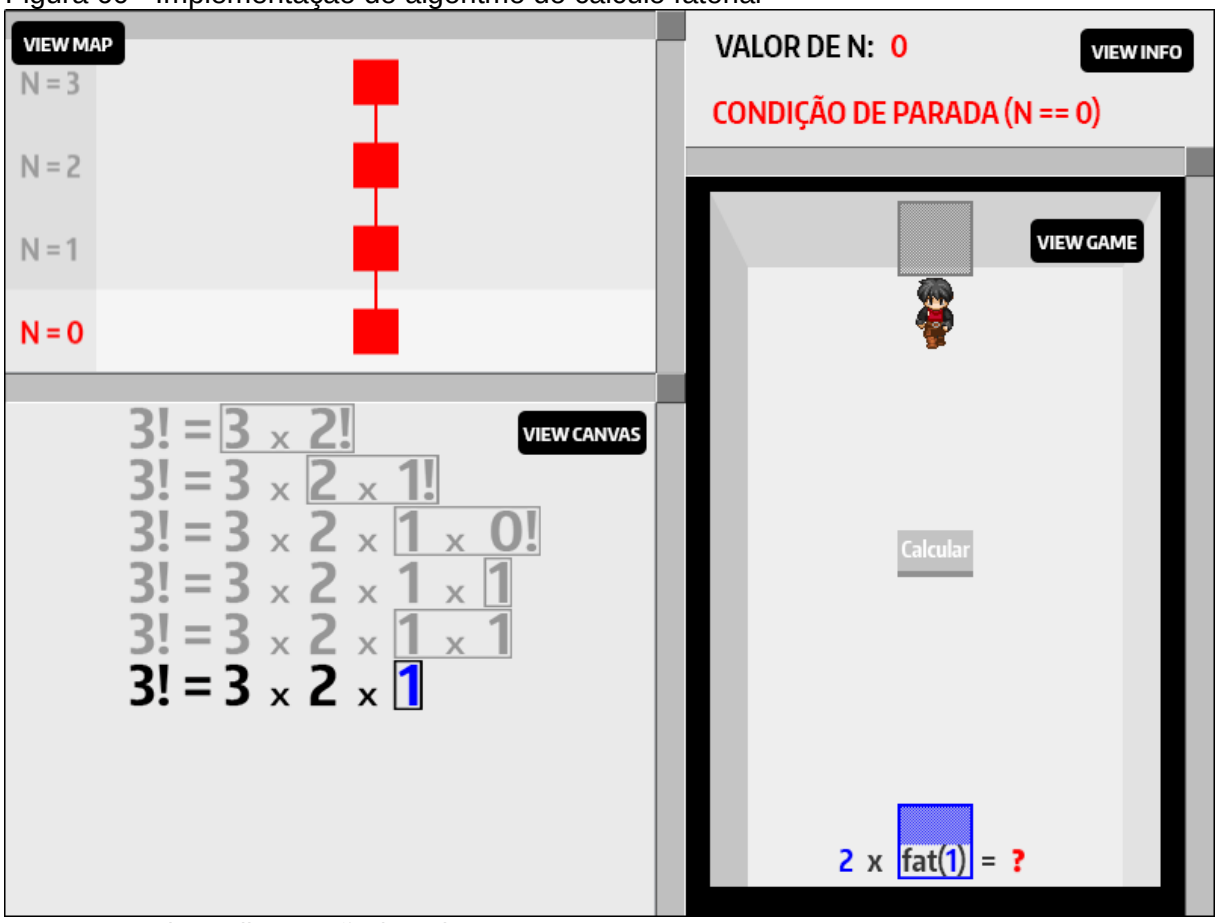
Conclui-se, portanto, que a complexidade da implementação recursiva da resolução fatorial reside na incapacidade de se reconhecer o resultado da expressão até que a última chamada recursiva seja alcançada, juntamente com a satisfação de sua condição de parada.

Desta forma, buscou-se implementar no modelo, além dos aspectos já abordados nas implementações anteriores, mecanismos metafóricos para ilustrar os seguintes aspectos acerca deste algoritmo:

a) A impossibilidade de se encontrar o resultado das operações algébricas durante a execução das etapas, até que seja alcançada a condição de parada e os resultados possam ser sequencialmente retornados.
b) O mecanismo utilizado para retornar os valores para as etapas anteriores.

A Figura ilustra como foi implementado no modelo a representação da resolução algorítmica fatorial recursiva.

Figura 60 - Implementação do algoritmo do cálculo fatorial



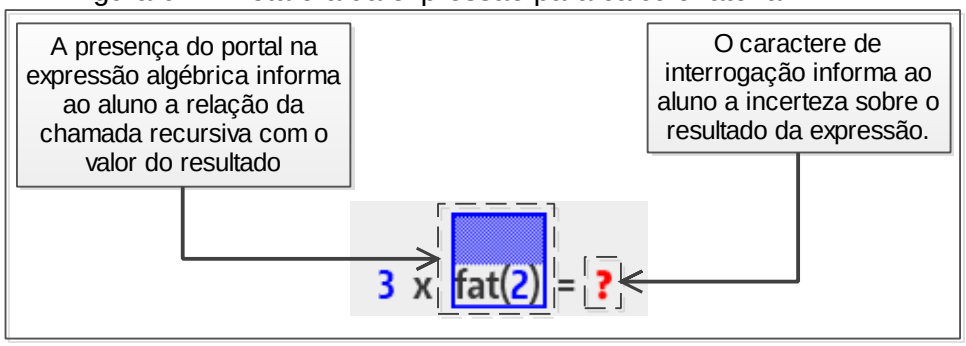
Fonte: Autor desta dissertação (2016).

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. De maneira análoga às implementações anteriores, a Perspectiva *Map* apresenta a estrutura hierárquica inerente a execução do algoritmo e 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, onde o segundo operador é representado pelo portal que direciona o personagem paras as etapas seguintes do processo de resolução, fazendo alusão à linha 6 da Figura .

O caractere ‘?’ ao final da expressão representa a incerteza sobre o resultado da expressão para aquele nível recursivo. A Figura ilustra esta metáfora.

Figura 61 – Metáfora da expressão para cálculo fatorial



Fonte: Autor desta dissertação (2016).

Uma vez alcançada a condição de parada, dá-se início ao processo de retorno de valores, conforme ilustrado na Figura . Entende-se que para ilustrar este aspecto, faz-se necessário mostrar 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 a resolução do problema.

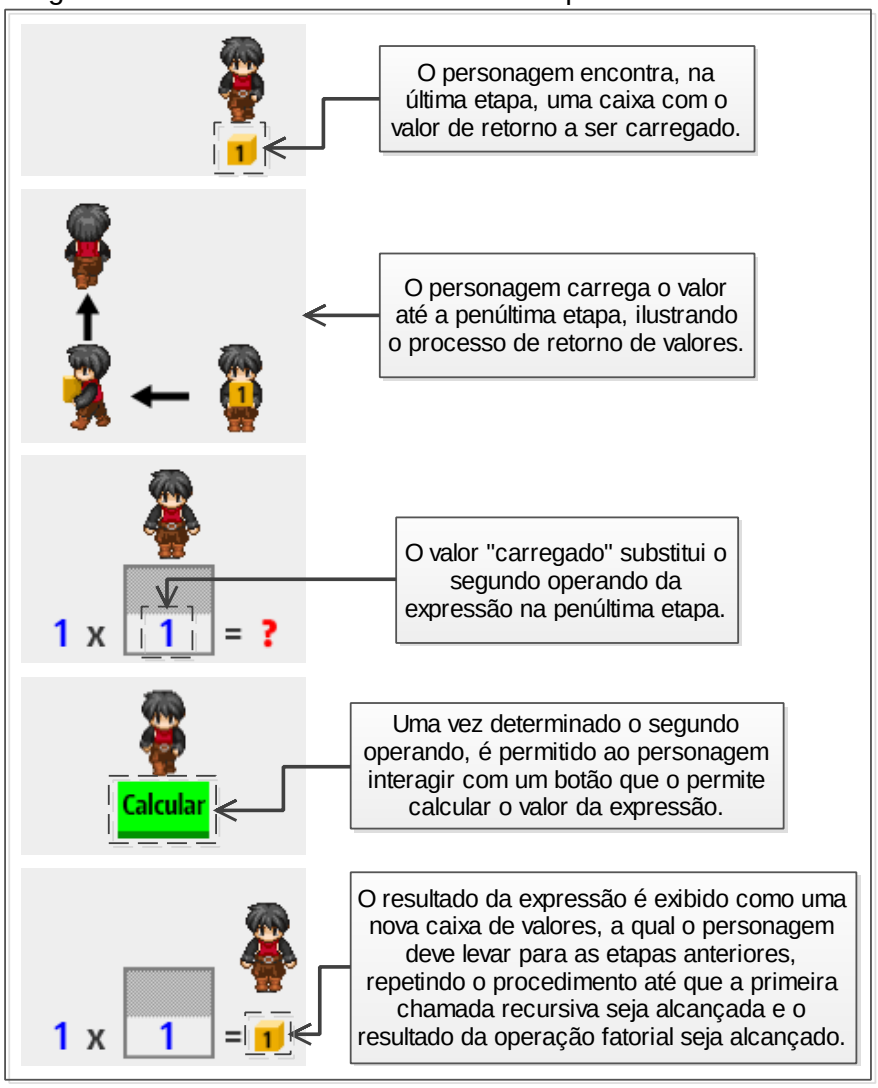
A

Figura 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.

A Figura ilustra os estados desta metáfora durante as etapas da resolução fatorial recursiva para o valor ‘3’. É possível observar que, após alcançada a condição de parada, uma caixa é retornada. Ela substitui o portal na etapa anterior, e o símbolo de interrogação é substituído por uma nova caixa, a qual deve ser novamente carregada, repetindo o processo até que seja alcançada a primeira chamada recursiva.

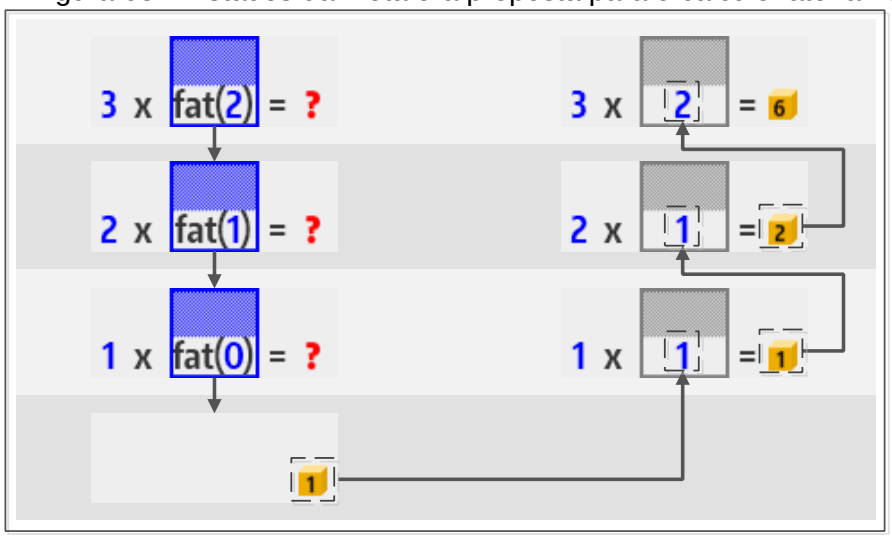
O objetivo é fazer com que o aluno enxergue de forma clara o mecanismo utilizado pela implementação recursiva para calcular o fatorial de um número, criando assim os corretos modelos mentais sobre o fluxo de resolução para este tipo de algoritmo.

Figura 62 – Metáfora da caixa de valores para o cálculo do fatorial



Fonte: Autor desta dissertação (2016).

Figura 63 – Estados da metáfora proposta para o cálculo fatorial recursivo

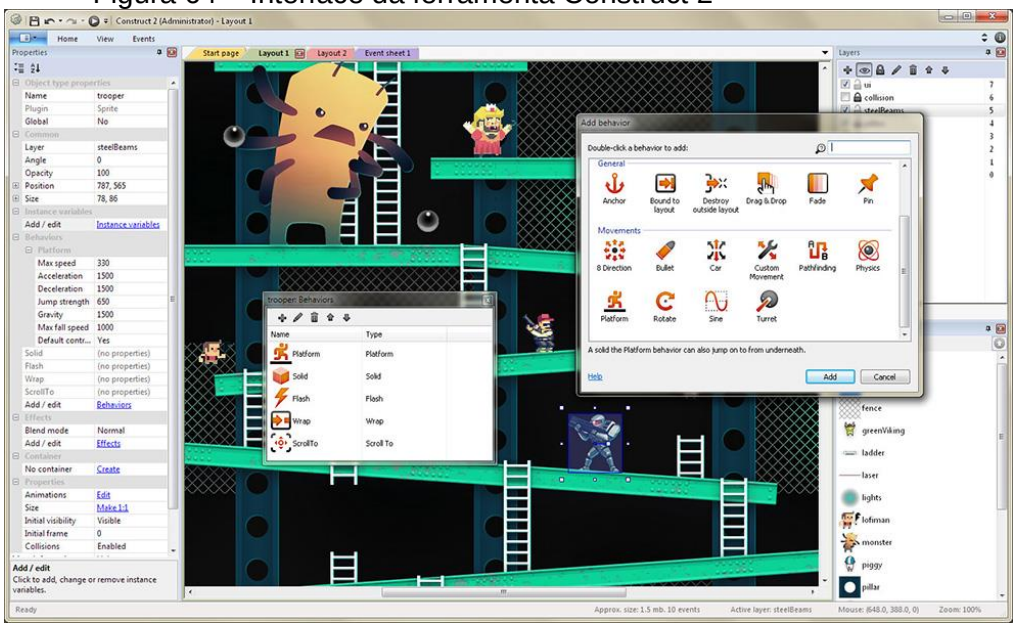


Fonte: Autor desta dissertação (2016).

4.6 PROCESSO DE DESENVOLVIMENTO

O modelo proposto foi transformado em três ferramentas por meio do software de desenvolvimento denominado Construct 2 (Figura), uma ferramenta proprietária que fornece uma versão *comunity*, com algumas limitações para o seu uso. Apesar de ser uma ferramenta comercia, as funcionalidades fornecidas por sua versão gratuita foram suficientes para realizar o objetivo desta dissertação.

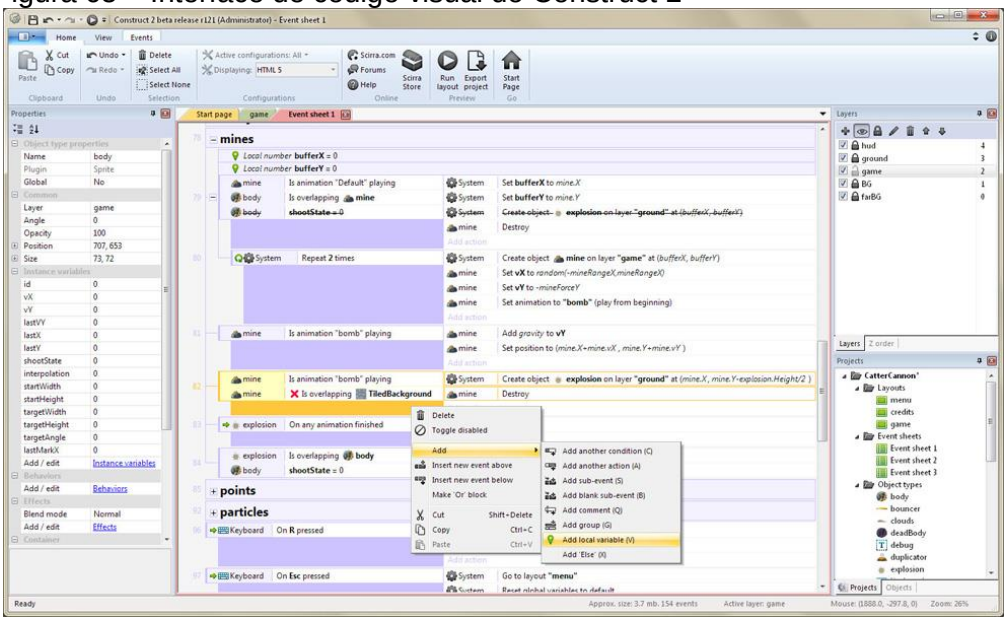
Figura 64 – Interface da ferramenta Construct 2



Fonte: Autor desta dissertação (2016).

Outro aspecto importante do software é a sua facilidade de uso. Ela fornece um ambiente de desenvolvimento de código visual. Por meio de elementos gráficos, é possível criar constructos de linguagem de programação, de maneira ágil e rápida (Figura).

Figura 65 – Interface de código visual do Construct 2



Fonte: Autor desta dissertação (2016).

Após finalizado o processo de desenvolvimento das ferramentas, os softwares foram compilados em formato web (html5 + javascript), o único formato permitido pela versão gratuita do software. As ferramentas foram hospedadas como links públicos nos servidores do *Dropbox*, com o objetivo de torna-las executáveis para a execução do experimento.

4.7 CONCLUSÕES

Neste capítulo foi especificado um modelo para o ensino de conceitos de recursividade, fundamentado na psicologia cognitiva. Foi possível observar a complexidade dos algoritmos recursivos por meio da análise de sua execução. Ficou claro que o entendimento deste tipo de algoritmo está fortemente ligado aos conceitos de navegação em estruturas de árvores hierárquicas. Foi possível observar também que a recursividade impõe um fluxo dinâmico de funcionamento, propício a ser explorado sob uma abordagem lúdica.

O objetivo da ferramenta descrita pelo modelo não é ser um ambiente interativo de programação, tal como o *Scratch* (LIFELONG;KINDERGARTEN, 2016) ou *Alice* (CARNEGIE MELLON UNIVERSITY, 2016). Seu objetivo não é gerar código e nem construir software, mas ser uma ferramenta visual, de apoio ao aprendizado dos processos de programação, de forma que os alunos possam interagir, visualizar e refletir sobre a lógica de funcionamento do conceito de recursividade. Neste sentido, é possível concluir que ele proporciona o aprendizado e desenvolvimento das habilidades e dos modelos mentais dos alunos, necessários para a progressão dos mesmos nos cursos de computação.

5 METODOLOGIA DE PESQUISA

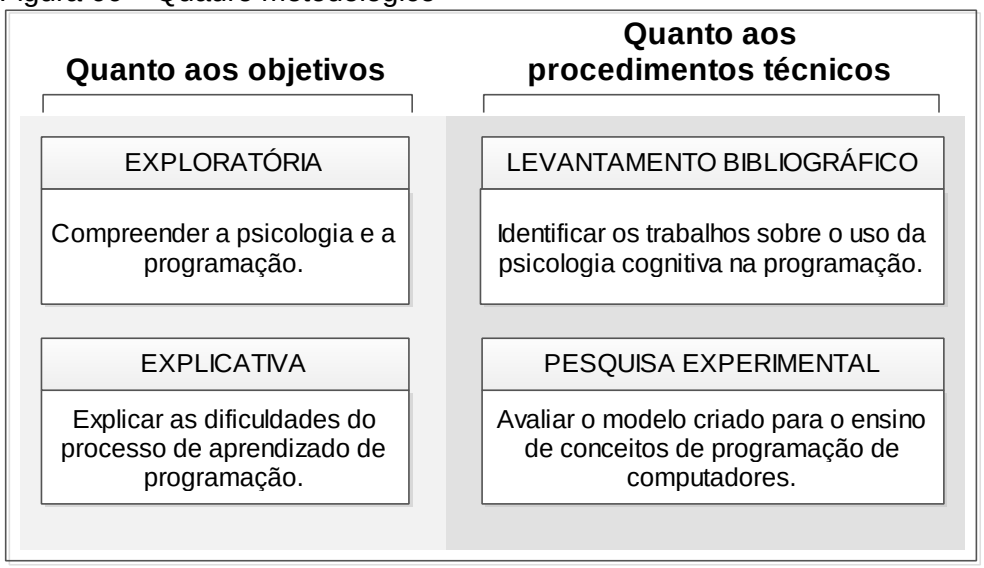
Neste capítulo são discutidos os aspectos metodológicos desta dissertação. Desde o processo de pesquisa bibliográfica até o experimento científico, realizado como forma de validação da metodologia proposta.

A princípio é apresentado o quadro metodológico (Figura), o qual descreve, à luz da bibliografia científica, quais os modelos utilizados, bem como onde e como eles foram utilizados. Por conseguinte, é apresentada, sob uma perspectiva linear e cronológica, as etapas realizadas durante todo o processo de pesquisa deste trabalho (Figura). Adiante são relatados os aspectos referentes à pesquisa experimental, a qual faz uso da metodologia *DECIDE* (SHARP et al., 2007) para descrever seus processos. Por fim são discutidas as variáveis e as ameaças à validade do experimento.

5.1 QUADRO METODOLÓGICO

De acordo com Gil (2000) as pesquisas científicas podem ser classificadas segundo os **objetivos** e segundo os **procedimentos técnicos**. Tendo em vista as definições de pesquisa definidas pelo autor e os objetivos desta dissertação, optou-se por utilizar os tipos de pesquisas descritos no quadro metodológico (Figura).

Figura 66 – Quadro metodológico



Fonte: Autor desta dissertação (2016).

5.1.1 Quanto aos objetivos

Com relação aos objetivos, definimos nossa pesquisa como **exploratória** e **explicativa**. Exploratória pois buscou compreender a psicologia cognitiva e a programação de computadores. Explicativa, pois, buscou explicar, por meio do estudo da psicologia cognitiva e suas teorias, as dificuldades inerentes ao processo de aprendizado da programação.

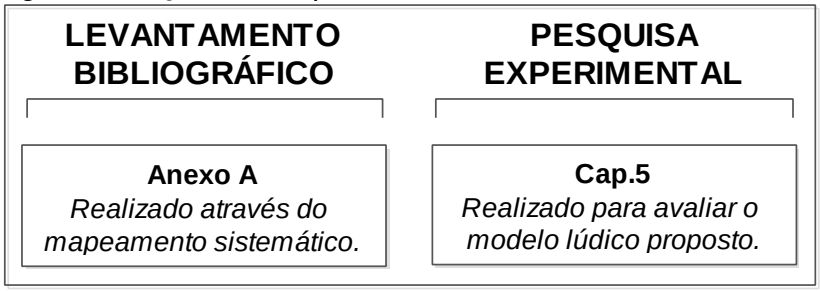
A *pesquisa exploratória* foi utilizada no capítulo 2 com o intuito de compreender a importância do aprendizado da programação de computadores. Isto foi realizado por meio da análise de trabalhos científicos e livros que relatavam sobre o assunto. No capítulo 3, ela foi utilizada para entender e conceituar a psicologia cognitiva e compreender as limitações da mente humana. Neste capítulo, também foram revisados livros e artigos científicos sobre o tema.

A *pesquisa explicativa* foi utilizada no capítulo 3 para explicar as limitações da mente humana durante o processo de aprendizado. Por meio da análise de experimentos científicos foi possível entender e explicar tais limitações.

5.1.2 Quanto aos procedimentos técnicos

Com base nas afirmações de Gil (2000), os procedimentos adotados foram **levantamento bibliográfico** e **pesquisa experimental**. A Figura melhor ilustra o processo.

Figura 67 – Quanto aos procedimentos técnicos



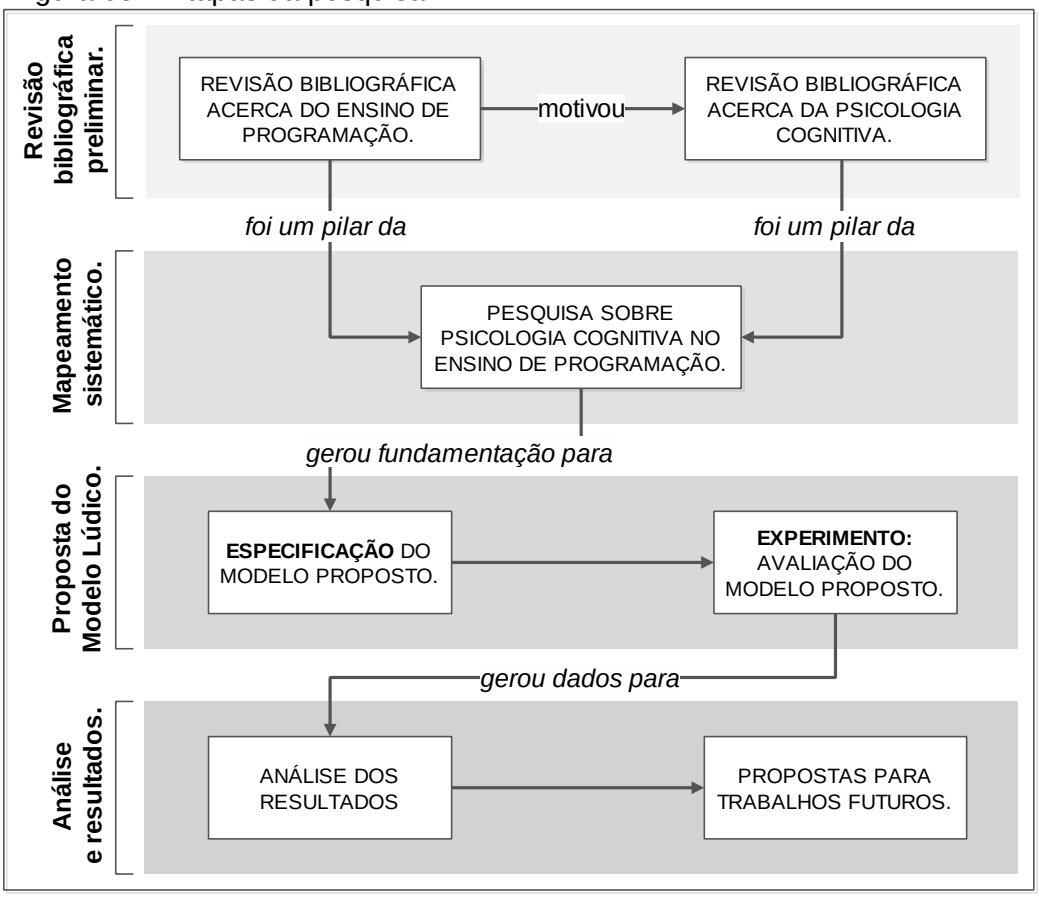
Fonte: Autor desta dissertação (2016).

No Anexo A foi realizado um *levantamento bibliográfico* com o objetivo de identificar e classificar os principais artigos que relatam o uso da psicologia cognitiva no ensino de programação de computadores. Este procedimento foi executado por meio de um mapeamento sistemático e suas recomendações, definidas por Petersen et al. (2015), com o intuito de entender de que forma esses estudos abordavam o tema.

A pesquisa experimental foi realizada com o intuito de avaliar o modelo proposto. Ela foi conduzida com base no método *pretest-posttest* (DIMITROV ; RUMRILL, 2003). A seção 0 descreve em detalhes este procedimento.

5.2 ETAPAS DA PESQUISA

Figura 68 – Etapas da pesquisa



Fonte: Autor desta dissertação (2016).

De forma resumida, o processo de pesquisa consistiu nas etapas ilustradas na Figura . O processo iniciou-se com uma revisão bibliográfica preliminar sobre o ensino de programação, onde foi percebida a associação de algumas iniciativas com

teorias relacionadas à psicologia cognitiva. Esta percepção motivou a pesquisa sobre o referido tema, com o intuito de construir a base conceitual necessária para a formalização de uma pesquisa mais aprofundada. Estes 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 para o ensino de conceitos de programação. A etapa final do trabalho consistiu em avaliar e validar tal modelo por meio da realização de uma pesquisa experimental.

5.3 PLANEJAMENTO DO EXPERIMENTO

A execução da pesquisa experimental teve o intuito de avaliar o modelo lúdico proposto por esta dissertação. 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 do experimento é obter informações sobre a eficácia de uma metodologia lúdica fundamentada na psicologia cognitiva para o ensino de conceitos de programação de computadores, com um interesse especial em analisar o ganho de desempenho dos alunos após a aplicação do experimento.

5.3.2 Explorar perguntas a serem respondidas

Tomando como base o objetivo a ser alcançado, foi elaborado um conjunto de perguntas que direcionam os experimentos à geração e análise dos dados:

a) Os alunos enfrentaram dificuldades para entender o funcionamento da ferramenta?
b) A ferramenta conseguiu construir os modelos mentais acerca do fluxo de execução dos algoritmos recursivos?

c) Ao final do experimento foi possível observar uma melhora geral de desempenho nos testes que pudesse ser atribuída à metodologia proposta?
d) Os alunos aprovaram a metodologia como ferramenta de apoio ao ensino de conceitos de programação de computadores?

5.3.3 Escolher o método de avaliação

A execução da pesquisa fez uso de dois (2) tipos de avaliações:

a) *Avaliação quantitativa*: por meio do método *pretest-posttest* (DIMITROV ; RUMRILL, 2003), foi realizado um experimento científico que teve como objetivo mensurar numericamente a eficácia da metodologia proposta por esta dissertação.
b) *Avaliação qualitativa*: por meio de questionário formatado em função da escala de Likert (1932), foi possível avaliar a opinião dos participantes com relação a determinados aspectos da metodologia proposta.

5.3.3.1 Análise quantitativa (pretest-posttest)

A execução da pesquisa fez uso do método *pretest-posttest*, que permite comparar grupos de participantes em um determinado experimento científico por meio da aplicação de testes. O referido método consiste na aplicação de um teste inicial, o pré-teste, seguido da aplicação do experimento objetivo do estudo e finalizado pela aplicação de um teste final, o pós-teste. Neste trabalho o modelo foi aplicado da seguinte maneira:

a) Aplicação do **pré-teste** – questionário com o intuito de avaliar o conhecimento inicial dos alunos acerca do conceito de recursividade.
b) **Experimento** – explicação do conceito de recursividade por meio da metodologia lúdica, com o intuito de desenvolver nos alunos melhores modelos mentais acerca do referido conceito.
c) Aplicação de um **pós-teste** – reaplicação de questionário, com o intuito de observar a retenção de conhecimento fornecida pelo experimento, observando-se o contraste com as notas do pré-teste.

Segundo Dimitrov e Rumrill (2003, p.159), o método *pretest-posttest* é amplamente utilizado em pesquisas comportamentais, pois, através dele, pode-se mensurar diferenças entre resultados em tratamentos experimentais. Eagle et al. (2008) utilizaram esse método para realizar as suas avaliações quantitativas.

5.3.3.2 Análise qualitativa (escala de Likert)

Segundo Kronbauer (2013), a escala de Likert (1932) consiste em atribuir valores numéricos às questões com o intuito de avaliar o grau de concordância ou discordância dos entrevistados com relação aos aspectos que estão sendo medidos.

O Anexo D desta dissertação contém o questionário qualitativo, baseado na escala de Likert, usado para avaliar a metodologia lúdica proposta por este trabalho.

5.3.4 Identificar e administrar as questões práticas

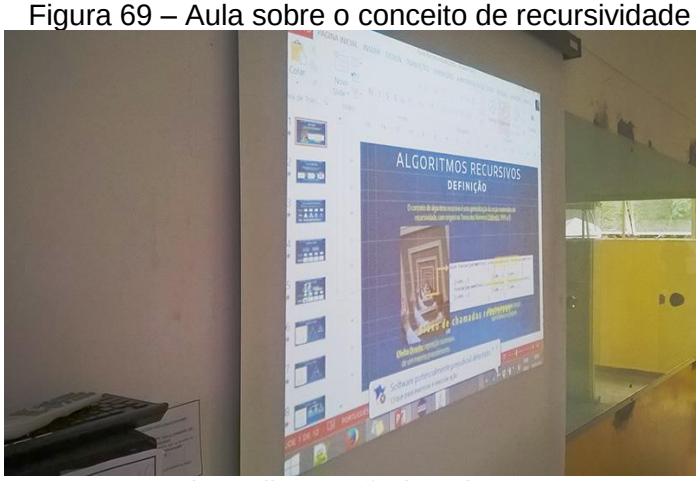
Nesta fase, foram levantados inúmeros pré-requisitos, dentre os quais podem ser destacados: (i) a distribuição dos participantes do experimento; (ii) os materiais necessários para a realização do experimento; e (iii) a lista de exercícios a ser aplicada no pré-teste e pós-teste.

A pesquisa experimental foi realizada com 38 alunos, selecionados de forma aleatória, dos cursos de Engenharia da Computação, Elétrica e Mecatrônica da Universidade Salvador. Os alunos foram divididos em três turmas, não respectivas, nos períodos da manhã, tarde e noite, todas referentes ao próprio dia letivo para a matéria de Estrutura de Dados, a qual é comum aos três cursos.

O experimento foi realizado no dia 20 de abril de 2016 no laboratório de informática do *Pavilhão de Aulas 7* (PA7), o qual dispunha de um total de cerca de 28 computadores. A cada computador foi atribuído apenas um participante, com o intuito de evitar possíveis ameaças internas, as quais serão discutidas na seção 0.

O início do experimento consistiu em uma aula expositiva sobre o conceito de recursividade (Figura), 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.

Figura 69 – Aula sobre o conceito de recursividade

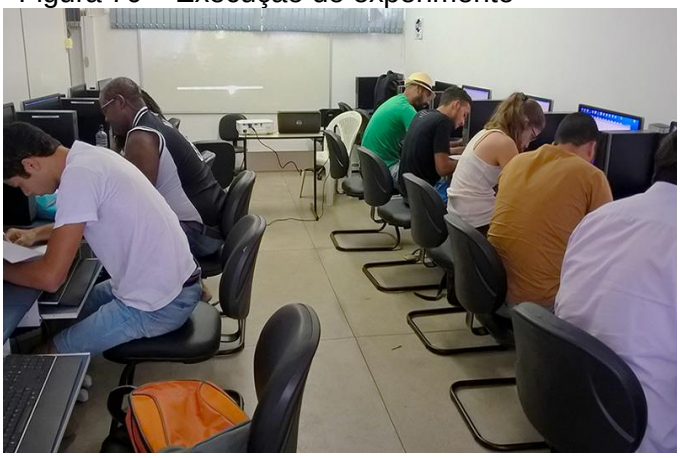


Fonte: Autor desta dissertação (2016).

Após a aula, os alunos foram submetidos ao pré-teste, com o intuito de avaliar seus conhecimentos iniciais acerca do conceito de recursividade. O pré-teste consistiu em uma lista de exercícios com questões de múltipla escolha (Anexo C), no qual os alunos foram desafiados a resolver os três algoritmos propostos (Seção 0), de forma textual.

Após a aplicação do pré-teste os alunos foram expostos à execução do experimento (Figura), o qual consistiu na execução dos mesmos três algoritmos, porém por meio da metodologia lúdica proposta. A execução foi mediada pelo autor deste trabalho, com o objetivo de garantir a correta aplicação da metodologia.

Figura 70 – Execução do experimento



Fonte: Autor desta dissertação (2016).

Após a execução do experimento os alunos foram novamente expostos aos questionários quantitativos (pós-teste), com o intuito de avaliar a retenção de conhecimentos provida pela metodologia lúdica proposta. É importante salientar que tanto o pré-teste como o pós-teste consistiam na resolução dos mesmos três algoritmos expostos no experimento, porém em forma de código fonte.

Com o objetivo de avaliar a opinião dos participantes sobre diversos aspectos referentes ao experimento, foi lhes submetido um *questionário qualitativo* (Anexo D), com o intuito de obter um indicador de qualidade para a metodologia proposta. Como forma de obter a opinião direta dos participantes, foi pedido, na última questão, que os mesmos opinassem de forma discursiva sobre os pontos positivos e negativos da metodologia.

5.3.5 Decidir como lidar com as questões éticas

Pelo fato do experimento envolver seres humanos, houve a necessidade de submeter o projeto ao Comitê de Ética da Universidade Salvador, no âmbito de obter aprovação para a realização do mesmo. O projeto foi devidamente relatado, submetido e aprovado por parte do comitê. No Anexo E encontra-se o Parecer Consubstanciado, o qual representa o documento de aprovação do experimento.

O experimento foi conduzido preservando o anonimato dos participantes. Foi especificado no termo de Consentimento Livre e Esclarecido (Anexo F) que as

informações pessoais não serão divulgadas. Além disso, todos os voluntários participantes possuíam mais de dezoito (18) anos de idade e gozavam de plena capacidade física e mental.

5.3.6 Estabelecer forma de avaliar, interpretar e apresentar os resultados

A apresentação dos resultados foi primariamente realizada em forma de gráficos com dados quantitativos, posteriormente, foram realizadas análises qualitativas e subjetivas a respeito das informações coletadas. Os resultados encontram-se no Capítulo 0 desta dissertação.

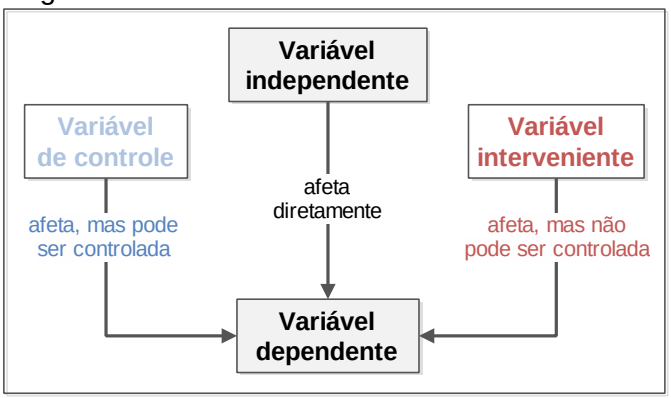
5.4 VARIÁVEIS DO EXPERIMENTO

Segundo Jung (2004) as variáveis são grandezas que fazem parte do fenômeno observado na experimentação científica e que podem variar ao longo do tempo. Segundo o autor elas podem ser classificadas como:

a) **Variável independente** – Representa os aspectos determinantes para que ocorra um determinado resultado. É a causa direta de uma consequência a ser observada.
b) **Variável dependente** – Representa os aspectos observados no experimento. É a consequência ou resposta de algo que foi estimulado.
c) **Variável de controle** – São aspectos que interferem na variável dependente, mas podem ser mitigados por meio de manipulação deliberada.
d) **Variável interveniente** – De maneira análoga à variável de controle, são aspectos que interferem na variável dependente, mas que não podem ser manipuladas pelo observador.

A Figura ilustra o conceito de variáveis de um experimento.

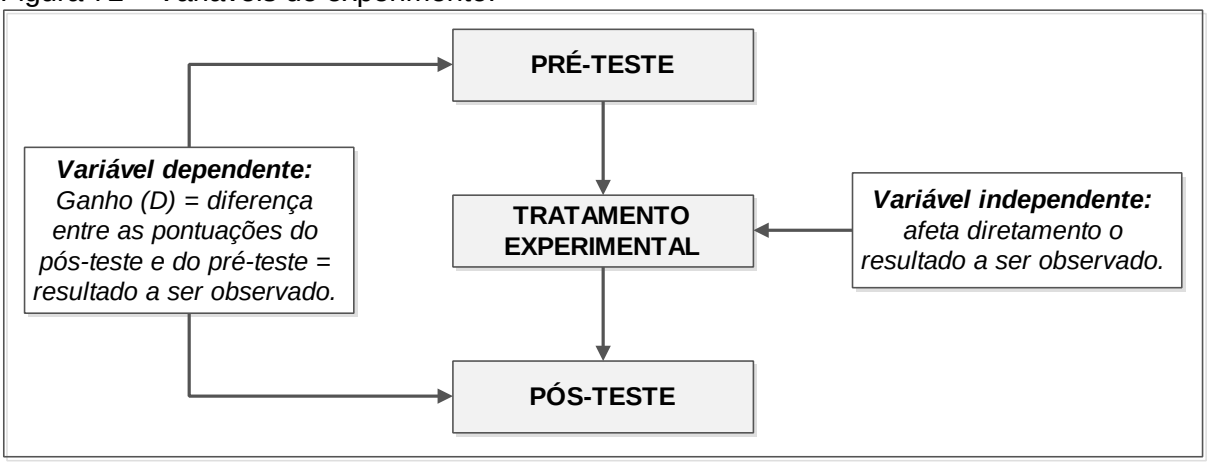
Figura 71 – Conceito de variável



Fonte: Autor desta dissertação (2016).

Neste trabalho, a diferença de desempenho entre o pré-teste e o pós-teste (D) foi definido como variável dependente (VD), pois ela representa o aspecto a ser observado como resultado. O tratamento experimental (T) foi definido como variável independente (VI), tratamento este que é representado pela aplicação da metodologia lúdica aos alunos. Isto foi definido pois considerou-se o experimento como o aspecto que afeta diretamente o resultado a ser observado. A Figura ilustra a definição das variáveis do experimento proposto por esta dissertação.

Figura 72 – Variáveis do experimento.



Fonte: Autor desta dissertação (2016).

5.5 AMEAÇAS À VALIDADE

Os dados obtidos por um experimento científico podem ser influenciados por outros fatores não considerados, os quais afetam a confiabilidade dos resultados. Portanto,

faz-se necessária a definição das principais ameaças que podem alterar a validade do experimento, com o intuito de mitigá-las.

Neste contexto, a bibliografia científica classifica a validade de um experimento como **validade interna** e **validade externa**, bem como as ameaças a essas validades. Segundo Bandeira (2013), a **validade interna** se refere à confiabilidade dos procedimentos adotados durante a execução do experimento. A **validade externa** de uma pesquisa se refere à capacidade de generalizar os resultados, obtidos a partir de uma amostra, para toda população, com o objetivo de tornar as conclusões úteis para todo o grupo populacional.

No contexto deste trabalho, entende-se o **grupo amostral** como sendo os alunos participantes do experimento proposto e o **grupo populacional** como sendo todos os alunos iniciantes dos cursos de computação e engenharias.

5.5.1 Ameaças à validade interna

Bandeira (2013) define os principais vieses que ameaçam a validade interna da VI de uma pesquisa científica. Para garantir a validade interna do experimento realizado, foram adotados os seguintes procedimentos:

a) ***Distribuição randomizada dos participantes*** – visou garantir a distribuição equilibrada, anulando os efeitos do viés ***regressão estatística em direção à média***.
b) ***Testes individuais*** – visou mitigar o viés ***história interna***, pois evita a interação entre os participantes e a possível interferência desleal de um determinado grupo em detrimento do outro.
c) ***Aplicação em dia letivo*** – visou mitigar o viés ***história***, pois entende-se que a aplicação do experimento em dia letivo não interfere nas expectativas rotineiras dos participantes.
d) ***Aplicação consecutiva –*** visou mitigar os vieses ***mortalidade experimental***, ***maturação*** e ***testagem***. Entende-se que evitando grandes intervalos de tempo, como dias, evita-se a evasão dos participantes, possíveis mudanças em seus fatores biológicos, psicológicos e que eles familiarizem-se com os padrões de resposta dos questionários.

e) ***Manutenção do estilo das questões*** – visou mitigar o viés ***instrumentação***, mantendo o estilo de questionário tanto no pré-teste como no pós-teste. Isso evita que possíveis mudanças nos resultados possam ser atribuídas às mudanças no instrumento de medida adotado.

Uma observação importante é que não houve definição de grupo de controle para este experimento. Portanto, pode este representar uma possível ameaça à validade aos resultados aqui apresentados. É importante salientar também que, como não houve definição de grupo de controle, o viés ***seleção*** não foi considerado.

5.5.2 Ameaças à validade externa

De maneira análoga, Bandeira (2013) define os principais vieses que ameaçam a validade externa. Para garantir a validade externa do experimento realizado foram adotados os seguintes procedimentos:

- **Executar o experimento no próprio ambiente acadêmico** – visou mitigar o viés ***efeitos reativos da situação experimentada***. Entende-se que, desta forma, evita-se a criação de um ambiente artificial, o qual pode influenciar negativamente nas expectativas dos participantes para com o experimento.

Uma observação importante é que, pelo fato de não ter havido grupo de controle, pode o viés ***interação entre o pré-teste e a intervenção*** representar uma possível ameaça à validade aos resultados aqui apresentados.

É importante salientar que, pelo fato de o delineamento dos grupos ter sido realizado de forma aleatória, não houve a necessidade de se considerar o viés ***Interação entre a seleção e a intervenção***, pois, desta forma, considera-se que os participantes foram alocados de forma homogênea. Outro fato importante é que o experimento teve apenas uma VI, portanto, o viés ***interferência de tratamentos múltiplos*** também foi desconsiderado.

6 RESULTADOS E ANÁLISE DO EXPERIMENTO

Este capítulo apresenta e discute os resultados da pesquisa experimental realizada como método de avaliação da metodologia lúdica proposta por esta dissertação. A princípio são apresentados os resultados da análise quantitativa. Por conseguinte, são apresentados os resultados da análise qualitativa. Por fim são apresentadas as conclusões.

6.1 INTRODUÇÃO

Conforme fora descrito no delineamento da pesquisa (Seção 0), o experimento realizado consistiu em duas análises: (i) *Análise quantitativa*, a qual, por meio do método *pretest-posttest* (DIMITROV; RUMRILL, 2003), buscou reunir indicadores de desempenho dos alunos antes e depois da aplicação da metodologia proposta. (ii) *Análise qualitativa*, a qual, por meio de um questionário formatado em função da escala de Likert (1932), buscou reunir, por parte dos alunos participantes do estudo, indicadores qualitativos acerca da metodologia aplicada.

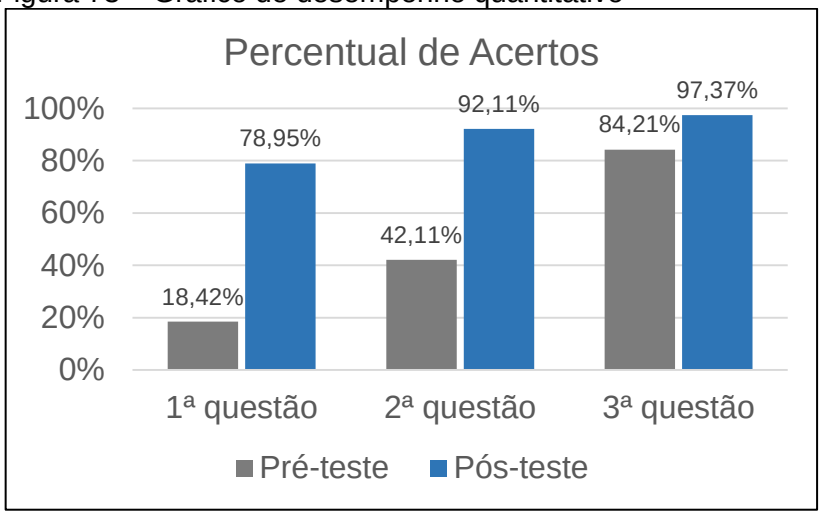
É importante reafirmar que ambas as análises foram realizadas em um mesmo dia, com estudantes da Universidade Salvador, dos cursos de Engenharia da Computação, Engenharia Elétrica e Engenharia Mecatrônica. Foram utilizadas três turmas da disciplina de Estrutura de Dados com Orientação a Objetos, contemplando os turnos matutino, vespertino e noturno. Vale ressaltar que as turmas são mistas, envolvendo alunos dos três cursos de engenharia.

6.2 ANÁLISE QUANTITATIVA

O processo de análise quantitativa teve por objetivo medir numericamente a eficácia do modelo proposto por esta dissertação. Para tanto, são apresentados em forma de gráfico os dados numéricos resultantes dos questionários aplicados na pesquisa. Por meio do contraste de desempenho dos alunos entre o pré-teste e o pós-teste, foi mensurado o ganho de conhecimento que eles obtiveram acerca do conceito de recursividade.

O gráfico ilustrado na Figura , mostra os resultados do desempenho de todos os 38 alunos participantes dos três turnos do experimento. O contraste entre os índices de acerto entre o pré-teste e o pós-teste, mostra o real ganho de desempenho dos alunos provido pela metodologia.

Figura 73 – Gráfico de desempenho quantitativo



Fonte: Autor desta dissertação (2016).

A primeira questão (Triângulo de Sierpinski) apresentou o menor índice de acertos. Apenas 7 alunos acertaram o pré-teste e 30 acertaram o pós-teste. A segunda questão (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 (Cálculo Fatorial) apresentou o maior índice de acertos, 32 alunos acertaram a questão no pré-teste e 37 no pós-teste.

Quando se analisa os índices de acertos para as três questões, fica perceptível o fato de ter havido um crescimento progressivo de dificuldade entre as questões. O Quadro mostra o percentual e o número absoluto de acertos das questões para todos os participantes, onde é possível observar que houve um aumento progressivo no número de acertos entre as Questões 1, 2 e 3.

Quadro 3 – Quadro de desempenho

Questão	Pré-teste		Pós-teste	
	%	Nº	%	Nº
1ª Questão	18,42%	7	78,95%	30
2ª Questão	42,11%	16	92,11%	35
3ª Questão	84,21%	32	97,37%	37

Fonte: Autor desta dissertação (2016).

De maneira geral, os dados obtidos na análise quantitativa mostram que houve alteração positiva da variável dependente (contraste de desempenho dos alunos entre o pré-teste e o pós-teste) e que esta alteração pode sim ser atribuída à intervenção experimental proposta pela pesquisa. O Quadro mostra, em valores, a alteração da variável dependente, onde é possível observar que, para todas as questões, houveram variações positivas, totalizando, entre as três questões, uma média de **41,23%** de ganho desempenho por parte dos alunos.

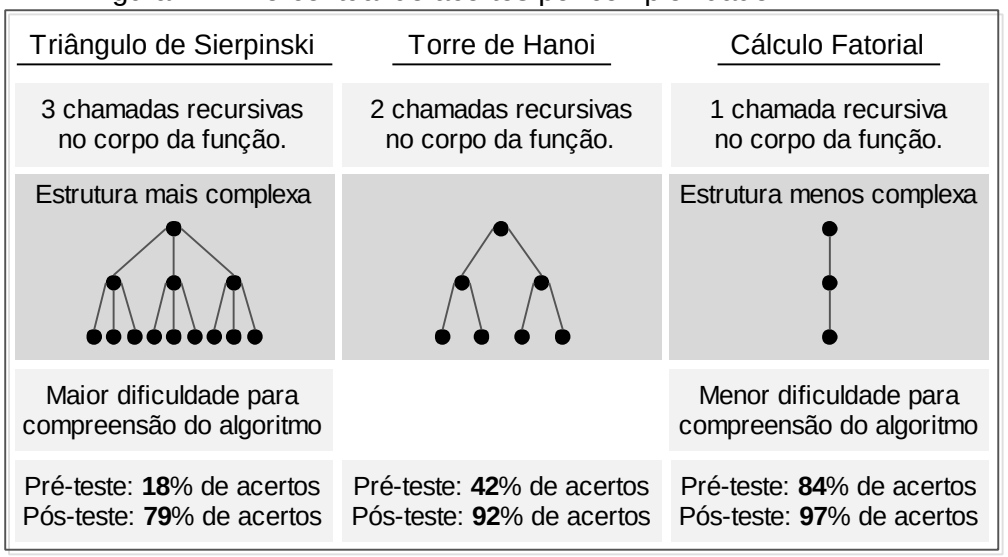
Quadro 4 – Alteração da variável dependente

ALTERAÇÃO DA VARIÁVEL DEPENDENTE (Contraste no ganho de desempenho entre o pós-teste e o pré-teste)			
Questão	Desempenho Pré-teste	Desempenho Pós-teste	Variável Dependente - variação de desempenho (*pós-teste – pré-teste*)
1ª Questão	18,42%	78,95%	60,53%
2ª Questão	42,11%	92,11%	50%
3ª Questão	84,21%	97,37%	13,16%
MÉDIA	**48,24%**	**89,47%**	**41,23%**

Fonte: Autor desta dissertação (2016).

É importante observar que esta diferença progressiva do número de acertos entre as questões é proporcional à diferença de complexidade entre as mesmas. Complexidades essas, referentes às estruturas hierárquicas de suas chamadas recursivas, as quais foram abordadas em detalhes na Seção 0. Esta observação sugere que a complexidade da estrutura hierárquica das chamadas recursivas e, consequentemente, a quantidade de chamadas recursivas existentes no corpo da função são diretamente proporcionais à complexidade para o entendimento e compreensão do algoritmo. A Figura ilustra esta percepção, onde é possível observar que, ao passo que a estrutura hierárquica do algoritmo torna-se mais complexa, o percentual de acertos diminui.

Figura 74 – Percentual de acertos por complexidade



Fonte: Autor desta dissertação (2016).

Esta relação de causa e efeito observada sugere grande responsabilidade de se construir nos alunos iniciantes fortes modelos mentais sobre estruturas hierárquicas em forma de árvores. Neste sentido, entende-se que o modelo proposto por esta dissertação possui grande relevância instrucional, pois fornece um mecanismo de observação direta da relação existente entre o código recursivo e a complexidade de sua estrutura hierárquica.

Com relação à terceira questão (Cálculo do fatorial), percebeu-se a sua má formulação, pois não se premeditou que o método matemático para sua resolução fosse bem mais simples de se utilizar, quando comparado ao método algorítmico. Chegou-se a esta conclusão por dois fatores: (i) houve pouco contraste entre o pré-teste e o pós-teste para a terceira questão (conforme mostra a Figura); e (ii) os próprios alunos identificaram este aspecto (conforme relata o Quadro).

A solução proposta para tal erro foi, no lugar de pedir o resultado da expressão, pedir que o aluno identifique a quantidade de vezes que a função recursiva é evocada. Entende-se que desta forma, o aluno não poderia utilizar o método matemático para resolver o fatorial, sendo ele obrigado a realizar a análise algorítmica de chamadas recursivas.

A proposta seria substituir:

a) ***"Com base no algoritmo abaixo, informe o valor da variável result (linha 13):"*** Por:

b) ***"Com base no algoritmo abaixo, informe a quantidade de vezes que a função void Hanoi (...) é chamada:"***

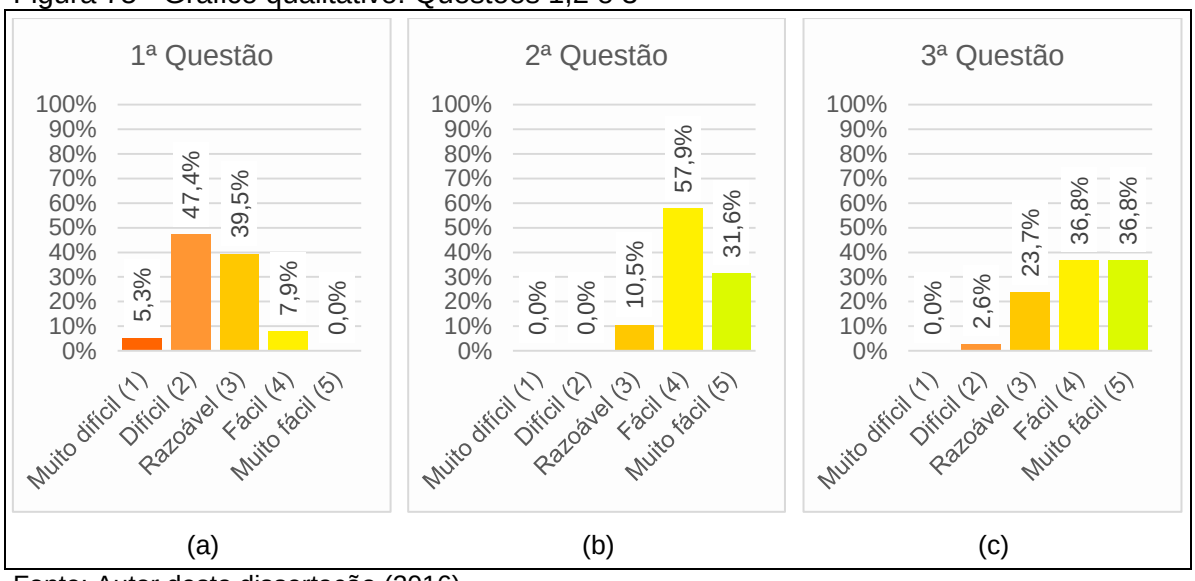
Para maiores detalhes sobre a referida questão acesse o Anexo C.

6.3 ANÁLISE QUALITATIVA

A análise qualitativa teve por objetivo verificar o grau de concordância ou discordância dos alunos com relação aos diversos aspectos da metodologia proposta. A análise foi aplicada por meio de um questionário contendo sete perguntas formatadas em função da escala de Likert (1932) e uma questão discursiva, na qual os alunos opinaram com suas próprias palavras acerca da metodologia. O questionário foi submetido aos alunos ao final do experimento. A cópia deste questionário encontra-se no Anexo D desta dissertação.

6.3.1 Resultados do questionário qualitativo

Figura 75 - Gráfico qualitativo: Questões 1,2 e 3



Fonte: Autor desta dissertação (2016).

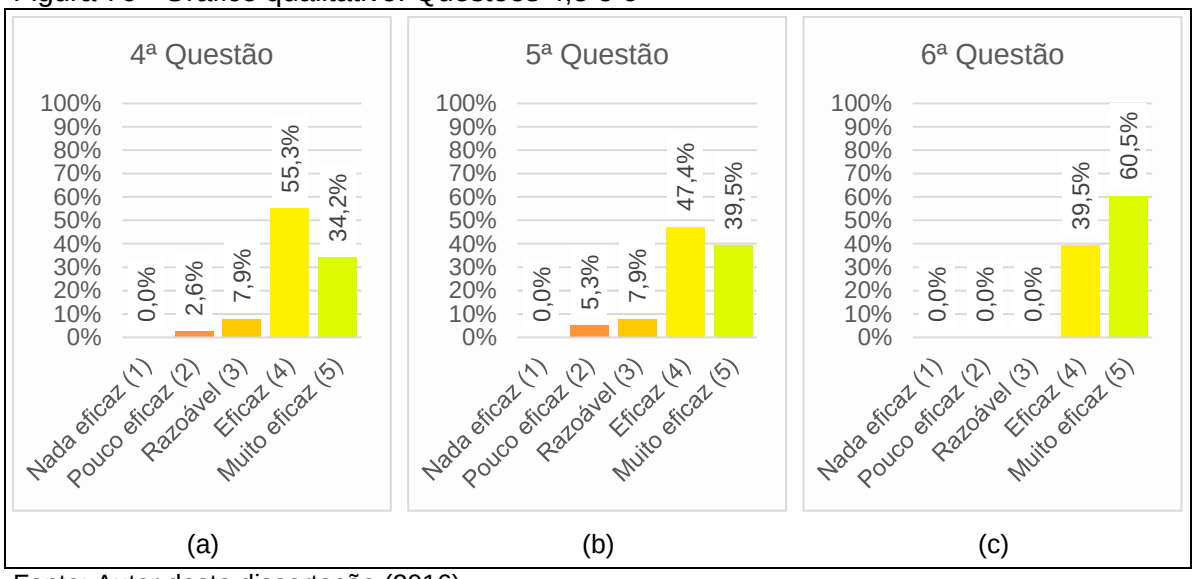
Na primeira questão foi pedido aos alunos que expressassem o grau de dificuldade enfrentada por eles para responder ao **pré-teste**. O Gráfico A apresentado na Figura mostra que a maioria dos alunos avaliaram como “difícil” as questões aplicadas no pré-teste, antes da aplicação da metodologia lúdica.

Este quadro se inverte na Questão 2 (Gráfico B - Figura), na qual foi pedido aos alunos que expressassem o grau de dificuldade enfrentada por eles para responder ao **pós-teste**. A maior parte dos alunos avaliam as questões aplicadas no pós-teste como “fácil”. Portanto, entende-se que após a aplicação da metodologia lúdica os alunos obtiveram uma melhor compreensão dos problemas algorítmicos.

Na Questão 3 foi pedido que os alunos expressassem a dificuldade enfrentada para **compreender o funcionamento da metodologia**. O Gráfico C da Figura mostra que a maior parte dos participantes do experimento avaliaram com “fácil” e “muito fácil”. Isso deixa claro que a metodologia não apresentou dificuldades em ser aplicada como ferramenta de ensino.

Nas Questões 4, 5 e 6 foi pedido aos alunos que avaliassem a eficácia da metodologia para a compreensão dos algoritmos.

Figura 76 - Gráfico qualitativo: Questões 4,5 e 6



Fonte: Autor desta dissertação (2016).

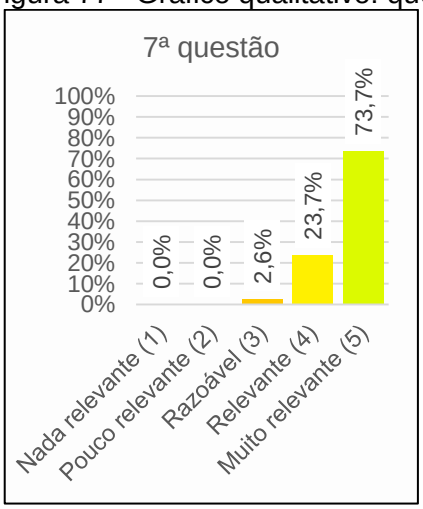
Na 4ª Questão foi pedido que eles avaliassem a eficácia da metodologia para compreensão do algoritmo de Sierpinski. Os resultados (Figura – Gráfico A) mostram que os alunos a avaliaram como “eficaz” e “muito eficaz”, denotando a importância da metodologia para a compreensão do referido algoritmo.

De maneira análoga, os resultados expostos para a 5ª Questão (Figura – Gráfico B), na qual foi pedido que os alunos avaliassem a metodologia para a compreensão do algoritmo de Hanoi, reafirma a eficácia da proposta.

Na 6ª Questão ((Figura – Gráfico C) foi pedido que os alunos avaliassem a metodologia para compreensão do algoritmo do Fatorial. Os resultados mostram, mais uma vez, a potencialidade da metodologia.

A sétima e última questão qualitativa (Figura) pediu para que os alunos avaliassem a relevância do uso de uma abordagem lúdica, no lugar da textual, para o ensino de conceitos de programação. Os resultados ilustrados no gráfico mostram que a maior parte dos alunos qualificou este aspecto como “muito relevante”, indicando o alto grau de aceitação da metodologia proposta.

Figura 77 - Gráfico qualitativo: questão 7



Fonte: Autor desta dissertação (2016).

6.3.2 Questão discursiva

No Quadro são exibidas as mais relevantes opiniões dos alunos participantes do experimento. Tais apreciações foram obtidas por meio das respostas à Questão 8 da avaliação qualitativa. Elas foram categorizadas com base na similaridade existente entre os assuntos que tratam, no âmbito de facilitar a discussão dos resultados. Por questões éticas, conforme descrito no termo de consentimento (Anexo F), omitiu-se os nomes dos alunos.

Quadro 5 – Principais opiniões dos participantes do experimento

Assunto	Opinião
Sobre a vantagem de substituir a abordagem baseada em código (sintax-driven) pela metodologia lúdica	*"[...] já conheci alunos que tiveram problemas em entender o código, inclusive eu no início.* ***Uma professora minha me explicou no semestre seguinte com menos código e tornou bem mais fácil o entendimento****, gostei bastante da proposta."*
	"Com a proposta lúdica, eu pude compreender tudo o que ocorria dentro do algoritmo. O design da proposta torna o método de aprendizagem ***muito mais simples do que se eu tentasse entender lendo o código puro****. [...]"*
	"Uma ideia excelente, pois a forma como o método lúdico dá ***o passo-a-passo da lógica, fica registrada na mente mais facilmente do que com a versão textual****."*
	"É uma proposta interessante, pois ***imagens ou animações facilitam para as pessoas que têm dificuldades em absorver o código*** *[...]."*
Sobre as vantagens da metodologia lúdica para a compreensão de conceitos abstratos	*"****Gostei da proposta lúdica para entender melhor os algoritmos****, principalmente o de Hanoi, que eu não entendi nada a primeira vez que vi."*
	"Achei uma ***maneira muito mais eficiente para se entender o que o problema pede****, tanto para quem tem dificuldades, como para quem não tem. Foi uma experiência muito produtiva."*
	"Foi muito bom, pois ***pudemos ver na prática todos os passos percorridos pelo sistema, trazendo inovação e interatividade para o método de ensino****."*
	"A proposta lúdica se mostrou ***muito viável para melhorar o aprendizado****."*
	"Consegui aumentar meu entendimento sobre os assuntos abordados em 75% com a metodologia lúdica. ***Meu cérebro conseguiu captar mais sobre o funcionamento do algoritmo*** *e será mais difícil esquecê-lo."*
	"Muito interessante a proposta, pois ***a programação fica muito mais fácil de ser entendida quando comparada com a realidade****."*
	"Achei uma proposta bem interessante e que ***deixou muito mais clara a visualização da lógica*** *do conteúdo abordado."*
	"Bastante interessante, uma vez que tal abordagem ***faz com que o interesse do aluno cresça*** *e transponha limites meramente formais! Parabéns!"*
	"[...] No mais, só queria parabenizar pela iniciativa, pois ***a proposta lúdica retoma o interesse pela programação****, livrando-nos da monotonia de programas enormes."*

Assunto	Opinião
Sobre o uso da metodologia como ferramenta de ensino	*"Considero como uma **ótima ferramenta de aprendizagem**, visto o auxílio da interface gráfica ao passar informações de forma simplificada e objetiva."*
	*"Gostei bastante, **deveria começar a ser aplicado em sala de aula**."*
	*"Achei bastante interessante. A integração da parte lúdica aplicada ao conceito de recursividade ajuda a compreender melhor o assunto abordado, **muito relevante a aplicação para alunos da matéria de estruturas de dados**."*
	*"Achei a proposta muito boa e, **caso entre em funcionamento, ajudará muitos iniciantes que têm dificuldades**."*
	*"De extrema ajuda, maneira prática de visualizar algo abstrato, porém **não deve ser usado como material substituto, funciona como um excelente complemento**. Não alteraria nada."*
	*"Interessante, **dá pra ser utilizada em conjunto com a metodologia atual**."*
Propostas de melhorias: inclusão do código fonte	*"Eu achei interessante essa proposta, até porque facilita o entendimento dos códigos que geralmente são difíceis de entender. Uma sugestão seria **colocar os códigos durante a execução desses programas para servir como um guia**."*
	*"Toda a ideia do projeto é muito interessante e realmente faz uma diferença no ensino. Eu mudaria ou adicionaria dificuldade [...]. Talvez também **adicionar uma forma de ver o algoritmo ou código criado até o momento**."*
	*"A proposta lúdica apresentada possibilitou um melhor entendimento de como o algoritmo segue sua ordem e sua lógica. **Adicionaria uma tela com o código sendo seguido passo-a-passo**, simultaneamente ao jogo."*
	*"A proposta apresentada foi de grande simplicidade para poder compreender a lógica do código. Contudo, ambos são complementos um do outro. Então, seria interessante que ao longo da interatividade da metodologia lúdica **seja, também, mostrado as linhas do código** referente."*
	*"É uma boa iniciativa, uma vez que, de fato, é mais difícil entender apenas a linha de código. Sugiro que os jogos **apresentem o código que está sendo utilizado, em tempo real**."*
Outras propostas de melhorias	*"Não colocaria **indicações tão explícitas** dos passos necessários para o desenvolvimento e conclusão da atividade lúdica."*
	*"[...] **mudaria o caminho 'premeditado' do passo a passo**, sem dicas, para que o aluno tente por conta própria a direção que o personagem no game deve ir."*
	*"Proposta muito eficiente, porém, houve uma dificuldade inicial para compreender o funcionamento. Para melhorá-la, seria interessante **adicionar instruções**."*
	*"[...] Não mudaria nada, e, em questão de adicionar, eu apenas espero que **mais algoritmos sejam explicados com essa proposta**. Isso pode influenciar mais pessoas a programar."*
	*"[...] não é somente linhas de código e sim também uma aprendizagem fácil e divertida. **Sugiro mais animações e layouts**."*
	*"Muito bom, pois muitas vezes pessoas têm facilidade de aprender olhando e interagindo com o processo, **acredito que dando opção de alterar o valor de 'n'**."*
	*"No pré-teste colocar opções como **(nenhuma das alternativas) para que o aluno não acerte por 'chute'**. A **questão de fatorial é fácil, pois o aluno conhece o conceito matemático da função**, e já parte direto para o cálculo sem analisar o código. **Reformatar a pergunta** ou questionar usando outro exemplo de função recursiva talvez seja mais eficaz."*

Fonte: Autor desta dissertação (2016).

Com base nas opiniões dos estudantes, expressas no Quadro , foi possível perceber que houve real aceitação da metodologia. Desta forma, conclui-se que ela:

a) Torna mais eficiente a compreensão do problema.
b) É uma inovação metodológica.
c) Torna concreto um conceito abstrato.
d) Aumenta o interesse do aluno pela programação.
e) Tem grande potencial como metodologia complementar para alunos iniciantes de universidades.
f) Deve ser utilizada em conjunto com a abordagem baseada em código.

6.4 CONCLUSÕES DO EXPERIMENTO

Os resultados obtidos por meio do experimento sugerem a eficácia da metodologia proposta por esta dissertação. Os gráficos quantitativos apresentados mostraram que os alunos obtiveram um ganho de desempenho nos testes submetidos e que existe uma relação de causa e efeito entre este desempenho e a metodologia proposta.

Portanto, com base nas informações obtidas, pode-se responder às perguntas definidas para a pesquisa (Seção 0):

a) *Os alunos enfrentaram dificuldades para entender o funcionamento da ferramenta?* **Resposta**: Conforme ilustra a Questão 3 (Gráfico C da Figura), os alunos avaliaram a ferramenta como de “fácil” ou “muito fácil” entendimento, sugerindo que não houve dificuldades com esse aspecto.
b) *A ferramenta conseguiu construir os modelos mentais acerca do fluxo de execução dos algoritmos recursivos?* **Resposta**: Conforme os próprios comentários dos alunos, apresentados no Quadro , houve um melhor esclarecimento acerca da lógica de funcionamento dos algoritmos apresentados.
c) *Ao final do experimento foi possível observar uma melhora geral de desempenho nos testes que pudesse ser atribuída à metodologia proposta?* **Resposta**: Conforme ilustra o Quadro , a variável dependente indica que

houve melhora significativa de desempenho dos alunos entre o pré-teste e o pós-teste.

d) *Os alunos aprovaram a metodologia como ferramenta de apoio ao ensino de conceitos de programação de computadores?* **Resposta**: Conforme os próprios comentários dos alunos, apresentados no Quadro , houve grande aceitação da metodologia e expectativa para que ela seja utilizada como ferramenta de apoio no ensino de conceitos de programação de computadores.

7 CONCLUSÕES E TRABALHOS FUTUROS

O trabalho descrito por esta dissertação propôs um modelo lúdico voltado para o ensino de conceitos de programação de computadores. O trabalho propôs conciliar a investigação da problemática do ensino da programação com o estudo da psicologia cognitiva, com o intuito de obter prerrogativas que fundamentassem a criação de uma metodologia que, por meio de metáforas, permitisse que os alunos criassem bons modelos cognitivos.

A revisão bibliográfica forneceu importantes evidências sobre a problemática do ensino de programação de computadores, servindo de fundamentação para a proposta e mostrando a grande relevância da pesquisa científica na área. Os conhecimentos obtidos a partir do estudo da psicologia cognitiva permitiram a criação de critérios que nortearam a definição dos aspectos visuais do modelo proposto.

A metodologia foi proposta por meio da descrição detalhada de um modelo para o ensino de recursividade, o qual foi fundamentado com base na psicologia cognitiva e implementado por meio da construção de ferramentas lúdicas que simulam a execução de três algoritmos recursivos.

O modelo foi validado por meio de uma pesquisa experimental que comprovou, através de análises quantitativa e qualitativa, sua relevância e seu grande potencial como metodologia de ensino de conceitos de programação a ser aplicada nos cursos superiores.

Desta forma, entende-se que este trabalho cumpriu o seu papel, realizando todas as etapas necessárias para o alcance do seu objetivo:

a) **Objetivo**: *Propor uma metodologia lúdica, baseada na psicologia cognitiva, para facilitar o aprendizado do conceito de programação por parte dos alunos iniciantes dos cursos de computação e engenharias*.

No que diz respeito à problemática levantada no escopo inicial, conclui-se que o trabalho realizado nesta dissertação conseguiu reunir informações suficientes para respondê-la:

b) **Problema**: *Como mitigar os elevados índices de desistência dos alunos iniciantes em computação, devido às dificuldades enfrentadas por eles com as matérias introdutórias de programação de computadores?*
c) **Resposta**: Implementando abordagens lúdicas à abordagem textual que é utilizada atualmente nas universidades.

No que diz respeito à bibliografia científica, conclui-se que os resultados obtidos por este trabalho estão de acordo com as conclusões de Hannafin e Rieber (1989), que condenam o uso da tecnologia para ditar as diretrizes do ensino e propõem a adaptação da tecnologia para atender às demandas do aprendizado. Entende-se que a interação lúdica proposta foi construída não com o intuito de ditar as diretrizes do ensino, mas como apoio ao aprendizado do estudante. Houve uma investigação das complexidades inerentes ao conceito de recursividade e, a partir de então, adaptou-se a tecnologia para atender às necessidades imposta por este conceito.

Hannafin et al. (1988) relatam que o objetivo principal dos modelos instrucionais é o de proporcionar atividades que gerem informações contextuais, as quais possam apoiar o processo de recordação por meio da criação do que eles chamam de "plano de retorno" mental. Neste contexto, a pesquisa realizada por este trabalho mostrou que a metodologia proposta criou na mente dos alunos os referidos "planos de retorno", que os ajudaram a lembrar dos processos de resolução algorítmicos.

Outra pesquisa que está de acordo com os resultados dessa dissertação é o trabalho de Burkhard (2004), que advoga pela criação de ferramentas de visualização direcionadas a cada área do conhecimento, sob a prerrogativa de que o cérebro humano possui melhor capacidade de reconhecer padrões em representações visuais. O referido autor afirma que as capacidades cognitivas são aumentadas quando as habilidades visuais são utilizadas. Neste sentido, este trabalho mostrou que o uso das habilidades visuais, incorporadas à metodologia

instrucional de ensino de programação, proporcionou um significativo aumento das capacidades cognitivas dos alunos.

No que diz respeito à Cognitive Load Theory (CLT), principal teoria da psicologia cognitiva identificada no mapeamento sistemático e que advoga pela redução das cargas cognitivas durante o processo de aprendizado, foi possível perceber que a metodologia conseguiu gerenciar a complexidade do assunto por meio de metáforas visuais, conforme formaliza Abdul-Rahman e du Boulay (2014). Entende-se, do ponto de vista da CLT, que os resultados apresentados por este trabalho demonstraram o sucesso da metodologia em gerenciar as cargas cognitivas denominadas *Extraneous Load* (EL) e *Intrínsic Load* (IL), conforme fora afirmado por Morrison et al. (2014), liberando espaço na mente dos alunos para a carga cognitiva denominada *Germane Load* (GL), permitindo, desta forma, a construção dos modelos mentais acerca do conceito de recursividade. Para maiores detalhes sobre a CLT veja a Seção 0.

Os resultados obtidos expõem a real validade da iniciativa proposta por esta dissertação e encorajam o seu desenvolvimento, no âmbito de torna-la uma ferramenta instrucional. Neste sentido propõe-se prosseguir esta pesquisa por meio de:

a) Melhorias na metodologia proposta. Principalmente por meio da inclusão da perspectiva *Code*, na qual o aluno poderá visualizar o código fonte gerado pelo algoritmo (conforme fora solicitado pelos alunos participantes do experimento).
b) Implementação de mais algoritmos recursivos, com o intuito de avaliar a visualização de outros problemas.
c) Extensão da metodologia aos outros conceitos computacionais, tais como, *estruturas de repetição*, *arrays* e *gerenciamento de memória* (conforme fora solicitado pelos alunos participantes do experimento), com o intuito de tornar a metodologia aplicável como ferramenta de apoio ao ensino nas universidades.

No que diz respeito aos métodos de avaliação, pretende-se realizar mais experimentos com alunos iniciantes:

a) Utilizando maior quantidade de participantes.
b) Definindo melhores delineamentos, que incluam grupos de controle.
c) Utilizando técnicas de análise quantitativa mais precisas, tal como, o método estatístico *Analysis of Variance* (ANOVA), no âmbito de tornar os resultados mais confiáveis e aplicáveis à comunidade científica.

REFERÊNCIAS

ABD-EL-HAFIZ, Salwa K.; BASILI, Victor R. A knowledge-based approach to the analysis of loops. **IEEE Transactions on Software Engineering**, v. 22, n. 5, p. 339-360, 1996.

ABDUL-RAHMAN, Siti-Soraya; DU BOULAY, Benedict. Learning programming via worked-examples: Relation of learning styles to cognitive load. **Computers in Human Behavior**, v. 30, p. 286-298, 2014.

ALAOUTINEN, Satu; SMOLANDER, Kari. Student self-assessment in a programming course using bloom's revised taxonomy. In: ANNUAL CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION, 15., 2010. **Proceedings …** 2010. p. 155-159.

AMBRÓSIO, Ana Paula et al. Identifying cognitive abilities to improve CS1 outcome. In: **Frontiers in** EDUCATION CONFERENCE (FIE), 2011. **Proceedings …** 2011. p. F3G-1-F3G-7.

ATKINSON, Richard C.; SHIFFRIN, Richard M. Human memory: A proposed system and its control processes. **Psychology of learning and motivation**, v. 2, p. 89-195, 1968.

BANDEIRA, M. **Validade interne e externa de uma pesquisa.** 2013. Disponível em: http://www.ufsj.edu.br. Acesso em: 21 jan. 2016.

BURCH, C. **Writing a Towers of Hanoi program**, 1999. Disponível em: http://www.cs.cmu.edu. Acesso em: 16 mar. 2016.

BURKHARD, Remo Aslak. Learning from architects: the difference between knowledge visualization and information visualization. In: INTERNATIONAL CONFERENCE INFORMATION VISUALISATION, 8., 2004. **Proceedings…** 2004. p. 519-524.

CARNEGIE MELLON UNIVERSITY. **Alice Project**. Disponível em: http://www.alice.org. Acesso em: 4 fev. 2016.

CASPERSEN, Michael E.; BENNEDSEN, Jens. Instructional design of a programming course: a learning theoretic approach. In: INTERNATIONAL WORKSHOP ON COMPUTING EDUCATION RESEARCH, 3., 2007. **Proceeding…** 2007. p.111-122.

CHI, Michelene TH et al. Self-explanations: How students study and use examples in learning to solve problems. **Cognitive science**, v. 13, n. 2, p. 145-182, 1989.

COLLIN, C. **O livro da psicologia**. Tradução de Clara M. Hermeto e Ana Luisa Martins. São Paulo: Globo, 2012.

COMPUTING RESOURCE ASSOCIATION. **Taulbee Survey**. Disponível em: <http://cra.org/resources/taulbee-survey> Acesso em: 12 ago. 2015.

COOPER, Stephen; NAM, Yoon Jae; SI, Luo. Initial results of using an intelligent tutoring system with Alice. In: ACM ANNUAL CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION, 17., 2012. **Proceedings …** 2012. p. 138-143.

COVINGTON, R.; BENEGAS, L. A cognitive based approach for teaching programming to computer science and engineering students. In: ASEE PEER 2005 ANNUAL CONFERENCE, 2005, Portland, OR, USA. ASEE. **Proceedings…** 2005. p.10.17.1-10.17.25.

DE SMET, Benoít et al. Taupe: Visualizing and analyzing eye-tracking data. **Science of Computer Programming**, v. 79, p. 260-278, 2014.

DILLON, Edward; ANDERSON, Monica; BROWN, Marcus. Comparing mental models of novice programmers when using visual and command line environments. In: ANNUAL SOUTHEAST REGIONAL CONFERENCE, 50., 2012. **Proceedings…** 2012. p. 142-147.

DIMITROV, Dimiter M.; RUMRILL JR, Phillip D. Pretest-posttest designs and measurement of change. **Work**, v. 20, n. 2, p. 159-165, 2003.

EAGLE, M.; BARNES, T. Wu's Castle: Teaching arrays and loops in a game. In: ANNUAL CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION ITiCSE, 13., 2008, Madrid, Spain. **Proceedings…** Madrid, Spain: ACM, 2008. p. 245-249.

FRENSCH, P.A. Cognitive psychology: overview. In: INTERNATIONAL Encyclopedia of the Social & Behavioral Sciences. [S.l]: [s.n.], 2001. p.2147-2154.

GIL, A.C. **Como elaborar projetos de pesquisa**. 5. ed. São Paulo: Atlas, 2000.

GUIBERT, N.; GUITTET, L.; GIRARD, P.; A study of the efficiency of an alternative programming paradigm to teach the basics of programming. In: IFIP WORLD CONFERENCE ON COMPUTERS IN EDUCATION WCCE, 8., 2005, Cape Town, South Africa. **Proceedings…** South Africa: Emerald Group Publishing Ltd, 2005.

GUIMARÃES, A.; LAGES, N. **Algoritmos e estruturas de dados**. 38. ed. Porto Alegre: Bookman, 2012.

PHILLIPS, Timothy L.; HANNAFIN, Michael J.; TRIPP, Steven D. The effects of practice and orienting activities on learning from interactive video. **Educational Technology Research and Development**, v. 36, n. 2, p. 93-102, 1988.

HANNAFIN, Michael J.; RIEBER, Lloyd P. Psychological foundations of instructional design for emerging computer-based instructional technologies: Part II. **Educational Technology Research and Development**, v. 37, n. 2, p. 102-114, 1989.

HERNANDEZ, C.C. et al. **Teaching programming principles through a game engine**. New York: Sciences-New York, p. 1-8, 2010.

INCEPTION. Christopher Nolan. United Kingdom: Warner Bros. Pictures, 2010.148 min.

JESUS, E.A.; RAABE, A.L.A. Interpretações da Taxonomia de Bloom no Contexto da Programação Introdutória. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO SBIE, 2009, Florianópolis. **Anais...** 2009. p.2-6.

JUNG, C. F. **Metodologia científica:** ênfase em pesquisa tecnológica. 4. ed. Rio de Janeiro: Axel Books, 2004.

KHAIRUDDIN, N.N.; HASHIM, K. Application of Bloom's taxonomy in software engineering assessments. In: CONFERENCE ON APPLIED COMPUTER SCINCE, 8., 2008, Venice, Italy. **Proceedings…** 2008. p.66-69.

KOFFKA, K. **Principles of Gestalt Psychology**. London, UK: Lund Humphries, 1935.

KRONBAUER, A.H. **Um Modelo de avaliação de usabilidade de aplicativos para smartphones baseado na captura automática de interações com o usuário**. 2013. 248 f. Tese (Doutorado)- Universidade Federal da Bahia (UFBA), Universidade Salvador (UNIFACS), Universidade Estadual de Feira de Santana (UEFS), Salvador, Bahia, Brasil, 2013.

KUMAR, A.N. The effect of interleaving an alternate task during tutoring and testing. In: FRONTIERS IN EDUCATION CONFERENCE FIE, 2012, Seattle, WA, USA. **Proceedings…** 2012. p.1-5.

LAUREANO, M. **Estrutura de dados com algoritmos e C**. 1. ed. Rio de Janeiro: Brasport, 2008.

LEWANDOWSKI, G. et al. What novice programmers don't know. In: INTERNATIONAL WORKSHOP ON COMPUTING EDUCATION RESEARCH ICER 5., 2005, Seattle, WA, USA. **Proceedings…** 2005. p.1-12.

LIFELONG KINDERGARTEN. **Scratch project**. Disponível em: <https://scratch.mit.edu/> Acesso em: 9 fev. 2016.

LIKERT, R. **A Technique for the measurement of attitudes**. **Archives of Psychology,** n.140, p. 1-55, 1932.

LISTER, R.; SIMON, B. et al **Not seeing the forest for the trees**. In: ANNUAL SIGCSE CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION ITICSE '06, 11., 2006, Bolonha, Itália. **Proceedings…** 2006. p.118-122.

MA, L. et al. Investigating and improving the models of programming concepts held by novice programmers. **Computer Science Education**, n.21, p.57-80, 2011.

MAYER, R.E. The psychology of how novices learn computer programming. **ACM Computing Surveys CSUR**, n.13, p.121-141, 1981.

MCKEOWN, J. The use of a multimedia lesson to increase novice programmers' understanding of programming array concepts. **Journal of Computing Sciences in Colleges**, n. 19, p.39-50, 2004.

MCLEOD, S.A. **Cognitive Psychology**, 2007a. Disponível em: <http://www.simplypsychology.org>. Acesso em: 18 nov. 2015.

MCLEOD, S.A. **Jean Piaget.** 2009. Disponível em: <http://www.simplypsychology.org>. Acesso em: 5 nov. 2015.

MCLEOD, S.A. **Multi Store Model of Memory**, 2007b. Disponível em: <http://www.simplypsychology.org>. Acesso em: 18 nov. 2015.

MILLER, G.A. The magical number seven, plus or minus two: some limits on our capacity for processing information. Psychological Review, n. 63, p.81-97, 1956.

MORRISON, B.B.; DORN, B.; GUZDIAL, M. Measuring cognitive load in introductory CS: Adaptation of an instrument. In: INTERNATIONAL COMPUTING EDUCATION RESEARCH ICER '14., 2014, Glasgow, Scotland. **Proceedings…** 2014. p.131-138.

NEISSER, U. **Cognitive psychology**. New York: Appleton-Century-Crofts, 1967.

ODIFREDDI, P. **Classical recursion theory:** the theory of functions and sets of natural numbers. Amsterdam: Elsevier, 1999.

PAIVIO, A. **Imagery and verbal processes**. Hillsdale, NJ: Lawrence Erlbaum Associates, 1979.

PARNIN, Chris; RUGABER, Spencer. Resumption strategies for interrupted programming tasks. **Software Quality Journal**, v. 19, n. 1, p. 5-34, 2011.

PETERSEN, Kai; VAKKALANKA, Sairam; KUZNIARZ, Ludwik. Guidelines for conducting systematic mapping studies in software engineering: An update. **Information and Software Technology**, v. 64, p. 1-18, 2015.

POLLACK, Irwin. Information of elementary multidimensional auditory displays. **The Journal of the Acoustical Society of America**, v. 24, n. 2, p. 745-749, 1952.

RENUMOL, V.G.; JANAKIRAM, D.; JAYAPRAKASH, S. Identification of cognitive processes of effective and ineffective students during computer programming. **ACM transactions on computing education**, n.10, p.1-21, 2010

SHAFFER, D.; DOUBE, W.; TUOVINEN, J. Applying Cognitive Load Theory to Computer Science Education. In: ANNUAL WORKSHOP OF THE PSYCHOLOGY OF PROGRAMMING INTEREST GROUP – PPIG, 15.,2003, Keele University, United Kingdom. **Proceedings…** 2003. p. 333-346.

SHARP, H.; ROGERS, Y.; PREECE, J. **Interaction design:** beyond human-computer interaction. 2. ed. New York: John Wiley & Sons, 2007.

SIEGMUND, J. et al. Understanding understanding source code with functional magnetic resonance imaging. In: InTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING ICSE '14, 36.,2014, Hyderabad, India. **Proceedings…** 2014. p.378-389.

SORVA, J. **Visual program simulation in introductory programming education**. 2012. 432 f. Tese (Doutorado)- Aalto University School of Science, Espoo, Finland, 2012.

STEWART, T.C.; ELIASMITH, C. Neural Cognitive Modelling: A Biologically Constrained Spiking Neuron Model of the Tower of Hanoi Task The Tower of Hanoi. In: ANNUAL MEETING OF THE COGNITIVE SCIENCE SOCIETY – COGSCI, 33., 2011, Boston, MA, USA. **Proceedings…** p. 656-661.

STOREY, M. Theories, tools and research methods in program comprehension: past, present and future. In: INTERNATIONAL WORKSHOP ON PROGRAM COMPREHENSION IWPC, 13., 2005, St. Louis, MO, USA. **Proceedings…** 2005. p.181-191.

STROUSTRUP, B. **Programming:** principles and practice using C++. 1. ed. Tradução de Maria Lúcia Blanck Lisbôa. Porto Alegre: Bookman, 2012.

SWELLER, John; VAN MERRIENBOER, Jeroen J.G; P.A.A.S, Fred GWC. Cognitive architecture and instructional design. **Educational psychology review**, v. 10, n. 3, p. 251-296, 1998.

THOMPSON, Errol; KINSHUK. The nature of an object-oriented program: how do practitioners understand the nature of what they are creating?. **Computer Science Education**, v. 21, n. 3, p. 269-287, 2011.

VAN MERRIENBOER, Jeroen JG; KRAMMER, Hein PM. Instructional strategies and tactics for the design of introductory computer programming courses in high school. **Instructional Science**, v. 16, n. 3, p. 251-285, 1987.

VANDEVENNE, L. **Sierpinski Fractals.** 2007. Disponível em: <http://lodev.org>. Acesso em: 15 mar. 2016.

WATSON, John B. Psychology as the behaviorist views it. **Psychological review**, v. 20, n. 2, p. 158, 1913.

WILCOCKS, Derek; SANDERS, Ian. Animating recursion as an aid to instruction. **Computers & Education**, v. 23, n. 3, p. 221-226, 1994.

ZAVALA, A.B.P. **O estudo de funções exponenciais e logarítmicas motivado pela geometria fractal**. 2007. Monografia (Especialização para Professores de Matemática)- UFPR, 2007.

APÊNDICE A – MAPEAMENTO SISTEMÁTICO: PSICOLOGIA DA PROGRAMAÇÃO

Esta etapa teve por objetivo identificar e classificar, por meio de um mapeamento sistemático, os principais estudos que relatavam sobre o uso da psicologia cognitiva no ensino de programação. O intuito foi de entender de que forma eles abordavam o tema no processo de ensino-aprendizagem e quais as suas principais contribuições para o ensino de programação. O capítulo foi finalizado com a sintetização deste conhecimento, o qual foi utilizado como fundamentação científica para a elaboração do modelo lúdico proposto por esta dissertação.

PLANEJAMENTO DO MAPEAMENTO SISTEMÁTICO

Segundo Petersen et al. (2015) o principal objetivo de um mapeamento sistemático é fornecer uma visão geral de uma determinada área de pesquisa e identificar o tipo e a quantidade de trabalhos disponíveis. Além disso, o autor afirma que a principal contribuição do mapeamento sistemático é descobrir padrões, quantidades e frequências de elementos referentes às publicações científicas ao longo do tempo. Nesse sentido, este capítulo visa obter um espectro acerca das publicações do uso da psicologia cognitiva no ensino de programação de computadores, bem como, entender como estes trabalhos e as suas contribuições se posicionam no estado da arte do tema.

Objetivo

O objetivo deste estudo é identificar os mais recentes estudos que relatam o uso da psicologia cognitiva no processo de ensino/aprendizado de conceitos de programação de computadores.

Questões PICO

Com base no objetivo, foram desenvolvidas três questões de pesquisa que norteiam as investigações deste estudo:

1. Como a psicologia cognitiva é aplicada no ensino de conceitos de programação?
2. Quais são os principais tópicos da psicologia cognitiva aplicados no ensino de conceitos de programação?
3. Quais as principais dificuldades identificadas para o aprendizado de programação.

A estrutura *PICO* foi definida da seguinte maneira:

- **População**: artigos científicos que relatam o uso da psicologia cognitiva no ensino de conceitos de programação de computadores.
- **Intervenção**: serão observadas as metodologias para a abordagem da psicologia cognitiva no ensino de programação, suas fundamentações científicas e os resultados expostos nos trabalhos.
- **Comparação**: serão comparadas as técnicas, teorias e a forma como a psicologia foi abordada em cada um dos artigos.
- **Resultados**: são esperados modelos, teorias e afirmações que norteiem e fundamentem a construção do modelo lúdico proposto por esta dissertação.

Critérios de inclusão e exclusão

De acordo com o protocolo definido para esse mapeamento, foram estabelecidos os critérios de inclusão e exclusão, para garantir a seleção de estudos relevantes que podem responder às questões de pesquisa. Eles são definidos no Quadro .

Quadro 6 – Critérios de inclusão e exclusão

INCLUSÃO	EXCLUSÃO.
CI-1: Apresenta conceitos aparentemente próximos à psicologia cognitiva para o ensino de programação.	CE-1: Estudos que não estejam na língua portuguesa ou inglesa.
CI-2: Apresenta-se extremamente importante para o objetivo desta dissertação, pois referem-se diretamente à literatura da psicologia da programação.	CE-2: Relatórios ou documentos que estejam disponíveis na forma de resumos ou apresentações.
CI-3: Aborda o ensino de programação devidamente associado à psicologia cognitiva.	CE-3: Estudos que estejam parcialmente ou totalmente indisponíveis, bem como artigos duplicados.
CI-4: São estudos recentes.	CE-4: Estudos que apresentem resultados pouco significativos ou menos expressivos para o objetivo da dissertação.

Fonte: Autor desta dissertação (2016).

Seleção de fontes de busca

Com o intuito de não ignorar trabalhos científicos possivelmente importantes para o mapeamento, foram adotadas seis bases científicas:

- ACM Digital Library – em modo “*advanced search*.”
- IEEE Xplore Digital Library – em modo “*command search*” e com a opção de pesquisa “*Full Text & Metadata*.”
- ScienceDirect – em modo “*expert search*.”
- Springer Link – com os filtros: *content type="article"*, *discipline="computer science."*
- Scorpus – em modo “*advanced search.*” Com os filtros: *document type="conference paper” e "article", subject area="computer Science."*
- Compendex (Engineering Village) – em modo “*expert search*.”

Construção da string de busca

Segundo às questões de pesquisa anteriormente definidas, foram identificas as principais palavras chaves e seus respectivos sinônimos:

- "teaching programming" – "teach programming" – "teaching to program" – "teach to program" – "teaching coding" – "teach coding" – "teaching to code" – "teach to code" – "learning programming" – "learn programming" – "learning to program" – "learn to program" – "learning coding" – "learn coding" – "learning to code" – "learn to code";
- "cognitive psychology" – "cognitive science"

Foram utilizados os operadores lógicos AND e OR, sendo o AND – de caráter exclusivo – utilizado para separar as palavras chaves, e o OR – de caráter inclusivo – utilizado para separar os sinônimos de cada palavra chave. Como resultado obteve-se a seguinte *string* de busca genérica:

- ("teaching programming" OR "teach programming" OR "teaching to program" OR "teach to program" OR "teaching coding" OR "teach coding" OR "teaching to code" OR "teach to code" OR "learning programming" OR "learn programming" OR "learning to program" OR "learn to program" OR "learning

coding" OR "learn coding" OR "learning to code" OR "learn to code") AND ("cognitive psychology" OR "cognitive science")

Partindo do princípio de que algumas bases de pesquisa abrangem variados domínios do conhecimento – tal como a *Springer*, que abrange ciência, tecnologia, matemática e medicina – e que os termos específicos podem ter significados distintos em cada domínio, houve a necessidade de se realizar uma análise qualitativa para cada sinônimo referente ao ensino de programação. Durante esse processo cada sinônimo foi submetido à cada uma das bases.

O resultado esperado por esta etapa foi possuir uma clara referência de quais sinônimos melhor se adequavam a cada uma das bases, permitindo o refinamento das *strings* e a consequente diminuição da quantidade de artigos indesejados. O Quadr mostra o resultado desta avaliação, onde são exibidas as quantidades de artigos retornados para cada sinônimo, acompanhada de uma classificação que os qualifica como:

- **G** (*good*) – aparentemente todos os artigos abordam programação;
- **R** (*regular*) – grande quantidade de artigos que não tratam do tema;
- **B** (*bad*) – a maior parte dos artigos são indesejados.

Quadro 7 – Matriz avaliativa de palavras chaves

Sinônimos	.ACM		IEEE		ScienceDirect		Springer		Scorpus		Compendex	
	Quantidade	Qualidade	Quantidade	Qualidade	Quantidade	Qualidade	Quantidade	Qualidade	Quantidade	Qualidade	Quantidade	Qualidade
"teaching programming"	1340	G	638	G	301	G	254	G	861	G	376	G
"teach programming"	561	G	293	G	159	G	254	G	142	G	144	G
"teaching to program"	3	G	1	G	2	G	2	R	0		0	
"teach to program"	1	G	1	B	0		2	R	0		0	
"teaching coding"	12	R	14	R	8	B	1	B	12	G	3	G
"teach coding"	9	R	5	G	1	B	1	B	3	G	2	G
"teaching to code"	1	B	0		0		0		0		0	
"teach to code"	0		0		0		0		0		0	
"learning programming"	939	G	571	G	315	G	533	G	769	G	268	G
"learn programming"	648	G	452	G	216	G	533	G	237	G	144	G
"learning to program"	1339	G	473	G	475	G	67	R	914	G	185	G
"learn to program"	490	G	236	R	180	R	68	R	88	G	60	G
"learning coding"	16	R	23	B	25	B	25	B	40	B	3	G
"learn coding"	21	R	10	B	6	B	25	B	1	B	2	G
"learning to code"	43	G	15	G	35	B	3	B	6	G	6	G
"learn to code"	50	R	30	B	36	B	3	B	5	G	1	B

Fonte: Autor desta dissertação (2016).

Este processo permitiu que fosse observado quais termos referentes ao ensino de programação deveriam ser considerados para a construção das *strings* a serem submetidas a cada uma das fontes de pesquisa.

Depois de avaliados os termos, o processo de construção das strings consistiu em unir as palavras chaves avaliadas como ***G*** por meio do conectivo lógico OR, para que a pesquisa considerasse todos os trabalhos relevantes acerca do ensino de programação. Por conseguinte, os termos "*cognitive psychology" e/ou "cognitive science"* foram adicionados à *string* por meio do operador lógico AND, para filtrar, dentro desses estudos, as iniciativas relacionadas ao uso da psicologia cognitiva. As *strings* resultantes deste processo para cada uma das bases foram:

- **ACM** – ("teaching programming" OR "teach programming" OR "teaching to program" OR "teach to program" OR "learning programming" OR "learn programming" OR "learning to program" OR "learn to program" OR "learning to code") AND ("cognitive psychology");
- **IEEE** – ("teaching programming" OR "teach programming" OR "teaching to program" OR "teach coding" OR "learning programming" OR "learn programming" OR "learning to program" OR "learning to code") AND ("cognitive psychology");
- **ScienceDirect** – ("teaching programming" OR "teach programming" OR "teaching to program" OR "learning programming" OR "learn programming" OR "learning to program") AND ("cognitive psychology");
- **Springer** – ("teaching programming" OR "teach programming" OR "learning programming" OR "learn programming") AND ("cognitive psychology");
- **Scorpus** – ("teaching programming" OR "teach programming" OR "teaching coding" OR "teach coding" OR "learning programming" OR "learn programming" OR "learning to program" OR "learn to program" OR "learning to code" OR "learn to code") AND ("cognitive psychology");
- **Compendex** – ("teaching programming" OR "teach programming" OR "teaching coding" OR "teach coding" OR "learning programming" OR "learn programming" OR "learning to program" OR "learn to program" OR "learning coding" OR "learn coding" OR "learning to code") AND ("cognitive psychology").

Processo de seleção dos estudos

O pesquisador executou a busca nas fontes selecionadas utilizando as *strings* de busca elaboradas. A partir dos critérios de inclusão e exclusão (Quadro) os artigos foram selecionados em três etapas: ***seleção inicial***, ***seleção refinada*** e **revisão aprofundada**. Estes artigos selecionados foram documentados no ***formulário de seleção de estudos*** (Quadro). As cópias de todos os artigos incluídos como resultados da pesquisa inicial foram revisados, inteiramente, por apenas um pesquisador.

CONDUÇÃO DO MAPEAMENTO SISTEMÁTICO

O mapeamento sistemático foi realizado com o apoio da ferramenta Mendeley, no período de maio a outubro de 2015. O Quadro 8 ilustra o resumo do processo.

Quadro 8 – Processo do mapeamento sistemático

ETAPA	ATIVIDADE
Aplicação das *Strings* nas bases	Catalogação dos resultados na ferramenta Mendeley.
Seleção inicial	Leitura de título, resumo e palavras chaves.
	Aplicação dos critérios de inclusão e exclusão.
Seleção refinada	Leitura dos estudos, introdução e conclusões.
	Aplicação dos critérios de inclusão e exclusão.
Revisão aprofundada	Leitura completa dos artigos.
	Sumarização e Conclusão.

Fonte: Autor desta dissertação (2016).

Processo de pesquisa

As *strings* especificas foram submetidas às suas respectivas bases de dados. Os artigos retornados foram catalogados na ferramenta Mendeley. O Quadro mostra o resumo da condução por repositório de pesquisa.

Quadro 9 – Resumo da condução por repositório de pesquisa

Base Eletrônica		ACM	IEEE	Science Direct	Springer	Scorpus	Compendex	**Total**
Busca inicial		121	31	145	45	88	2	**432**
Seleção inicial	Incluídos	34	14	31	4	25	2	**110**
	Excluídos	87	17	114	41	63	0	**322**
Seleção Refinada	Incluídos	15	4	17	4	6	1	**47**
	Excluídos	19	10	14	0	19	1	**63**

Fonte: Autor desta dissertação (2016).

Etapa de seleção inicial

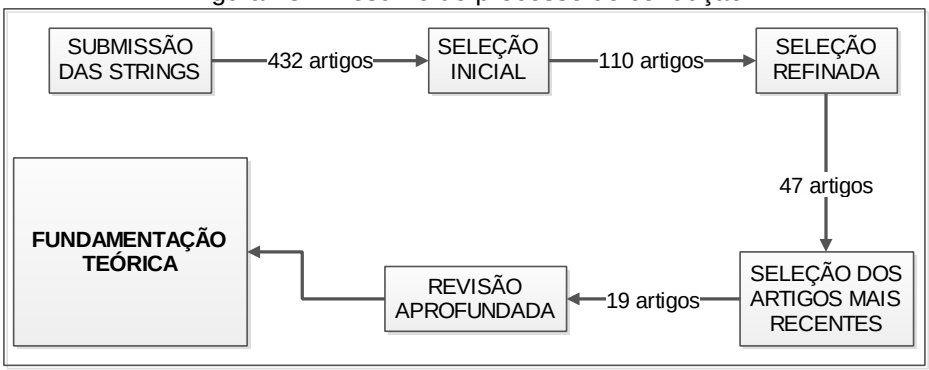
A submissão das *strings* nas bases retornou um total de 432 artigos. Estes foram submetidos ao processo de seleção inicial, onde foram analisados o título, o resumo, as palavras chaves e foram aplicados os critérios de inclusão CI-1 e CI-2, além dos critérios de exclusão CE-1 e CE-2. Este processo resultou na remoção de 322 artigos, restando 110 artigos a serem submetidos à etapa de seleção refinada.

Etapa de seleção refinada

Para o processo de seleção refinada foram lidas as introduções e conclusões dos 110 artigos resultantes da seleção inicial. Foram aplicados os critérios de inclusão CI-3 e os de exclusão CE-3 e CE-4. Este processo resultou em 47 artigos que foram submetidos ao processo de sumarização.

Destes 47 artigos, 19 foram selecionados, com base no critério CI-4, para o processo de revisão aprofundada, que consistiu na leitura completa e extração das afirmações que responderam às questões de pesquisa e fundamentaram o modelo proposto por esta dissertação. A Figura 78 ilustra o mapa processual realizado durante a condução do mapeamento sistemático.

Figura 78 – Resumo do processo de condução



Fonte: Autor desta dissertação (2016).

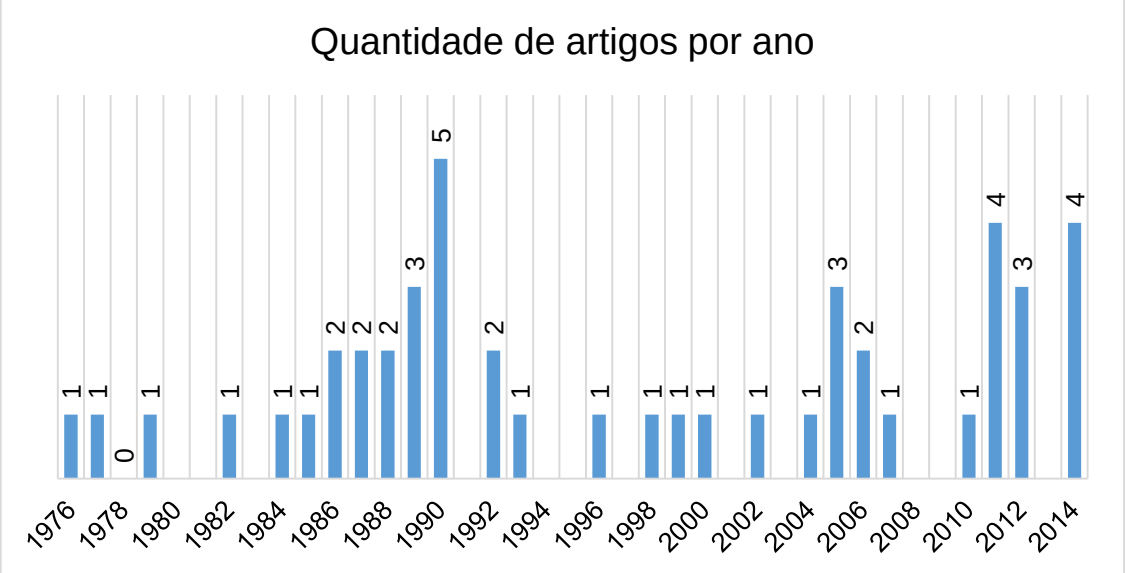
SUMARIZAÇÃO DO MAPEAMENTO

O mapeamento demonstrou que o estudo da psicologia da programação é um tema tratado desde a década de 1970, fato este que pôde ser observado através do trabalho de Shneiderman (1976), o mais antigo trabalho científico da área encontrado em todo o processo de mapeamento, no qual o autor realiza experimentos de memorização e lógica com estruturas da linguagem *Fortran,* baseando-se em conceitos da psicologia cognitiva. Outra evidência da importância desse tema foi encontrada em Storey (2005), que afirma que os desafios com o aprendizado de programação são conhecidos desde antes do primeiro *workshop* de engenharia de software realizado em 1968.

Análise da distribuição dos trabalhos

Esta fase teve por objetivo catalogar os dados identificados no mapeamento, gerando os indicadores quantitativos resultantes desse processo. A Figura apresenta todos os 47 artigos retornados pela seleção refinada e organizados por ano.

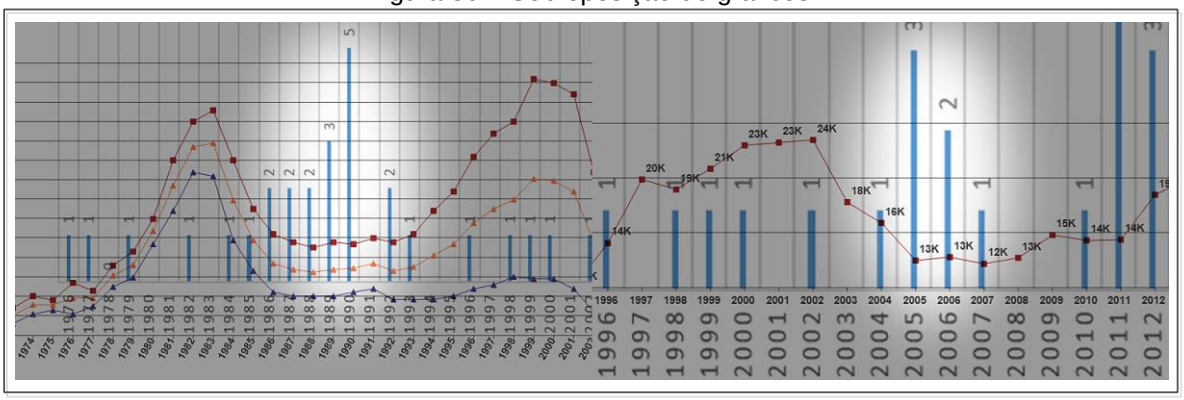
Figura 79 – Resumo da condução por repositório de pesquisa



Fonte: Autor desta dissertação (2016).

Quando analisamos a distribuição dos trabalhos ao longo dos anos (Figura), fica evidente uma grande concentração de publicações no início da década de 1990, em meados da década de 2000 e após 2010. Essas concentrações demonstraram claramente uma relação muito interessante com os dados identificados nas Figura e Figura , apresentadas no Capítulo 2, onde é possível observar que o número de publicações ao longo dos anos se demonstrou inversamente proporcional ao número de matrículas nos cursos de computação para os respectivos períodos de tempo. A Figura demonstra uma sobreposição dos referidos gráficos e comprova esse fato relatado.

Figura 80 – Sobreposição de gráficos.



Fonte: Autor desta dissertação (2016).

Apesar do fato de que o padrão observado não se aplicar à década de 2010, podemos hipotetizar que o aumento da publicação de trabalhos científicos nas referidas décadas foi motivado pelo que fora confirmado por Hernandez et al. (2010) descrito no capítulo 2. Seria uma resposta ao grande número de reprovações e desmotivação dos alunos para com os cursos de computação, oriunda das dificuldades enfrentada com as matérias de programação. Não podemos afirmar que este padrão se estende às demais áreas de pesquisa no ensino de programação de computadores, mas já denota uma forte preocupação e interesse com a problemática levantada nesta dissertação.

Análise dos trabalhos mais recentes

Os dezenove artigos mais recentes selecionados para uma leitura mais aprofundada são exibidos no Quadro.

Quadro 10 – Formulário de seleção de estudos.

AUTOR	TÍTULO
McKeown (2004)	The use of a multimedia lesson to increase novice programmers' understanding of programming array concepts.
Guibert, Guittet e Girard (2005)	A study of the efficiency of an alternative programming paradigm to teach the basics of programming.
Covington and Benegas (2005)	A cognitive-based approach for teaching programming to Computer Science and engineering students.
Lewandowsk et al. (2005)	What novice programmers don't know.
Lister et al. (2006)	Not seeing the forest for the trees.
Storey (2005)	Theories, tools and research methods in program comprehension: past, present and future.
Caspersen and Bennedsen (2007)	Instructional design of a programming course: a learning theoretic approach.
Renumol et al. (2010)	Identification of Cognitive Processes of Effective and Ineffective Students During Computer Programming.
Ambrosio et al. (2011)	Identifying cognitive abilities to improve CS1 outcome.
Ma et al. (2011)	Investigating and improving the models of programming concepts held by novice programmers.
Thompson (2011)	The nature of an object-oriented program: How do practitioners understand the nature of what they are creating?

AUTOR	TÍTULO
Parnin and Rugaber (2011)	Resumption strategies for interrupted programming tasks.
Cooper et al. (2012)	Initial results of using an intelligent tutoring system with Alice.
Dillon et al. (2012)	Comparing mental models of novice programmers when using visual and command line environments.
Kumar (2012)	The effect of interleaving an alternate task during tutoring and testing.
Abdul-Rahman and du Boulay (2014)	Learning programming via worked-examples: Relation of learning styles to cognitive load.
De Smet at al. (2014)	Taupe: Visualizing and analyzing eye-tracking data.
Morrison et al. (2014)	Measuring cognitive load in introductory CS.
Siegmund et al. (2014)	Understanding understanding source code with functional magnetic resonance imaging.

Fonte: Autor desta dissertação (2016).

Problemáticas identificadas

O Quadro apresenta um resumo das principais problemáticas identificadas pelos autores acerca das dificuldades com o ensino de programação.

Quadro 11 – Principais problemáticas identificadas acerca da programação

PROBLEMA IDENTIFICADO	AUTOR
Má preparação dos alunos no ensino médio.	(COVINGTON ; BENEGAS, 2005)
Falta de maturidade do ensino de programação.	
Má formulação dos livros didáticos de programação, que são orientados à sintaxe.	
Equivocadas motivações do aluno para ingressar na universidade.	
As modernas IDEs omitem importantes conceitos quando simplificam etapas.	(DILLON et al., 2012)
A grande quantidade de Processos Cognitivos (CPs) envolvidos na tarefa da programação a torna complexa.	(RENUMOL et al., 2010)
As matérias introdutórias de programação são responsáveis pelas altas taxas de desistência.	(AMBROSIO et al., 2011)
Mesmo os alunos que entendam os problemas e suas soluções, eles sentem dificuldades para expressá-los computacionalmente.	
A abordagem do ensino de programação tem sido a mesma nos últimos 40 anos.	(COOPER et al., 2012)
Os alunos não conseguem escrever programas razoáveis após dois semestres com matérias de programação.	(CASPERSEN ; BENNEDSEN, 2007)

PROBLEMA IDENTIFICADO	AUTOR
Os livros didáticos abordam apenas processos de programação sem a devida contextualização do problema.	
A retenção do conhecimento é prejudicada quando há sobrecarga da **working memory**. Quando o processamento requerido pela atividade excede a capacidade limitada da mente.	(MORRISON et al., 2014)
As matérias introdutórias de programação representam o início de um processo demorado e pouco compreendido.	(LISTER et al., 2006)
Os programadores iniciantes formam representações de conhecimento insuficientes, baseadas apenas na funcionalidade do código.	
Os alunos iniciantes não fazem as devidas implicações das estruturas de programação com o restante do código.	
As interrupções das atividades de programação aumentam a carga de trabalho mental.	(PARNIN ; RUGABER, 2011)
A maior parte dos programadores demoram muito tempo para retomar suas atividades de programação	
O conhecimento dos programadores iniciantes é uma mistura de estruturas conceituais bem formadas, outras incompletas e outras confusas.	(LEWANDOWSKI et al., 2005)
Os desafios com o aprendizado de programação são conhecidos desde antes do primeiro *workshop* de engenharia de software, em 1968.	(STOREY, 2005)

Fonte: Autor desta dissertação

Com relação às principais problemáticas identificadas para o ensino de programação, foi constatado que há falta de maturidade em relação à abordagem utilizada nas universidades, as quais não condizem com as expectativas dos alunos iniciantes. Neste contexto, alguns autores relatam a falta de preparo dos alunos e outros fazem críticas às metodologias baseadas no ensino de sintaxe.

Lewandowski et al. (2005) afirmam que o conhecimento dos iniciantes é incompleto e confuso. Os alunos não possuem uma base de conhecimentos bem estabelecida para aprender a programar.

Caspersen e Bennedsen (2007) relatam sobre a tendência de mudar a abordagem de ensino de detalhes sintáticos para o desenvolvimento de habilidades de resolução de problemas genéricos, abordagem essa denominada pattern-based approach. Os autores relatam que tal proposta vem sendo motivada pelo fato de que os 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.

Covington e Benegas (2005) propõem uma mudança da abordagem baseada em sintaxe, definida por eles como *syntax-driven*, para uma abordagem de ensino denominada *schema-driven*, que se propõe a desenvolver no aluno a capacidade de reconhecer quando um determinado tipo de problema pertence a um determinado padrão de problema bem conhecido, e que resolve uma classe inteira de problemas similares.

Os autores relatam que o ensino de sintaxe deve ser realizado apenas quando o aluno tiver desenvolvido os modelos mentais acerca de como aplicar os conceitos aprendidos na resolução de problemas reais. Eles ainda afirmam que essa abordagem está totalmente embasada nos resultados de pesquisas bem estabelecidas da psicologia cognitiva que relatam sobre como os estudantes aprendem de forma mais eficiente.

Quando se analisa as afirmações de Covington and Benegas (2005), que relatam que os livros didáticos de programação são orientados à sintaxe, e as afirmações de Cooper et al. (2012), que a abordagem do ensino de programação tem sido a mesma nos últimos quarenta anos, compreende-se o que dizem Ambrosio et al. (2011), que afirmam que as matérias introdutórias de programação são responsáveis pelas altas taxas de desistência nos cursos de computação. Ou seja, entende-se que os alunos desistem pois não encontram nas universidades uma metodologia de ensino que leve em consideração seus níveis de conhecimentos.

Portanto, entende-se que na perspectiva dos autores, um modelo de ensino baseado na psicologia cognitiva deve focar não em aspectos da sintaxe, mas em mecanismos que ilustrem o correto entendimento do problema.

Conceitos identificados

O Quadro apresenta os principais conceitos levantados pelo mapeamento e que representam como a psicologia cognitiva é abordada no ensino de programação de computadores.

Quadro 12 – Principais conceitos identificados acerca da psicologia cognitiva

CONCEITO	DESCRIÇÃO	AUTORES QUE ABORDAM
Cognitive load theory	Teoria que descreve o modelo de funcionamento da memória, onde a mesma tem sua capacidade limitada à carga cognitiva aplicada sobre ela.	(MORRISON et al., 2014), (ABDUL-RAHMAN ; DU BOULAY, 2014), (CASPERSEN ; BENNEDSEN, 2007), (COOPER et al., 2012), (COVINGTON ; BENEGAS, 2005), (MCKEOWN, 2004)
Long-term memory	Sistema inerte, organizado e teoricamente ilimitado da mente humana, responsável pelo armazenamento permanente das informações.	(CASPERSEN ; BENNEDSEN, 2007), (COVINGTON ; BENEGAS, 2005), (MCKEOWN, 2004)
Working memory	Sistema ativo da mente humana, responsável pelo armazenamento temporário das informações e todo processamento cognitivo consciente durante as atividades de raciocínio, compreensão e aprendizado.	(SIEGMUND et al., 2014), (MORRISON et al., 2014), (ABDUL-RAHMAN ; DU BOULAY, 2014), (CASPERSEN ; BENNEDSEN, 2007), (MCKEOWN, 2004)
Short-term memory	Sistema de armazenamento cerebral limitado e temporário pertencente à **working memory**.	(COVINGTON ; BENEGAS, 2005)
Cognitive Load	Carga cognitiva imposta à **working memory** do indivíduo durante o processo de resolução de problemas, pensamento e raciocínio.	(SIEGMUND et al., 2014), (MORRISON et al., 2014), (ABDUL-RAHMAN ; DU BOULAY, 2014), (CASPERSEN ; BENNEDSEN, 2007), (COOPER et al., 2012), (RENUMOL et al., 2010), (MCKEOWN, 2004)
Instrinsic, Extraneous e Germane load. (IL, EL e GL)	Tipos de **cognitive loads** descritas na **Cognitive Load Theory**.	(MORRISON et al., 2014), (ABDUL-RAHMAN ; DU BOULAY, 2014), (CASPERSEN ; BENNEDSEN, 2007)

CONCEITO	DESCRIÇÃO	AUTORES QUE ABORDAM
Schemas	Construções mentais ou modelos conceituais, que representam o conhecimento e possuem a capacidade de tratar grandes quantidades de informações como se fossem apenas uma.	(ABDUL-RAHMAN AND DU BOULAY, 2014), (CASPERSEN E BENNEDSEN, 2007), (COVINGTON AND BENEGAS, 2005)
Schema acquisition	Modelo conceitual do processo de aprendizado, onde os **schemas** são decodificados a partir da **long-term memory**, transferidos para a **working memory**, incorporados à novas informações e transferidos de volta ao sistema inerte.	(CASPERSEN E BENNEDSEN, 2007)
Chunk	Identificado na literatura como sinônimo de **schema** e utilizado para referi-lo quando o mesmo se encontra presente na **working memory**.	(STOREY, 2005), (SIEGMUND ., 2014), (CASPERSEN ; BENNEDSEN, 2007), (COVINGTON ; BENEGAS, 2005)
Chunking	Processo de construção de **schemas** através da recordação de informações pré-existentes, relacionamento e posterior união das mesmas, criando **schemas** mais complexos.	(STOREY, 2005), (CASPERSEN ; BENNEDSEN, 2007), (COVINGTON ; BENEGAS, 2005)
Effects	Técnicas desenvolvidas pela **Cognitve Load Theory** para minimizar o **EL** e maximizar o **GL**.	(CASPERSEN; BENNEDSEN, 2007)
Worked-examples	Estratégia pedagógica, definida na **Cognitive Load Theory**, que defende o uso de exemplos como forma de melhorar o **schema acquisition**.	(ABDUL-RAHMAN ; DU BOULAY, 2014), (CASPERSEN ; BENNEDSEN, 2007)
Programming plans	Fragmentos de código de programação estereotipados e que representam soluções bem conhecidas, recorrentes a diversos tipos de problemas.	(STOREY, 2005), (ABDUL-RAHMAN ; DU BOULAY, 2014), (COVINGTON ; BENEGAS, 2005)
Cognitive processes	Processos mentais, relacionados à regiões cerebrais, responsáveis pelos processos de geração do conhecimento durante a compreensão.	(SIEGMUND et al., 2014), (RENUMOL et al., 2010)
Mental model	Representação cognitiva organizada da realidade externa, criada pelo programador durante o processo de entendimento do código. Identificado como sinônimo de **Schema**.	(STOREY, 2005), (AMBROSIO et al., 2011), (GUIBERT; GUITTET; GIRARD, 2005), (MCKEOWN, 2004)
Top-down comprehension	Processo de compreensão que se faz presente quando o programador possui familiaridade com o conteúdo estudado, onde conhecimentos de mais alto nível são reconhecidos, fazendo com que o problema seja desmembrado em partes menores.	(STOREY, 2005), (SIEGMUND et al., 2014)

CONCEITO	DESCRIÇÃO	AUTORES QUE ABORDAM
Bottom-up comprehension	Processo de compreensão que se faz presente quando o programador não possui familiaridade com conceitos presentes no código, gerando a necessidade de um esforço cognitivo para a agregação de conceitos de mais baixo nível, a fim de gerar entendimentos de mais alto nível.	(STOREY, 2005), (SIEGMUND et al., 2014)
Meaningful learning	Aquisição de conhecimento por meio da associação de uma nova informação à um **schema** já existente.	(COVINGTON ; BENEGAS, 2005)
Rote learning	Aquisição do conhecimento por meio da memorização, onde as informações não são associadas a nenhum **schema** pré-existente durante o aprendizado.	(COVINGTON ; BENEGAS, 2005)
Cognitive apprenticeship	Teoria do processo de ensino, onde especialistas ensinam os processos de resolução de problemas complexos para os aprendizes por meio da experiência guiada.	(CASPERSEN ; BENNEDSEN, 2007)
Adaptive Control of Thought-Relational (ACT-R)	Teoria do processamento humano de informações e representação do conhecimento que descreve uma arquitetura cognitiva baseada em experimentos da psicologia.	(COOPER et al., 2012)
7 plus or minus 2	Teoria que define a limitação da **short-term memory** entre 5 e 9 **chunks (schemas)**.	(COVINGTON ;BENEGAS, 2005), (CASPERSEN; BENNEDSEN, 2007), (MCKEOWN, 2004)

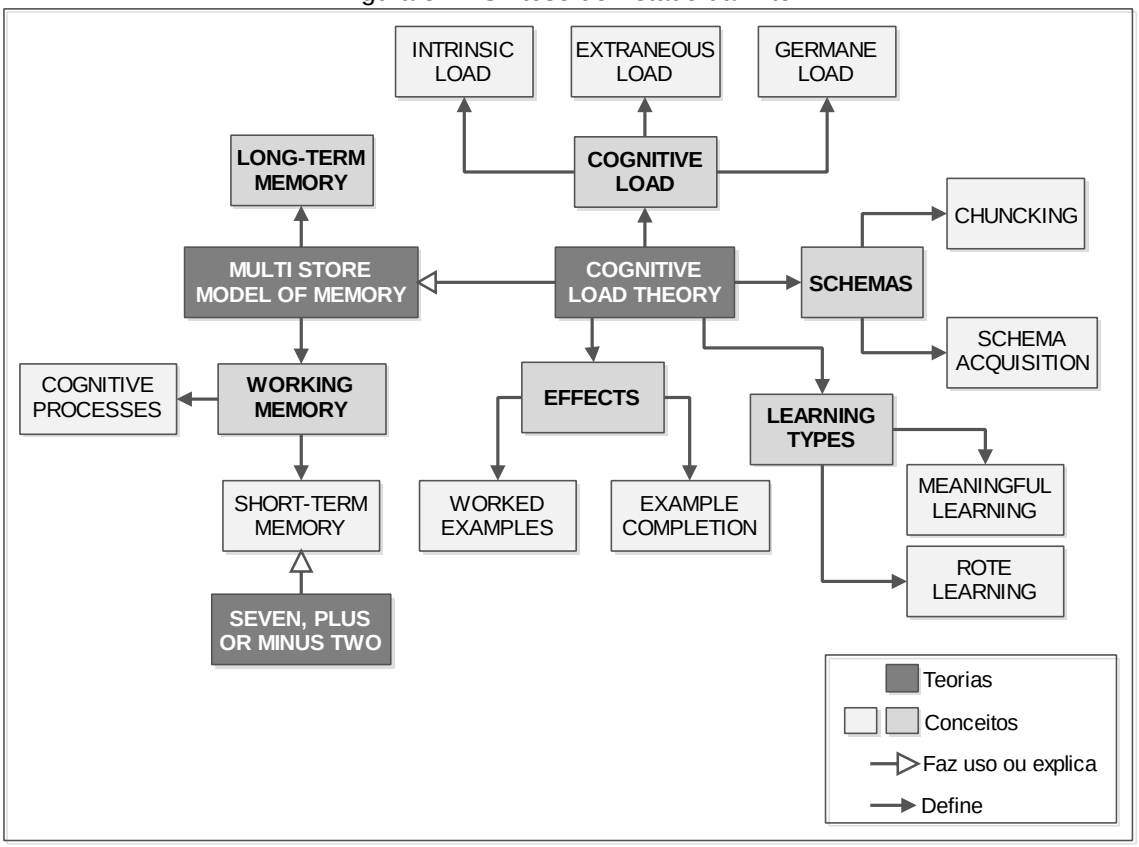
Fonte: Autor desta dissertação (2016).

Com base nos principais conceitos identificados no Quadro, pôde-se resumir todo o conhecimento adquirido através da Figura . Ela descreve os principais conceitos e teorias abordadas e discutidas nos artigos científicos, bem como a forma que eles estão relacionados uns com os outros. A figura representa a síntese do estado da arte, identificada nesse trabalho, de como a psicologia cognitiva é aplicada ao ensino da programação.

A leitura completa dos artigos mais recentes demonstrou que a teoria da carga cognitiva ou *Cognitive Load Theory* - CLT se apresentou como tópico de maior destaque, sendo o conceito mais recorrente nos estudos investigados. Foi possível perceber que a maioria das fundamentações dos trabalhos científicos estudados estavam associados diretamente ou indiretamente à CLT.

Foi possível também perceber que a CLT busca formas de melhorar o aprendizado por meio do estudo do funcionamento da mente humana, seja encontrando formas eficientes de acessá-la ou reduzindo o esforço mental despendido durante o processo de aprendizado.

Figura 81 – Síntese do Estado da Arte.



Fonte: Autor desta dissertação (2016).

As explicações detalhadas destes conceitos (Figura), bem como as citações dos autores e como eles os definem, encontram-se no Anexo B desta dissertação.

CONCLUSÕES

Considerando que as matérias introdutórias de programação representam o início de um processo demorado e pouco compreendido (LISTER et al., 2006), e que o aprendizado de programação requer grandes quantidades de processos cognitivos (Renumol et al., 2010), fica clara a importância de se propor novas metodologias baseadas em conceitos da psicologia cognitiva, com o intuito de investigar e propor

melhores formas de ensinar, levando em consideração o funcionamento e as limitações da mente dos alunos.

Neste contexto, a investigação apontou a CLT como sendo o assunto mais abordado direta e indiretamente nos trabalhos científicos recentes que tratam do tema *psicologia da programação*. Foi possível observar que os autores usam a CLT para propor alternativas à metodologia de ensino adotada nas matérias introdutória de programação de computadores.

A investigação demonstrou que os principais objetivos da CLT são:

1. Encontrar formas eficientes de acessar a memória LTM.
2. Encontrar formas de reduzir a carga cognitiva. Liberando mais espaço na memória para a construção de melhores modelos mentais.

Como resposta ao primeiro objetivo, foi possível perceber que a forma mais eficiente de acessar a memória LTM é utilizar uma abordagem baseada em *Schemas (modelos mentais)*, que evite qualquer tipo de memorização (*rote memorization*) e permita que as novas informações absorvidas possam ser associadas de forma imediata aos *Schemas* já existentes. Para que isso aconteça se faz necessária a organização restrita do conteúdo em acordo com a precedência de complexidade, para que as informações possam ser construídas na mente do aluno de forma organizada, fornecendo os corretos S*chemas* durante o processo de aprendizado. Isso permite que a mente recupere a informação de maneira mais eficiente durante o aprendizado de assuntos futuros e mais complexos.

A solução para o segundo objetivo da CLT é possível através da redução da complexidade do conteúdo instrucional, com o intuito de simplificar sua forma de apresentação e liberar capacidade cognitiva para que a construção dos modelos mentais se faça presente (Figura). Isso pode ser implementado por meio de modelos instrucionais que imponham o mínimo de dificuldades e informações desnecessárias no processo de explicação do conteúdo.

Resposta às questões de pesquisa

Retomando as três questões de pesquisa definidas no início deste capítulo:

1. Como a psicologia cognitiva é aplicada no ensino de conceitos de programação?
2. Quais são os principais tópicos da psicologia cognitiva aplicados no ensino de conceitos de programação?
3. Quais as principais dificuldades identificadas para o aprendizado de programação.

Resposta para a questão 01): Pode-se concluir que a psicologia cognitiva é aplicada no ensino de conceitos de programação de computadores através das diretrizes definidas na Teoria da Carga Cognitiva.

Resposta para a questão 02): Os principais tópicos da psicologia cognitiva aplicados no ensino de programação de computadores que foram identificados são:

- **Multi Store Model of Memory** – o modelo que define a mente humana como um conjunto de sistemas de armazenamento de informações de longo e curto prazo (Figura).
- **Cognitive Load** – tópico que define os impactos dos esforços cognitivos realizados pela mente durante o processo de aprendizado (Figura).
- **Schemas** – conceito que aborda a formação de blocos de informações que são utilizados pela memória como mecanismo para um aprendizado mais eficiente (Figura).

Resposta para a questão 03): As principais dificuldades enfrentadas para o aprendizado de programação são:

- A própria complexidade do assunto, que querer uma grande quantidade de tempo e esforço cognitivo para ser aprendido.
- A não aquisição de conhecimento básicos no ensino médio, por parte dos alunos iniciantes.
- A falta de maturidade do ensino de programação, que utiliza a mesma metodologia, baseada me sintaxe, há décadas.

No próximo capítulo, são utilizadas as afirmações e fundamentações discutidas aqui para a especificação do modelo lúdico proposto.

APÊNDICE B – CONCEITOS DA PSICOLOGIA COGNITIVA

Neste apêndice encontram-se as explicações detalhadas dos principais conceitos da psicologia cognitiva, identificados no mapeamento sistemático realizado no Capítulo 4 (Quadro). Os conceitos foram abordados em ordem de complexidade, visando o melhor entendimento ao passo em que eles são apresentados.

SEVEN, PLUS OR MINUS TWO

Como já foi abordado no Capítulo 3, a teoria no número mágico (*seven, plus or minus two*) de Miller (1956) afirma que o número máximo de elementos, denominados *chunks*, que ficam retidos na memória de curto prazo da mente humana é de “sete mais ou menos dois”, e que o processo de recordação da informação pode ser mais eficiente por meio da criação de elementos (*chuncks*) mais complexos e significativos. O detalhamento dessa teoria não será realizado aqui, sua explicação é abordada na Seção 0.

No mapeamento sistemático foi possível perceber que a teoria de Miller foi a base argumentativa para a teoria da carga cognitiva (Cognitive Load Theory - CLT), fato este que pôde ser reiterado por meio das afirmações de diversos autores.

Covington e Benegas (2005) relatam que, de acordo com a CLT, a memória humana de curto prazo (Short-Term Memory - STM) é limitada a armazenar apenas alguns itens por vez, e que isso representa a restrição fundamental da capacidade humana de aprendizagem, muitas vezes denominada como regra de “7 mais ou menos 2”.

Caspersen e Bennedsen. (2007, p.113) citam a teoria de Miller quando explicam sobre a arquitetura cognitiva humana: “[...] Miller descobriu que a working memory tem uma capacidade de cerca de ‘sete mais ou menos dois’ pedaços - independente do número de informações por pedaço.”

McKeown (2004, p.42) cita o trabalho de Miller em sua revisão de literatura:

> A maior limitação da working memory é a sua capacidade de lidar apenas com, mais ou menos, sete elementos de informação de maneira simultânea (MILLER, 1956). Caso este risco seja excedido, há o risco de perder toda ou parte da informação.

COGNITIVE LOAD THEORY

A identificação da CLT como o conceito mais relevante no estado da arte do uso da psicologia no ensino de programação de computadores surgiu por meio da reunião dos fragmentos de fundamentações identificados em cada um dos dezenove artigos científicos lidos.

A CLT descreve um *framework* genérico do funcionamento da mente humana. Seu criador John Sweller (SWELLER et al., 1998), com base nos trabalhos de Miller (1956), buscava desenvolver um modelo instrucional que reduzisse o esforço cognitivo dos alunos nas atividades de resolução de problemas. Sua principal justificativa era que a estratégia utilizada pelos alunos para resolver problemas, denominada *means-ends analisys*, era ineficiente do ponto de vista pedagógico, pois demandava grandes quantidades de processamento cognitivo, os quais, em sua maioria, estavam desassociados à construção dos modelos mentais referentes ao conteúdo de interesse.

Caspersen e Bennedsen (2007) definem a CLT como um conjunto de princípios de aprendizagem que lidam com a utilização eficiente da memória. Abdul-Rahman e du Boulay (2014) formalizam três princípios da CLT:

1. Encontrar formas de gerenciar a complexidade inerente ao que está sendo ensinado.
2. Simplificar a forma com a qual o conteúdo é apresentado, garantindo que outras informações não atrapalhem o percurso do aprendizado.
3. Encorajar os alunos a devotar seus recursos cognitivos às atividades em questão.

Diversos autores abordam a CLT no ensino de computação:

(i) Morrison et al. (2014) relatam um experimento de medição da carga cognitiva na área de introdução à Ciências da Computação.

(ii) Caspersen e Bennedsen (2007) usam a CLT para argumentar como um curso introdutório de programação orientada a objetos pode ser projetado.

(iii) Covington e Benegas (2005) usam as fundamentações da psicologia cognitiva para fazer uma crítica à abordagem de ensino de programação atual orientada a sintaxe e sugerem uma nova abordagem denominada *schema-driven*, que tem por objetivo desenvolver nos alunos a capacidade de resolver problemas.

(iv) Abdul-Rahman e du Boulay (2014) usam técnicas definidas na CLT para realizar um experimento científico com 117 alunos com o intuito de investigar as diferenças dos esforços mentais dos diferentes estilos de aprendizagem.

(v) Cooper et al. (2012) descrevem o uso de uma técnica denominada *Stencil* para a criação de um sistema tutor inteligente, incorporado no software Alice, com o objetivo de substituir o papel do professor no ensino de conceitos de programação.

Nem todos os artigos se fundamentavam especificamente na CLT, tais como:

(i) Siegmund et al. (2014), que realizaram um experimento controlado com 17 participantes dentro de um *scanner* fMRI, onde foram analisados padrões de ativações em regiões cerebrais dos participantes durante o processo de compreensão de trechos de código de programação. O experimento não cita explicitamente em suas fundamentações a Teoria da Carga Cognitiva, mas relata conceitos muito próximos, tais como *Cognitive Load*, *Working memory*, *Chunk*, *Cognitive processes*, *Top-down comprehension* e *Bottom-up comprehension*, os quais são também encontrados em outros artigos do mapeamento.

(ii) Renumol et al. (2010) realizaram um estudo com dezenove estudantes, onde buscaram identificar os processos cognitivos durante a programação, utilizando a técnica de Verbal Protocol Analysis (VPA). Apesar de não

fazerem referência direta à CLT, os autores também usam termos como *cognitive load, long-term memory e short-term memory.*

(iii) Lister et al. (2006) relataram o uso de uma taxonomia pedagógica denominada *SOLO*, mas não apresentaram nenhuma relação com a CLT.

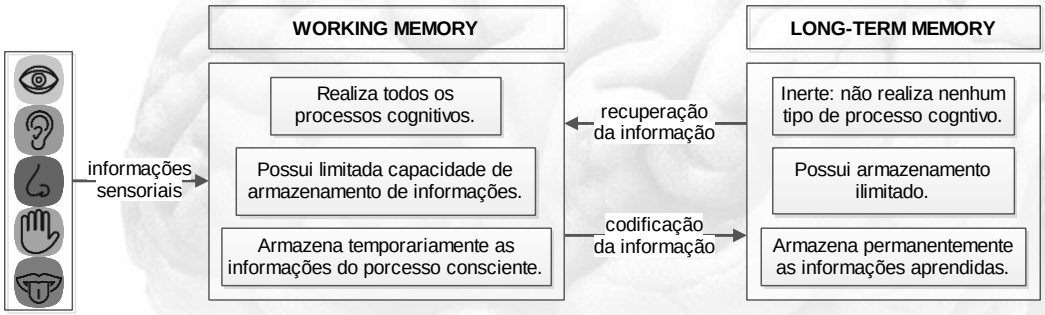
(iv) De Smet et al. (2014) relatam o desenvolvimento de uma ferramenta de código aberto e que utiliza a tecnologia *eye-tracking*, desenvolvida para que pesquisadores analisem dados sobre os processos cognitivos dos desenvolvedores durante a compreensão de programação. Os autores também realizam um estudo de caso, no qual a ferramenta fora utilizada em três experimentos científicos. Apesar de fazer muito uso da palavra *cogntive processes* e *cognitive psychology*, os autores não citam nenhum dos termos comuns identificados nos estudos da psicologia cognitiva, nem citam os principais autores e teorias.

Durante o processo de mapeamento foi possível perceber a repetição de determinados conceitos, os quais, com o tempo, foram melhor compreendidos e conceituados, permitindo a caracterização da CLT e a sua consequente identificação como conceito chave.

WORKING MEMORY E LONG-TERM MEMORY

Este tópico trata dos conceitos que definem o modelo humano de armazenamento de informações referidos no *Multi Store Model of Memory*, e proposto por Atkinson e Shiffrin (1968). Tal modelo, exposto na Seção 0, encontra-se dividido em dois principais sistemas distintos e complementares: *working memory* e *long-term memory* - LTM. A *working memory* é o sistema responsável pelos processos de raciocínio, pela compreensão, pelo aprendizado e pelo armazenamento temporário das informações recém adquiridas e das já aprendidas. A LTM é o sistema de armazenamento permanente e estático da mente humana, onde as informações ficam armazenadas por longos períodos de tempo. A Figura ilustra estes dois sistemas.

Figura 82 – Modelo de memória da mente humana.



Fonte: Autor desta dissertação (2016).

O ponto de encontro entre a CLT e o conceito da *working memory* pôde ser identificado por meio da afirmação de Morrison et al. (2014), onde os autores relatam que, segundo a Teoria da Carga Cognitiva, o conhecimento é prejudicado quando a quantidade de processamento requerido pela atividade excede a limitada capacidade da *working memory*. Eles afirmam que a limitação da *working memory*, identificada por Miller (1956), representa um grande desafio a ser considerado e resolvido pelos modelos instrucionais.

Caspersen e Bennedsen (2007) afirmam que toda atividade humana que envolve trabalho mental é baseada em dois sistemas de memória: a *work-memory* e a LTM. Eles afirmam que a LTM representa o sistema de armazenamento de grande capacidade, porém é um membro inerte do sistema, não sendo responsável por nenhum tipo de processo consciente de informação.

Sweller et al. (1998) afirmam que a limitação da capacidade da *work memory* restringe muitos processos cognitivos durante o aprendizado.

Covington e Benegas (2005) afirmam que as limitações da *working memory* são compensadas pela capacidade da LTM e, apesar de cada tipo de memória ter seus

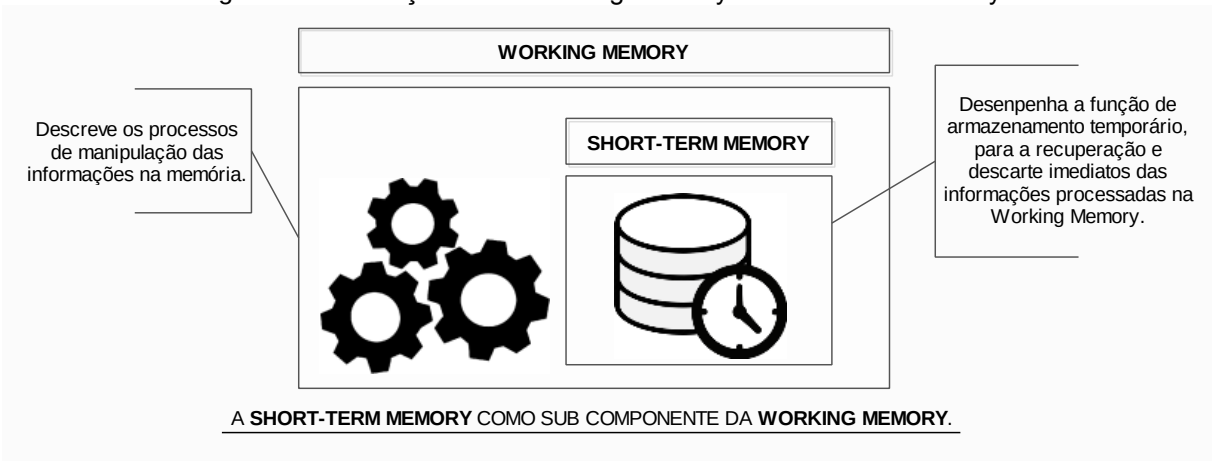
próprios mecanismos de aprendizagem, todos os alunos as usam de forma bem característica.

Foi identificada na bibliografia uma ambiguidade entre os termos *working memory* e *Short-Term Memory* - STM. Covington e Benegas (2005) ora usam um, ora usam outro para se referir ao sistema limitado de informações da mente. [...] a teoria da carga cognitiva afirma que a **short-term memory** é limitada a armazenar apenas alguns poucos itens por vez [...] A limitação da **working memory** é compensada pelas capacidades da long-term memory.

Apesar de ambos os termos serem utilizados pelo autor para se referir ao mesmo sistema de armazenamento da mente, foi possível distingui-los a partir das definições de Atkinson e Shiffrin (1968), eles afirmam: “[...] O segundo componente fundamental do nosso sistema é a short-term memory. Este armazenamento pode ser considerado como a working memory do indivíduo.”

Contudo, ficou clara a distinção entre ambos os sistemas. STM se refere apenas ao sistema de armazenamento volátil da mente, enquanto que a *working memory* se refere ao sistema mais complexo, que contém o primeiro e realiza todos os processos cognitivos conscientes. Essa distinção é ilustrada na Figura.

Figura 83 – Distinção entre “working memory” e “short-term memory”.



Fonte: Autor desta dissertação (2016).

De acordo com os trabalhos relatados nesta seção, é possível concluir que o modelo descrito na CLT descreve a mente humana como um sistema composto por dois tipos de memória diferentes e que complementam suas deficiências:

- Working memory. Sistema responsável por todos os processos cognitivos da mente consciente e que possui um componente de armazenamento volátil de informações denominado STM.
- Long-Term memory. Sistema de memória subconsciente, inerte e permanente, que armazena o conhecimento aprendido e os disponibiliza para a *working memory* durante os processos cognitivos.

COGNITIVE LOAD

O termo *cognitive load*, ou carga cognitiva, é a parte da CLT, descrita por John Sweller, que se refere ao esforço cognitivo despendido pela mente durante os processos de aprendizado e que interfere de forma direta na eficiência e eficácia da aquisição do conhecimento.

Pode-se entender a *cognitive load* como o esforço realizado pela *working memory* durante os processos cognitivos da mente. Neste mapeamento sistemático identificou-se a medição e a redução da carga cognitiva como os principais objetivos das abordagens analisadas.

Morrison et al. (2014) afirmam que a medição de carga cognitiva começou a ser estudada quando os pesquisadores percebem como as atividades de solução de problemas interferiam no aprendizado. Os autores relatam que a *working memory* era forçada a realizar tanto a tarefa de solução de problemas como o processo de aprendizado, portanto a redução da carga cognitiva se tornou uma importante meta instrucional.

Siegmund et al. (2014) afirmam que, se as dificuldades e os esforços cognitivos puderem ser medidos e correlacionados às atividades de programação, então os pesquisadores podem identificar e quantificar os trechos de código, tipos específicos

de problemas algorítmicos ou propor novas linguagens e/ou ferramentas para a programação.

Foi observado nos estudos do mapeamento que a carga cognitiva se divide em três tipos: *Intrinsic Load* - IL ou carga instrínseca, *Extraneous Load* – EL ou carga alheia, *Germane Load* - GL ou carga pertinente:

> [...] A Intrinsic load é gerada pela complexidade inerente do material a ser aprendido em termos do número dos seus elementos e de suas interações. Extraneous load é gerada pela forma e meios através dos quais o material é experimentado. Por exemplo, um material fácil pode ser apresentado de maneira complexa, e vice-versa. De acordo com Sweller (2010), a Germane load é de uma categoria distinta e por isso difere da Intrinsic e Extraneous load. […] (ABDUL-RAHMAN; DU BOULAY, 2014).

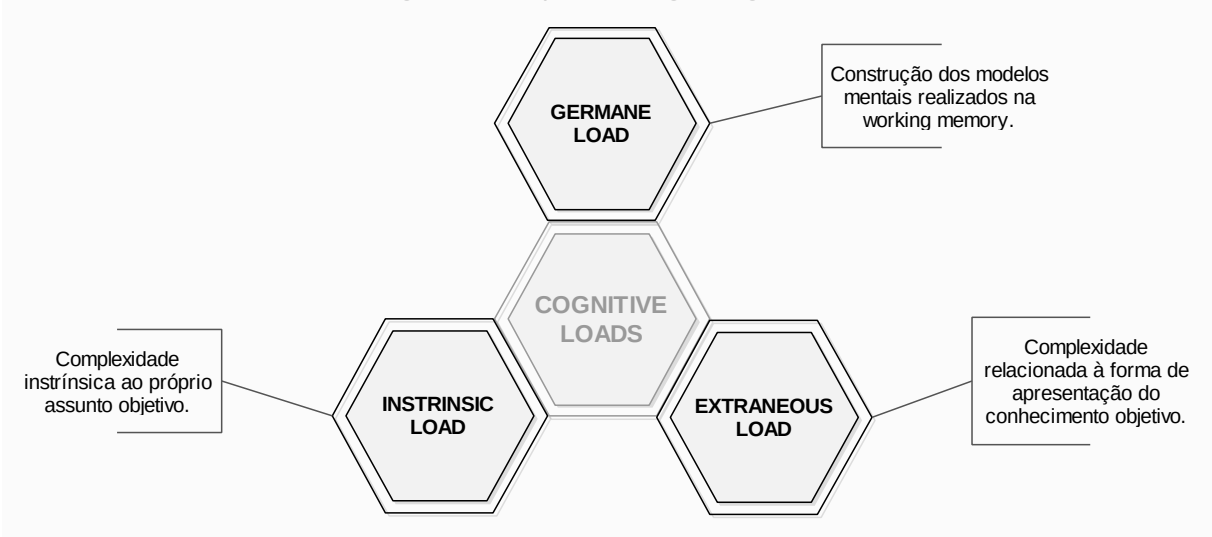
A carga cognitiva denominada IL está relacionada à complexidade do conhecimento objetivo. Refere-se ao esforço cognitivo relacionado ao assunto estudado e pode ser reduzida por meio da divisão das tarefas em etapas menores. Morrison et al. (2014) definem a IL como a dificuldade intrínseca ao próprio assunto. Devido à grande quantidade de interconexões do assunto com outros conhecimentos, se faz necessário processar de forma simultânea diversos elementos, os quais nem sempre são de domínio do estudante, a fim de entender suas relações.

A EL é um tipo de carga considerada negativa para o processo de aprendizado, pois se refere ao esforço cognitivo gerado pelas informações não relacionadas ao conhecimento objetivo. Representa a carga gerada pela forma de apresentação do material instrucional e pode ser reduzida através do uso de diagramas ou uso de exemplos visuais. Morrison et al. (2014, p.131) afirma que “a *Extraneous load* (EL) é a carga colocada na working memory que não contribui diretamente para o aprendizado do material.”

A GL é carga positiva para o processo de aprendizado, pois se refere ao esforço cognitivo gerado pela construção dos modelos mentais do conhecimento que acontecem na *working memory*. Morrison et al. (2014) definem GL como carga cognitiva gasta com as características extras que são necessárias ao processo de aprendizado.

Desta forma, fica clara a evidência de que esta classificação para as cargas cognitivas é definida pela CLT como forma de distinguir os esforços cognitivos durante o processo de aprendizado. A Figura ilustra o resumo dessa classificação.

Figura 84 – Tipos de cargas cognitivas



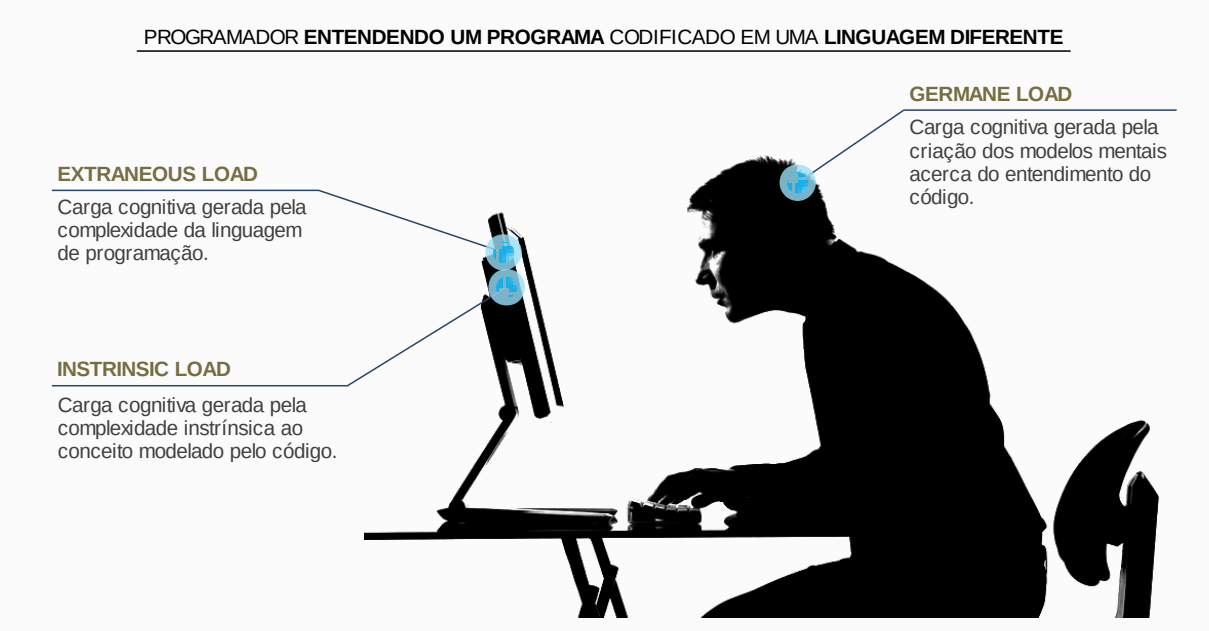
Fonte: Autor desta dissertação (2016).

Pode-se exemplificar o entendimento do conceito de cargas cognitivas da seguinte maneira: imagine um programador tentando compreender o funcionamento de um determinado software escrito em uma linguagem de programação diferente, na qual ele não possui profundos conhecimentos. Seu objetivo é entender o programa, o seu modelo de funcionamento, independente da forma ou linguagem com a qual ele foi escrito. O esforço devotado ao entendimento dos padrões de programação que formam o software representa a carga cognitiva IL, pois o mesmo se refere ao processo de entendimento objetivo.

Apesar de sua grande experiência com os padrões de programação, esse processo de entendimento de entendimento vai ser prejudicado pela complexidade imposta pela nova linguagem. Define-se linguagem. Define-se então o esforço devotado à tal complexidade como a carga EL, pois o mesmo pois o mesmo se refere aos aspectos de apresentação do conhecimento objetivo, neste caso, à neste caso, à linguagem de programação desconhecida. Por fim, define-se o esforço devotado à devotado à construção dos modelos mentais acerca do funcionamento do programa como a carga como a carga GL. Este exemplo é melhor ilustrado na

Figura.

Figura 85 – Cargas cognitivas durante o processo de aprendizado

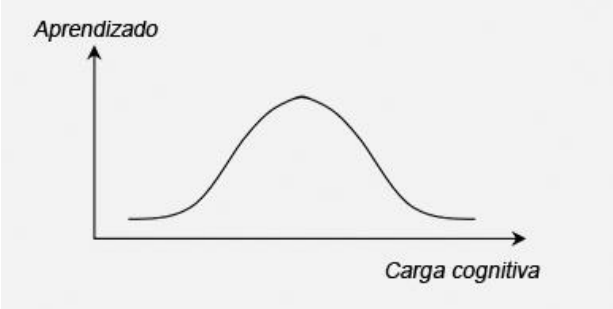


Fonte: Autor desta dissertação (2016).

Com relação à quantidade de carga cognitiva, Morrison et al. (2014) afirmam que o desempenho do aprendizado de um determinado conceito por parte do estudante está diretamente relacionado à quantidade de carga cognitiva que é usada durante a compreensão do estudo.

Caspersen e Bennedsen (2007) quantificam os resultados do aprendizado em função do uso dessas cargas, afirmando que o aprendizado é maximizado quando há um balanceamento das mesmas. Os autores exemplificam esta afirmação na Figura, onde é justificada que a falta de novas informações e atividades rotineiras não geram decodificação e integração de novos conhecimentos. E que o excesso de informações absorve a carga cognitiva que deveria ser utilizada para o processo de aprendizado (Germane Load).

Figura 86 – Aprendizado em função da carga cognitiva

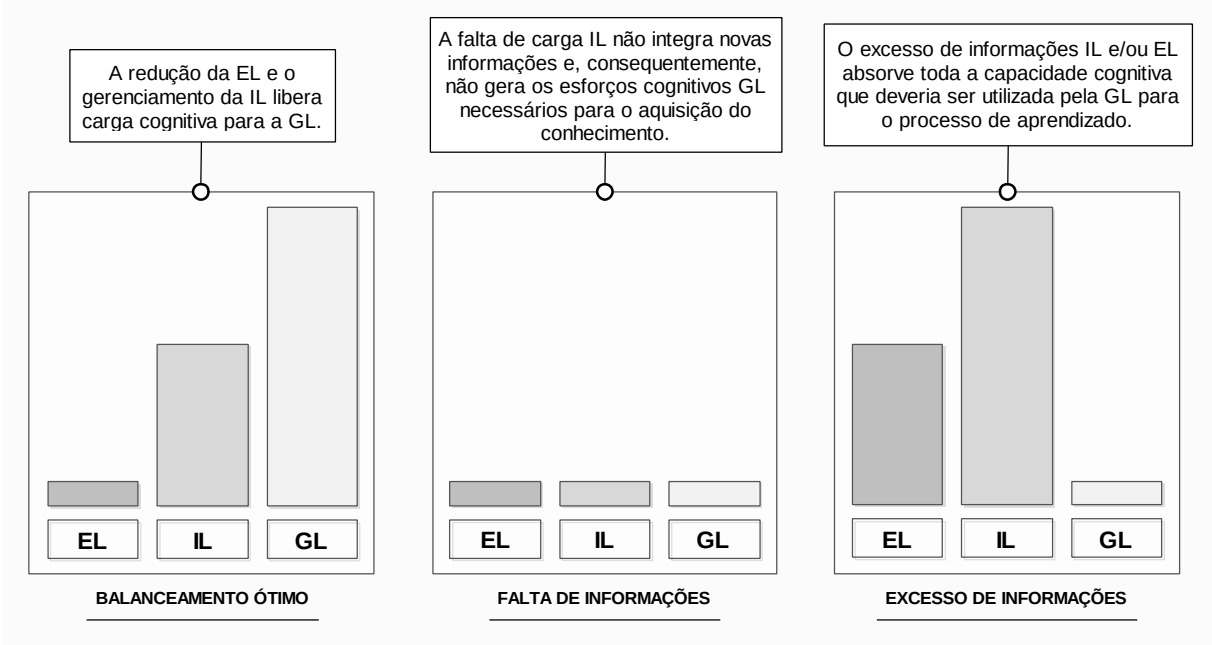


Fonte: Caspersen e Bennedsen (2007)

Os autores relatam que a maioria das pesquisas na área se concentram em identificar técnicas de redução de EL e aumento de GL em modelos instrucionais, e afirmam também que o desafio do balanceamento de carga cognitiva é o de minimizar o EL e maximizar o GL. Morrison et al. (2014) ainda afirmam que o objetivo instrucional é mitigar, além da EL, a IL, a fim de maximizar a quantidade de memória disponível para a GL.

As ideias e os objetivos dos autores com relação ao uso das cargas cognitivas são expostos na Figura.

Figura 87 – Estados do balanceamento das cargas cognitivas



Fonte: Autor desta dissertação (2016).

Independente das particularidades identificadas por cada trabalho, ficou claro que o objetivo da CLT é propor modelos instrucionais que proporcionem o gerenciamento das cargas cognitivas IL e EL, para que sobre a maior quantidade possível de capacidade cognitiva na *working memory* para que a GL se faça presente na criação dos modelos mentais necessários à aquisição do conhecimento.

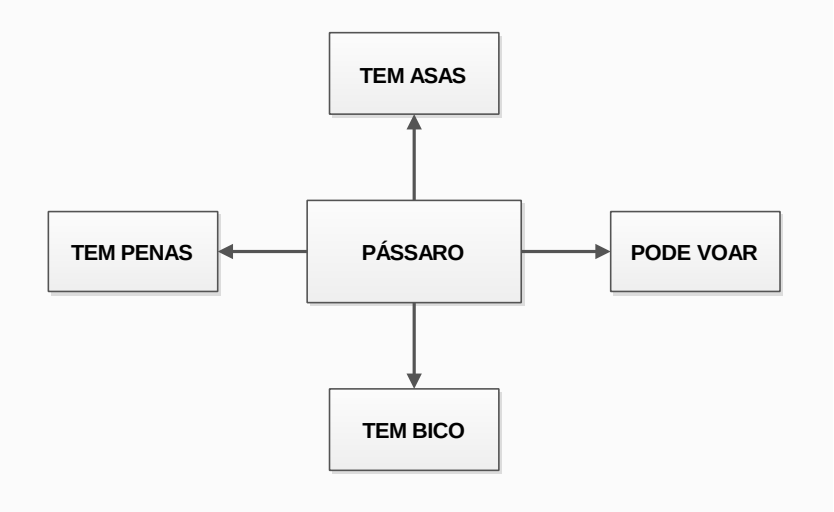
SCHEMAS

Os Schemas podem ser entendidos como um conjunto de informações distintas que possuem forte relação entre elas, e são tratadas pela memória com sendo apenas

uma informação. Pôde-se perceber que a sua definição está fortemente relacionada à teoria da formação de blocos citada por Miller (1956) e Collin (2012) e abordada no Capítulo 3 (Figura).

A Figura ilustra a definição do conceito de *Schema*, onde é possível observar que o conceito de pássaro está associado com outras definições relacionadas. Toda vez que o conceito de pássaro for trazido para a *working memory*, outras informações virão agregadas à ela, formando um conjunto de informações bem conceituadas, de tal forma que este conjunto será tratado na *working memory* como apenas um elemento (*chunk*).

Figura 88 – Diagrama básico de um schema para o conceito de pássaro



Fonte: Adaptado de McLeod (2009)

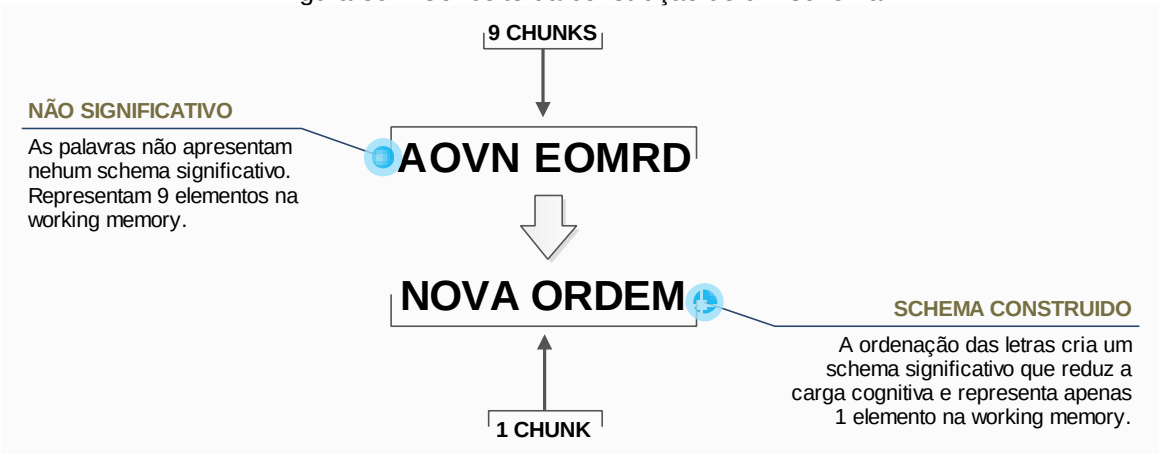
As pessoas usam S*chemas* com o intuito de simplificar estruturas de informação para o rápido raciocínio e para facilitar as futuras aquisições de conhecimento.

Caspersen e Bennedsen (2007, p.113) definem S*chemas* como sendo estruturas que nos permitem tratar um grande número de informações como se fosse apenas uma. Os autores também apresentam outros sinônimos utilizados para a definição do conceito tais como *chunks, plans, templates* e *idioms.*

Segundo Covington e Benegas (2005) o processo de aquisição de S*chemas*, que permite ao cérebro organizar as informações de uma maneira bem estruturada, é a forma que a LTM consegue compensar as limitações da *working memory*.

A Figura ilustra outro exemplo do conceito de *Schema*, no qual é possível observar como um conjunto de símbolos, representado por letras, pode ser agrupado e convertido em um conhecimento mais significativo, representado pela palavra “nova ordem”.

Figura 89 – Conceito da construção de um schema



Fonte: Autor desta dissertação (2016).

O esforço cognitivo necessário para guardar o conjunto de letras organizadas como “aovn eomrd” é muito maior que o necessário para guardar o mesmo conjunto organizado como “nova ordem”. Isto acontece porque o último conjunto representa um conhecimento pré-existente na LTM, e cria significância quando é apresentado ao observador, e representa, consequentemente, apenas um elemento (*chunk*) na *working memory*. Já o primeiro grupo representa apenas letras esparsas, que quando agrupadas não formam nenhum *Schema* significativo, pois não representam nenhum tipo de conhecimento pré-existente na LTM. Como consequência representam nove elementos na *working memory*.

Caspersen e Bennedsen (2007) afirmam que o conhecimento é potencializado quando a carga cognitiva da mente é direcionada para o processo de aquisição de S*chemas*, ou seja, quando a carga cognitiva GL é maximizada, potencializando a formação de S*chemas* pela mente.

Um aspecto importante percebido na pesquisa diz respeito à forma como os autores se referiram ao conceito. Nem todos os artigos identificados no mapeamento

utilizaram o termo S*chema* para se referir às estruturas de informações mentais que representam o conhecimento acerca de programação.

Dillon et al. (2012) fazem constante uso do termo *Mental Models* para referenciar a Schema. No trabalho, os autores realizaram um estudo de caso com o intuito de comparar como eram construídos os modelos mentais dos estudantes de programação durante o aprendizado da disciplina CS150.

Ambrosio et al. (2011) definem *Mental Models* como o formato cognitivo no qual a realidade externa é aprendida e organizada, porém faz uso indiscriminado do termo *schemas*, o que leva o leitor a enfrentar dificuldades de entendimento em um primeiro contato com o trabalho.

Em pesquisas externas, aquém da metodologia utilizada neste mapeamento, foi identificado o termo S*chema* como pertencente à teoria cognitiva de Sir Jean William Fritz Piaget, famoso epistemólogo suíço do século XX. Segundo McLeod (2009), o termo S*chema* foi definido por Piaget (1952) e representa o “bloco de construção básico” do comportamento inteligente.

Apesar do fato de não ter sido encontrado nenhuma referência direta à Piaget nos artigos do mapeamento que foram estudados, a definição do termo descrito por ele guarda semelhanças ao utilizado na Teoria da Carga Cognitiva. As principais hipóteses levantadas para que não tenham havido referências diretas ao trabalho do autor são: (I) O trabalho de Piaget não está associado, diretamente, à computação. (II) O trabalho de Piaget data da década de 1930, época anterior ao desenvolvimento da psicologia cognitiva (NEISSER, 1967).

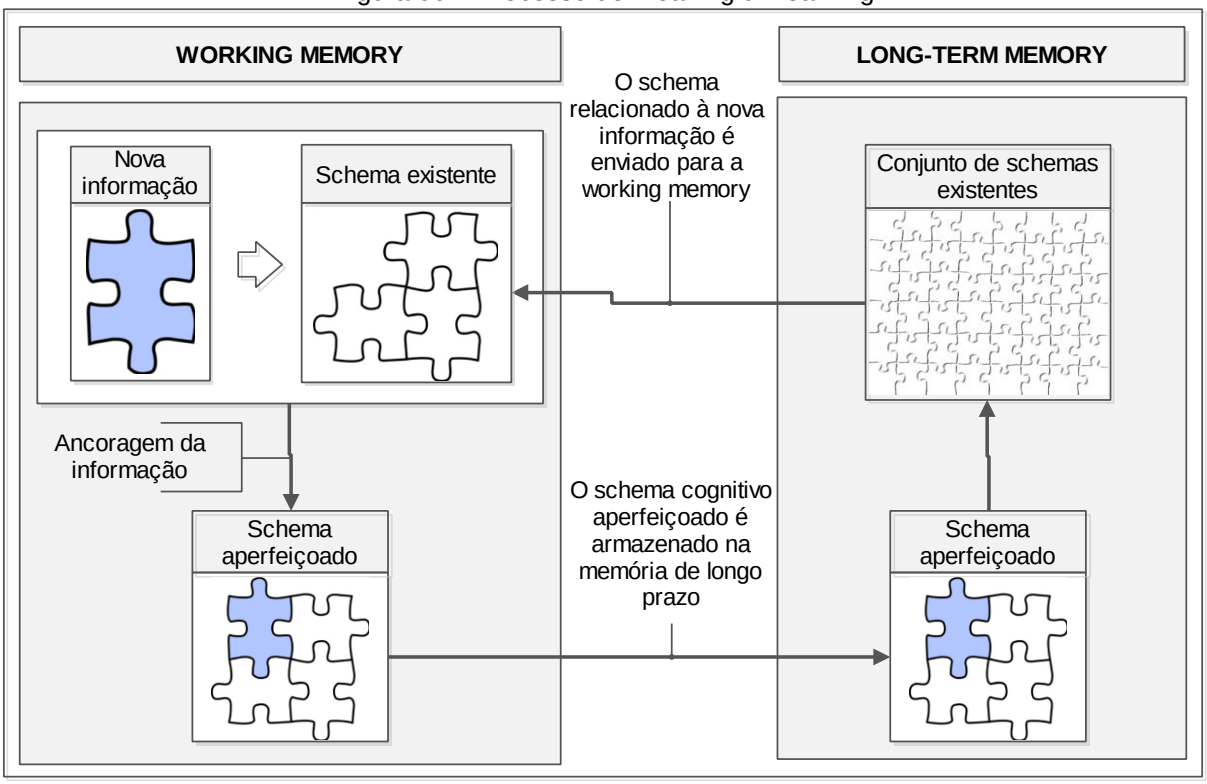
Contudo, ficam claros os vestígios da contribuição do trabalho de Jean Piaget para a computação, sendo necessário uma pesquisa mais aprofundada para descobrir as origens dessa influência.

LEARNING TYPES

Neste mapeamento, com base na CLT, foram identificados dois tipos de aprendizados: *Rote Learning* e *Meaningful Learning*.

O processo de aprendizado denominado *Meaningful Learning*, ou aprendizado significativo, ocorre quando a nova informação aprendida é imediatamente associada à alguma informação pré-existente na LTM, permitindo que esta seja contextualizada em um entendimento maior e mais complexo.

Figura 90 – Processo de Meaningful Learning



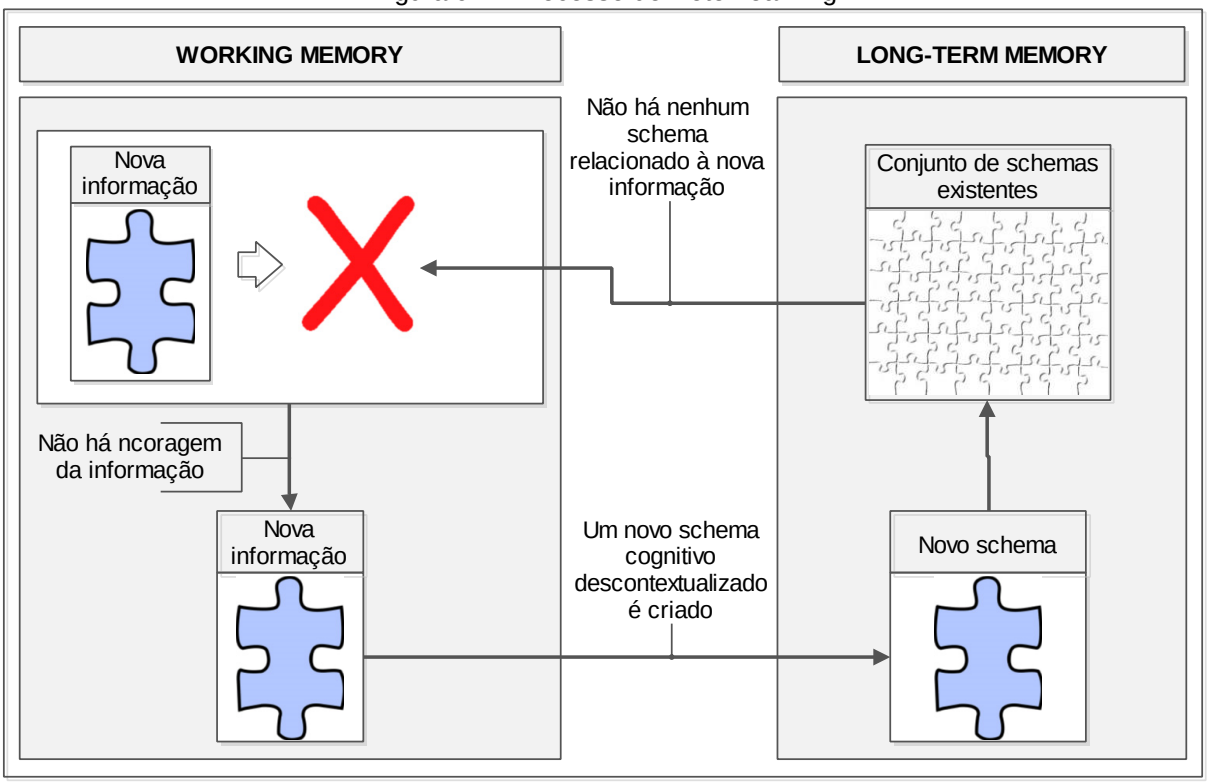
Fonte: Adaptado de Covington e Benegas (2005)

Segundo Mayer (1981), o aprendizado significativo ocorre quando o aluno consegue conectar uma nova informação à um *Schema* já existente na LTM. Também pode-se entender o conceito de aprendizado significativo por meio da explicação do mecanismo de *meaningful learning* descrito por Caspersen e Bennedsen (2007) (Figura), sendo possível perceber o processo de aprendizado onde informações pré-existentes são integradas com novas informações adquiridas, possibilitando o aprendizado contextualizado.

Covington e Benegas (2005) exemplificam o conceito de *meaningful learning* por meio do uso do sinal de “=”, cujo significado os alunos trazem de forma bem estabelecida da álgebra e cujo significado difere da programação. O autor sugere que os instrutores evitem que os *Schemas* existentes na mente dos alunos sobre o uso algébrico do sinal de “=” atrapalhem a formação de novos *Schemas* acerca de programação.

O processo de aprendizado denominado *Rote Learning* é aquele onde o aprendizado se faz através da memorização. Onde não há nenhum tipo de conhecimento pré-existente na LTM com o qual a nova informação possa ser ancorada e contextualizada. Pode ser melhor compreendida como o processo de aprendizado onde o aluno deve decorar a informação, pois não há nenhum tipo de conhecimento existente que o leve a relacionar tal informação. A Figura ilustra o conceito.

Figura 91 – Processo de Rote Learning



Fonte: Adaptado de Covington e Benegas (2005).

Covington e Benegas (2005) condenam o uso de abordagens de memorização para o ensino de programação, devido às limitações da STM e afirmam que o ensino de

tópicos técnicos depende do desenvolvimento de úteis abstrações de alto nível por parte do estudante.

EFFECTS

Para mitigar a carga cognitiva EL e maximizar a GL, como fora descrito por Morrison et al. (2014), a CLT desenvolve algumas técnicas denominadas *Effects*. Tais técnicas são citadas como recomendações instrucionais baseadas nas premissas defendidas pela CLT. Caspersen e Bennedsen (2007) afirmam que a maioria das pesquisas na área focam em identificar associações entre os *effects* e as técnicas instrucionais. Nesta dissertação as principais técnicas identificadas foram: *worked-examples* e *example completion*.

A técnica de *worked-examples* consiste em uma descrição detalhada de como resolver o problema, focando nos estados do problema e nas etapas necessárias para a sua resolução.

Abdul-Rahman e du Boulay (2014) afirmam que estudos demonstram que o uso de *worked-examples* tem sido uma importante estratégia pedagógica para a aquisição de habilidades cognitivas e habilidades em resolução de problemas de programação. Os autores relatam que a técnica é um formato instrucional fundamentado na CLT.

Cooper et al. (2012) relatam que seus estudos foram influenciados pela apresentação do autor John Sweller no ICER 2008. Eles citam que o próprio Sweller os aconselha a usarem a técnica para ensinar cada conceito de programação.

A técnica denominada *example completion* consiste em completar problemas de programação parcialmente resolvidos. Chi et al. (1989) afirmam que a técnica de *worked-examples* não funcionam com “estudantes menos capacitados” quando as atividades não assemelham-se a problemas convencionais. Como solução a este problema van Merrienboer e Krammer (1987) propõem o uso de *exemple completion*

para os alunos, para que os mesmos possam observar, modificar e ampliar os programas.

APÊNDICE C – LISTA DE EXERCÍCIOS

Neste apêndice estão contidas as descrições de todas as três questões aplicadas na pesquisa experimental, realizada nesta dissertação.

QUESTÃO 1 – TRIÂNGULO DE SIERPINSKI

Com base no algoritmo abaixo, identifique a quantidade de vezes que a função recursiva *sierpinski(...)* é chamada:

(a) 3 (b) 9 (c) 13 (d) 40

```
1.  public class Algo {
2.      void sierpinski(int n, Point p1, Point p2, Point p3) {
3.          if (n == 0) {
4.              // desenha os 3 lados do triangulo
5.              drawLine(p1, p2);
6.              drawLine(p2, p3);
7.              drawLine(p3, p1);
8.          } else {
9.              Point m1, m2, m3;
10.             // Calcula o ponto medio (m1) entre os pontos 1 e 2
11.             m1.x = (p1.x + p2.x) / 2;
12.             m1.y = (p1.y + p2.y) / 2;
13.             // Calcula o ponto medio (m2) entre os pontos 2 e 3
14.             m2.x = (p2.x + p3.x) / 2;
15.             m2.y = (p2.y + p3.y) / 2;
16.             // Calcula o ponto medio (m3) entre os pontos 3 e 1
17.             m3.x = (p3.x + p1.x) / 2;
18.             m3.y = (p3.y + p1.y) / 2;
19.
20.             // chama recursivamente o algoritmo de sierpinski
21.             sierpinski(n - 1, p1, m1, m3);
22.             sierpinski(n - 1, m1, p2, m2);
23.             sierpinski(n - 1, m3, m2, p3);
24.         }
25.     }
26.
27.     public static void main(String[] args) {
28.         Algo obj = new Algo();
29.         obj.sierpinski(2,new Point(400,0), new Point(780,740), new Point(20,740));
30.     }
31. }
```

QUESTÃO 2 – TORRE DE HANOI

Com base no algoritmo abaixo, identifique a correta ordem das impressões impostas pela sua execução:

(a)	(b)	(c)
mova disco 1 de A para B mova disco 2 de A para C mova disco 3 de A para C mova disco 1 de B para A mova disco 1 de B para C mova disco 1 de A para C mova disco 1 de C para B	mova disco 1 de A para C mova disco 2 de A para B mova disco 1 de C para B mova disco 2 de A para C mova disco 1 de B para A mova disco 1 de A para C mova disco 2 de B para C	mova disco 1 de A para C mova disco 2 de A para B mova disco 1 de C para B mova disco 3 de A para C mova disco 1 de B para A mova disco 2 de B para C mova disco 1 de A para C

```
1.  public class Algo {
2.      void Hanoi(int n, char origem, char neutro, char destino) {
3.          if (n == 1) {
4.              System.out.println("mova disco "+n+" de "+origem+" para "+destino);
5.          } else {
6.              Hanoi(n - 1, origem, destino, neutro);
7.              System.out.println("mova disco "+n+" de "+origem+" para "+destino);
8.              Hanoi(n - 1, neutro, origem, destino);
9.          }
10.     }
11.
12.     public static void main(String[] args) {
13.         Algo obj = new Algo();
14.         obj.Hanoi(3, 'A', 'B', 'C');
15.     }
16. }
```

QUESTÃO 3 – CÁLCULO FATORIAL

(a) 5 (b) 24 (c) 120 (d) 720

Com base no algoritmo abaixo, informe o valor da variável *result* (linha 13):

```
1.  public class Algo {
2.      int fatorial(int n)
3.      {
4.          if (n == 0)
5.              return 1;
6.          else
7.              return n * fatorial(n - 1);
8.      }
9.
10.     public static void main(String[] args) {
11.         Algo obj = new Algo();
12.         int result;
13.         result = obj.fatorial(5);
14.     }
15. }
```

APÊNDICE D – QUESTIONÁRIO QUALITATIVO

Com base em uma escala qualitativa de um (1) à cinco (5), responda aos seguintes questionamentos:

1) Qual o nível de **dificuldade** enfrentada por você para responder ao **pré-teste**?

1	Muito difícil
2	Difícil
3	Razoável
4	Fácil
5	Muito fácil

2) Qual o nível de **dificuldade** enfrentada por você para responder ao **pós-teste**?

1	Muito difícil
2	Difícil
3	Razoável
4	Fácil
5	Muito fácil

3) Qual o nível de **dificuldade** enfrentada por você para **entender o funcionamento** da metodologia lúdica proposta?

1	Muito difícil
2	Difícil
3	Razoável
4	Fácil
5	Muito fácil

4) O quão **eficaz** foi a metodologia lúdica proposta para que você pudesse compreender melhor o algoritmo de **Sierpisnki**?

1	Nada eficaz
2	Pouco eficaz
3	Razoável
4	Eficaz

5	Muito eficaz

5) O quão **eficaz** foi a metodologia lúdica proposta para que você pudesse compreender melhor o algoritmo de **Hanoi**?

1	Nada eficaz
2	Pouco eficaz
3	Razoável
4	Eficaz
5	Muito eficaz

6) O quão **eficaz** foi a metodologia lúdica proposta para que você pudesse compreender melhor o algoritmo do cálculo **Fatorial**?

1	Nada eficaz
2	Pouco eficaz
3	Razoável
4	Eficaz
5	Muito eficaz

7) O quão **relevante** você considera uma abordagem que utiliza metáforas lúdicas, no lugar da textual, para ensinar conceitos de programação de computadores?

1	Nada relevante
2	Pouco relevante
3	Razoável
4	Relevante
5	Muito relevante

8) Escreva com suas palavras o que achou da proposta lúdica para o ensino de conceitos de programação, e o que você mudaria ou adicionaria para melhorá-la.

_____

_____

_____

_____

_____

ANEXO A – PARECER

UNIVERSIDADE SALVADOR - UNIFACS/BA

PARECER CONSUBSTANCIADO DO CEP

DADOS DO PROJETO DE PESQUISA

Título da Pesquisa: METODOLOGIAS LÚDICAS APLICADAS NO ENSINO DE CONCEITOS DE PROGRAMAÇÃO DE COMPUTADORES
Pesquisador: Artur Henrique Kronbauer
Área Temática:
Versão: 1
CAAE: 54856416.3.0000.5033
Instituição Proponente: Universidade Salvador - UNIFACS/BA
Patrocinador Principal: Financiamento Próprio

DADOS DO PARECER

Número do Parecer: 1.487.894

O projeto METODOLOGIAS LÚDICAS APLICADAS NO ENSINO DE CONCEITOS DE PROGRAMAÇÃO DE COMPUTADORES, caracteriza-se por uma pesquisa experimental quantitativa, num estudo de caso com alunos iniciantes dos cursos de computação e engenharias da Universidade Salvador

Apresentação do Projeto:
O objetivo principal é o de "testar a hipótese da eficácia das abordagens lúdicas baseadas na psicologia para a melhoria do ensino de conceitos de programação de computadores", tendo como hipótese: "As metodologias lúdicas, fundamentadas na psicologia cognitiva, podem construir nos alunos iniciantes os corretos modelos mentais, necessários para facilitar o aprendizado de conceitos de programação de computadores".

Os objetivos específicos são:
a) Realizar uma análise quantitativa dos dados obtidos a partir da pesquisa.
b) Utilizar os resultados da análise para publicar artigos científicos e propor trabalhos futuros.

Continuação do Parecer: 1.366.396

Avaliação dos Riscos e Benefícios:

O risco é baixo, uma vez que não haverá levantamento de informações sobre os dados sociodemográficos dos estudantes.

Comentários e Considerações sobre a Pesquisa:

O projeto apresenta pertinência em todo o seu desenvolvimento.

Considerações sobre os Termos de apresentação obrigatória:

Todos os termos de apresentação obrigatória estão presentes.

Recomendações:

Sem recomendações.

Conclusões ou Pendências e Lista de Inadequações:

Sou de parecer favorável para aprovação

Considerações Finais a critério do CEP:

Mantido parecer do relator.

Este parecer foi elaborado baseado nos documentos abaixo relacionados:

Tipo Documento	Arquivo	Postagem	Autor	Situação
Informações Básicas do Projeto	PB_INFORMAÇÕES_BÁSICAS_DO_PROJETO_648359.pdf	04/04/2016 10:38:42		Aceito
Folha de Rosto	Folha_de_Rosto.pdf	04/04/2016 10:38:02	Artur Henrique Kronbauer	Aceito
Outros	Carta_de_Anuencia.pdf	31/03/2016 17:44:49	Artur Henrique Kronbauer	Aceito
Outros	Curriculo_Lattes_Almir.pdf	16/02/2016 16:49:18	Artur Henrique Kronbauer	Aceito
Outros	Curriculo_Lattes_Artur.pdf	16/02/2016 16:33:15	Artur Henrique Kronbauer	Aceito
Projeto Detalhado / Brochura Investigador	Projeto_Detalhado.pdf	16/02/2016 16:24:27	Artur Henrique Kronbauer	Aceito
TCLE / Termos de Assentimento / Justificativa de Ausência	Termo_de_Consentimento_Livre_e_Esclarecido.pdf	16/02/2016 16:09:00	Artur Henrique Kronbauer	Aceito
Orçamento	Orcamento.pdf	16/02/2016 16:07:04	Artur Henrique Kronbauer	Aceito
Cronograma	Cronograma.pdf	16/02/2016 16:04:06	Artur Henrique Kronbauer	Aceito

Continuação do Parecer: 1.366.396

Situação do Parecer:

Aprovado

Necessita Apreciação da CONEP:

Não

SALVADOR, 11 de Abril de 2016

Assinado por:

TATIANA SENNA GALVÃO NONATO ALVES

(Coordenador)

Endereço: Av. Luís Viana Filho 3146, 3º. andar -Torre Norte - Campus Paralela
Bairro: Paralela **CEP:** 41.720-200
UF: BA **Município:** SALVADOR
Telefone: (71)3271-2740 **Fax:** (71)3271-2740 **E-mail:** cep@unifacs.br

ANEXO B – TERMO DE CONSENTIMENTO

Termo de Consentimento Livre e Esclarecido

Comitê de Ética em Pesquisa da UNIFACS
Programa de Pós-Graduação em Sistemas e Computação (PPGCOMP)
Laureate International Universities
Avenida Luís Viana Filho, Nº 3146, Imbuí - CEP: 41720-200 - Salvador-Bahia.
Telefone (71) 3021-2800.

Metodologias lúdicas aplicadas no ensino de conceitos de programação de computadores

Eu_____;
estou sendo convidado(a) a participar de um experimento para avaliar a eficácia de uma abordagem lúdica baseada na psicologia para a melhoria do ensino de conceitos de programação de computadores.

A plataforma foi concebida como projeto de pesquisa do professor Dr. Artur Henrique Kronbauer, vinculado ao Programa de Pós-Graduação em Sistemas Computacionais (PPGCOMP) da Universidade Salvador. O referido pesquisador pode ser contatado pelo telefone (71) 99111-8409 ou pelo endereço de e-mail arturhk@gmai.com.

Recebi esclarecimentos sobre o experimento e estou ciente de que minha privacidade será respeitada, ou seja, meu nome e imagem serão mantidos em sigilo. Eu autorizo a utilização do questionário contendo as minhas respostas preenchidas durante a realização do experimento, entendo que as informações serão utilizadas somente para os fins desta pesquisa, bem como serão divulgadas apenas em artigos e na redação dos trabalhos científicos orientados pelo professor Dr. Artur Henrique Kronbauer.

Estou ciente que poderei solicitar esclarecimentos quanto a quaisquer dúvidas durante a realização do experimento e terei acesso aos resultados obtidos. Tenho ciência de que poderei me recusar a responder qualquer pergunta e que posso me negar a participar do estudo ou retirar meu consentimento a qualquer momento, sem prévia justificativa.

Manifesto meu livre consentimento em participar.

Salvador, _____ de _____ de _____.

_____
Nome e assinatura do participante

_____
Nome e assinatura do pesquisador