DOI: 10.5433/1679-0375.2013v34n1p45

# Jogo eletrônico para aprendizado e metaforização de algoritmos recursivos

## *Video game for learning and metaphorization of recursive algorithms*

Ricardo Inácio Álvares e Silva[1]; Rosilane Ribeiro da Mota[2]; Jacques Duílio Brancher[3]


## Resumo

O aprendizado de algoritmos recursivos em programação de computadores é problemático, pois sua execução e resolução não se dão de maneira natural à forma de pensamento com que as pessoas são treinadas e acostumadas desde cedo. Assim como em outros tópicos de algoritmos, utilizam-se metáforas para fazer paralelos entre o abstrato e o concreto para ajudar na compreensão do funcionamento de algoritmos recursivos. Porém, as metáforas clássicas usadas nessa área, como cálculo de fatorial recursiva e o jogo Torres de Hanói, podem confundir mais ou serem insuficientes. Neste trabalho, foi produzido um jogo eletrônico para computador para auxiliar alunos dos cursos de informática no aprendizado de algoritmos recursivos. Ele foi projetado para possuir características de um jogo comum, com narrativa e elementos de jogabilidade clássicos e comuns a esse tipo de produto. O auxílio ao ensino acontece através da metaforização, ou seja, através da apresentação de situações de jogo que remetem à algoritmos recursivos. Para tal, foram projetadas e imbuídas no jogo quatro metáforas válidas relacionadas à teoria, além de outras referencias menores ao tema.

**Palavras-chave:** Jogo eletrônico. Algoritmos recursivos. Metáforas válidas.


## Abstract

The learning of recursive algorithms in computer programming is problematic, because its execution and resolution is not natural to the thinking way people are trained and used to since young. As with other topics in algorithms, we use metaphors to make parallels between the abstract and the concrete to help in understanding the operation of recursive algorithms. However, the classic metaphors employed in this area, such as calculating factorial recursively and Towers of Hanoi game, may just confuse more or be insufficient. In this work, we produced a computer game to assist students in computer courses in learning recursive algorithms. It was designed to have regular video game characteristics, with narrative and classical gameplay elements, commonly found in this kind of product. Aiding to education occurs through metaphorization, or in other words, through experiences provided by game situations that refer to recursive algorithms. To this end, we designed and imbued in the game four valid metaphors related to the theory, and other minor references to the subject.

**Key words:** Video game. Recursive algorithm. Metaphorization. Mental models.

[1] Mestrando, Departamento de Computação, Universidade Estadual de Londrina. E-mail: ricardoinacio@me.com
[2] Mestre, Departamento de Computação, Pontifícia Universidade Católica de Minas Gerais.
[3] Doutor, Departamento de Computação, Universidade Estadual de Londrina. E-mail: jacques@uel.br

## Introdução

O aprendizado de algoritmos recursivos em programação de computadores é problemático, pois sua execução e resolução não se dá de maneira natural à forma de pensamento com que as pessoas são treinadas e acostumadas desde cedo. Além disso, a utilização de algoritmos recursivos pode parecer não natural às linguagens de programação. Inclusive, é natural que os alunos se espantem quando descobrem ou são ensinados que uma função pode chamar a ela mesma (SANDERS; GALPIN; GÖTSCHI, 2006).

Assim como em outros tópicos de algoritmos, utiliza-se metáforas para fazer paralelos entre o abstrato e o concreto para ajudar na compreensão do funcionamento de algoritmos recursivos (SANDERS; GALPIN; GÖTSCHI, 2006). Porém, as metáforas clássicas usadas nessa área, como cálculo de fatorial recursiva e o jogo Torres de Hanói, podem justamente confundir mais ou serem insuficientes, pois elas possuem ao menos uma das seguintes características: simplicidade demasiada; complexidade demasiada; não ser atraente; e derivarem de situações pouco conhecidas (WIRTH, 2008). Constatou-se que, ao utilizar essas experiências clássicas, muitos alunos criam metáforas que não são válidas para todos os casos de recursividade e acabam sendo induzidos ao erro (SANDERS; GALPIN; GÖTSCHI, 2006).

Pensou-se então em utilizar jogos eletrônicos como forma de auxílio para a metaforização de algoritmos recursivos por dois motivos básicos: eles são atraentes e possuem capacidade para prover experiências fictícias (WONG et al., 2007). Desde que foram criados, os jogos eletrônicos conquistaram a atenção do público infantil e jovem, se tornando para muitos a atividade preferida, alvo de devoção do tempo livre. No início deste século, notou-se que as gerações que cresceram com jogos mantiveram o costume da jogatina e o interesse na atividade. Nos relatos de usos de jogos eletrônicos para educação fica evidente a capacidade de atrair e de empolgar os alunos (EL-NASR; SMITH, 2006).

Além disso, os jogos tem a capacidade de contar histórias por meio de gráficos e a de fixação por meio da interação ágil com o usuário. Isso os confere também a capacidade de oferecerem lembranças e experiências, o que os torna, por tanto, uma ótima maneira para auxiliar na formação de metáforas (WONG et al., 2007).

*Objetivo*

Este trabalho teve como objetivo a implementação de um jogo que fosse atraente para seu público alvo, os alunos de cursos de Informática, e os auxiliassem no aprendizado de algoritmos recursivos. A abordagem não é a de um jogo que ensine diretamente, mas sim um que oferecesse uma abordagem metafórica válida sobre recursividade, de tal modo que, quando os alunos forem de fato aprender o assunto, estejam aptos fazer paralelos que facilitem no entendimento dos processos recursivos e que seja válido para todos os seus casos.

## Revisão da Literatura

Desde que foram inventados, os jogos eletrônicos conquistaram rapidamente o público jovem e evoluíram de simples blocos e figuras flutuantes, a complexas obras, com gráficos 3D, som de cinema e linha de história avançada (WONG et al., 2007). Entretanto, a possibilidade e capacidade de interação são as características que mais chamam a atenção, fazendo com que educadores de todas as áreas, desde escola primária até universidades, começassem a pensar em maneiras de usá-los para auxiliar no ensino de seus respectivos cursos (EL-NASR; SMITH, 2006).

Naturalmente, os jogos educativos chegaram também à área da computação, com produtos iniciais concentrados principalmente nas matérias

de programação básica, baseados em projetos em que os alunos têm que construir jogos aplicando as teorias aprendidas em sala de aula (BECKER, 2001). Os resultados mostraram que eles se esforçam mais nos trabalhos, buscam conteúdos extraclasses para realizar funções por conta própria e se esforçam para cumprir atividades extras propostas. Concluíram então que os alunos se sentem mais motivados na realização do trabalho e aprendem melhor as disciplinas necessárias e até mesmo outras fora de escopo, mas importantes, como consultas à documentação, processamento de imagens (JIMÉNEZ-PERIS; KHURI; PATIÑO-MARTÍNEZ, 1999), dentre outras.

Atualmente, é possível classificar os projetos existentes de ensino com jogos em três classes (SUNG et al., 2008):

1. Trabalhos de desenvolvimento de jogos por assunto de uma matéria. São menores e exigem que os alunos programem um aspecto de jogo que aborde especificamente uma teoria de alguma disciplina.

2. Trabalhos de desenvolvimento extensivo de jogos. São aqueles em que há a preocupação de se produzir bons jogos e feitos usando arcabouços, ferramentas profissionais, motores gráficos, dentre outras. São mais abrangentes, utilizam ferramentas de alto nível e, por vezes, não requerem muito conhecimento de programação.

3. Projetos que tentam ensinar utilizando de ambientes de jogo. Os educadores produzem um jogo com conteúdo pedagógico e os alunos o jogam. O jogo pode buscar ensinar a matéria ou apenas oferecer uma metáfora para facilitar o entendimento e aprendizado.

Dentre os projetos da primeira classe, podemos citar Jiménez-Peris, Khuri e Patiño-Martínez (1999), em que foi programado um arcabouço para aliviar as dificuldades de exibição gráfica e paralelismo existentes em jogos, com o objetivo de permitir aos alunos concentrar apenas no manuseio e implementação dos aspectos realmente relevantes. Em seguida, propuseram aos alunos que produzissem os jogos *Asteroids* e *Tetris*, e jogos de cartas. Constataram que eles foram úteis no treinamento de conceitos de programação básica, como laços, gerenciamento de memória, tipos abstratos de dados e programação orientada por objetos.

Seguindo essa tendência, Becker (2001) pediu que os alunos produzissem *Asteroids* e *Campo Minado*, mas sem arcabouço, em modo texto e apenas com recursos simples. O uso de recursos adicionais era opcional (valendo pontos extras), mas isso não impediu que a maioria dos alunos os implementassem, acarretando em aprendizado além do esperado.

Já dentre os projetos da segunda classe, temos como exemplos El-Nasr e Smith (2006) e Xu, Blank e Kumar (2008). No primeiro os alunos foram levados a fazer modificações nos jogos *Warcraft III* e *Unreal Tournament*, além da criação de conteúdo usando o motor *Web Driver*, enquanto no segundo os alunos trabalharam na modificação de um jogo de código livre escolhido por eles (o artigo cita *Doom*, *Quake* e *Crossfire*). Ambos se concentraram em prover ambientes de desenvolvimento de mais alto nível, que exigissem menos conhecimento técnico. Assim, os alunos puderam se concentrar mais nos elementos teóricos e no uso da capacidade criativa. Ainda assim, houve problemas de perda de tempo excessivo com detalhes menos interessantes como parte gráfica e definição de regras - mesmo com a programação em alto nível.

Dentre os projetos da terceira classe, temos Giguette (2003), que desenvolveu jogos simples para serem jogados antes das aulas que ensinam pilhas, árvores binárias e grafos (CORMEN et al., 2001). O objetivo dos jogos era apresentar exemplos concretos, prover ambientes para experimentação e criar a metáfora em que os algoritmos são estratégias, usados para vencer um jogo com determinadas regras. Há também Barnes et al. (2008), que desenvolveu dois jogos, *Saving Sera* e *The Catacombs*. No primeiro, os jogadores devem

resolver problemas simples, como organização de ovos em caixas, seguindo algoritmos, que devem ser corrigidos antes de usados. No segundo, os jogadores devem ajudar um aprendiz de mago a salvar crianças ao escolher as soluções corretas para passar pelos obstáculos.

O uso dos jogos no ensino de computação mostrou ter pelo menos duas vantagens claras:

- Motivação: os alunos se mostraram bastante motivados na realização e/ou participação dos projetos. Por esse motivo, muitos fizeram trabalhos além do que foi pedido, o que os levou a aprenderem mais. Muitos também aumentaram suas expectativas em relação ao seu respectivo curso de Computação e a empolgação contagiou alunos do ensino médio[2], fazendo-os se interessarem pela área (BARNES et al., 2008).

- Metaforização: os jogos se mostraram uma excelente maneira de metaforizar teorias da computação. Com eles, problemas abstratos são representados em situações concretas, visuais e interativas (GIGUETTE, 2003; BARNES et al., 2008).

Quando um professor apresenta aos seus alunos uma nova teoria ou matéria da disciplina, eles criam para si metáforas que os ajudam a compreender e empregá-las. O problema é que muitas vezes, eles não dispõem de bons exemplos práticos para essa metaforização, o que resulta no uso de metáforas indevidas que os atrapalham no aprendizado e uso, às vezes até confundindo-os. Por isso, é importante apresentar junto com a teoria, metáforas válidas e úteis (SANDERS; GALPIN; GÖTSCHI, 2006).

Ensinar algoritmos recursivos é notadamente complicado, envolve vários problemas. Dentre eles estão a falta de capacidade dos novatos em programação para modelar o caso base da recursão (HABERMAN; AVERBUCH, 2002). Outro problema é o fato de que muitos alunos tendem a criar metáforas inválidas, que funcionam apenas para algumas das situações, pois a recursividade é de difícil visualização e não natural em relação à maneira que estamos acostumados a pensar. Dentre os erros comuns causados por essas metáforas estão a percepção de que os algoritmos recursivos executam paralelamente e que a função recursiva chama a si mesma, ao invés de uma nova instância da mesma função (SANDERS; GALPIN; GÖTSCHI, 2006).

Por tanto, é necessário empenho antes das aulas de recursividade para exibição e construção de uma ou mais metáforas válidas para os alunos. As metáforas escolhidas devem incluir as ideias de que o trabalho é realizado por vários agentes similares, que funcionam de maneira igual, um agente divide o problema e entrega a pelo menos um outro e há um agente especial que inicia a resolução do problema (SANDERS; GALPIN; GÖTSCHI, 2006).

Desde que a recursividade foi incorporada às grades de ensino dos cursos de programação e computação, vários exemplos foram criados para auxílio no seu ensino. Normalmente os exemplos são apresentados aos alunos ao mesmo tempo em que se busca introduzir os conceitos da matéria, ou seja, buscam ensinar diretamente ao mostrar e explicar código.

Dentre estes exemplos temos a resolução do número de Fibonacci, do fatorial, da permutação, binomial, ordenação por *quicksort*, pesquisa binária e o problema das Torres de Hanói (CORMEN et al., 2001). Alguns deles tratam da resolução de problemas numéricos, o que é uma desvantagem por terem pouca conexão com a vida real.

A resolução do número de Fibonacci é um bom exemplo de quando não se deve usar recursividade. Enquanto funciona bem para casos onde há poucos cálculos, é uma técnica bastante ineficiente para casos mais extensos. A resolução de fatoriais sofre do mesmo problema do Fibonacci, além de ser simples demais, o que pode ser bom para uma introdução, mas pode também ser pouco esclarecedor.

[2] *High School* nos Estados Unidos.

O problema das Torres de Hanói, travessia de árvores binárias e *backtracking*[3] é que são complexos demais para novatos, principalmente para o caso de cursos que ensinam recursividade logo nas primeiras disciplinas. E todos esses casos supracitados, por se tratarem de exemplo direto em código, não são bons para a criação de metáforas por parte dos alunos (WIRTH, 2008).

Com este problema em evidência, vários professores e pesquisadores se empenharam em criar novos exemplos, com ênfase na concretude. Assim surgiram novos exemplos, baseados em fatos e acontecimentos do cotidiano, ou exemplos que sejam de fácil visualização e exibição. Dentre esses casos podemos citar:

- Kay (2000) desenvolveu um método de ensino que metaforiza com o filme *Guerra nas Estrelas*, que é bastante conhecido, principalmente entre os alunos de cursos de computação, o que ajuda a atrair a atenção deles na aula. O método consiste em ensinar recursividade de modo que ao atingir o problema, uma instância da função "usa a força" e chama outra instância para resolver o problema dela. Ao fazer este paralelo, o professor chama mais atenção dos alunos, que guardam melhor essa maneira de criar algoritmos recursivos.

- Gordon (2006) usou desenhos geométricos gerados recursivamente em suas aulas. Ele explica que a vantagem de tal método está no fato de que o resultado de cada instância recursiva é exibido no resultado final, ao invés de serem descartados ou ocultados como ocorre com os exemplos de problemas numéricos. Ele utilizou quatro abordagens de desenhos geométricos: triângulos aninhados, polígonos aninhados, fractais e divisão de plano.

- Goldwasser e Letscher (2007) desenvolveram um método teatral que ajuda na metaforização da recursividade ao colocar os alunos para atuar como se fossem estados distintos de um algoritmo recursivo em funcionamento. Nele, cada aluno recebe uma folha com instruções a serem seguidas, que nada mais são que o caso base do algoritmo recursivo escolhido, e o nome de um colega, a ser chamado para resolver o problema, quando ainda sem solução. Então o professor escolhe um aluno e lhe passa um parâmetro. Esse aluno segue os procedimentos e, em seguida chama o aluno seguinte, passando-lhe um novo parâmetro. Essa dinâmica ocorre até que seja atingido o caso base. Então os alunos retornam um valor ao colega que o chamou, sucessivamente, até que o valor seja retornado ao professor. Esse método evita a formação de um caso de metáfora inválida, referente ao falso caso de uma função chama a si mesmo, ao invés de chamar uma nova instância dela.

- Wirth (2008) criou um novo exemplo para ensino de recursividade baseado no problema de estacionar carros aleatoriamente em um espaço de tamanho predeterminado. Ele considera esse exemplo simples o suficiente para alunos novatos e com boa capacidade de auxílio na criação de metáforas válidas. Isso se deve principalmente por ser baseado em um problema real contemporâneo, que elimina a ideia de função que chama a si mesma.

- Yang (2008) apresentou uma nova analogia baseada na brincadeira de cascata de dominós para ensinar recursão linear a alunos novatos das disciplinas de algoritmos. Ela foi escolhida pela capacidade de ajudar no aprendizado e visualização da recursão linear, bem como por ser uma brincadeira comum a quase todas as crianças, o que cria uma conexão entre a teoria e a prática. Além disso, ela se baseia em semântica computacional ao invés de problemas matemáticos.

Além de exemplos, pesquisadores também desenvolveram *softwares* interativos e visuais para auxilio no ensino de recursividade. Dentre eles há o *SRec*, apresentado por Velázquez-Iturbide, Pérez-Carrasco e Urquiza-Fuentes (2008), um sistema que gera três níveis de visualização de um algoritmo

[3] Algoritmo usado, por exemplo, na resolução de labirintos (HADEN, 2006).

recursivo: código; pilha de execução; e árvore de ordem de execução. A visualização de árvores ainda oferece como opções a exibição de todas as chamadas feitas, exibição de chamadas passadas atenuadas ou exibição apenas das chamadas ativas. A execução do algoritmo é controlável utilizando-se botões estilo depurador, inclusive com operações de voltar a animação. Os autores avaliaram que o sistema é de fácil uso e ajuda o professor no ensino e os alunos no entendimento de recursividade.

No entanto, fica claro que essa abordagem não serve para metaforização, pois apenas representa e oferece visualização dos algoritmos de forma abstrata e técnica. Uma abordagem via *software*, visual e metaforizadora ocorre no projeto de Haden (2006). Nele foi implementada a capacidade de demonstrar o *backtracking* em ação, na forma de um rato que deve procurar pela saída de um labirinto. O problema é ser demasiadamente simples e não oferece uma representação completa dos problemas da recursividade.

## Metodologia

A metodologia de trabalho se deu nos seguintes passos: estudo e montagem de um *Documento de Design* (SCHUYTEMA, 2008); e estudo das ferramentas a serem usadas, e produção do jogo.

O trabalho começou com a realização de seções de *brainstorming* em um grupo de estudos, envolvido com projetos de pesquisa em jogos eletrônicos. Com as anotações dessas seções produziu-se um *Documento de Design*, prática comum adotada por profissionais da área de jogos. Em projetos maiores, sua função é detalhar todos os elementos do jogo, desde história e personagens até intrínsecas da jogabilidade, para evitar perda de direção e objetividade. Mas para projetos menores, recomenda-se não exagerar no esforço despendido com o processo. Portanto, foi produzido um documento contendo apenas as tarefas, também conhecidas como *puzzles* (quebra-cabeças em inglês), a serem implementados no jogo e sua relação metafórica com a teoria.

O próximo passo foi a implementação do jogo definido no *Documento de Design* e nas seções *brainstorming*. Ele envolveu a procura, o estudo e a interligação das várias ferramentas utilizadas, a produção do jogo e a criação de seus recursos artísticos.

*O Jogo*

O jogo produzido foi projetado para se encaixar em um estilo tradicional de ação, com o campo de visão do jogador por cima, tela deslizante e inimigos a serem combatidos. Foi lhe dado o nome *"Kobolds foram capturados!!!"*.

No início do jogo, há uma introdução em forma de história, que se utiliza de textos e figuras ilustrativas. Ele possui também possui três fases, dispostas em sequência, a serem vencidas pelo jogador, que são: a fase da montanha; a fase do labirinto; e a fase do fantasma.

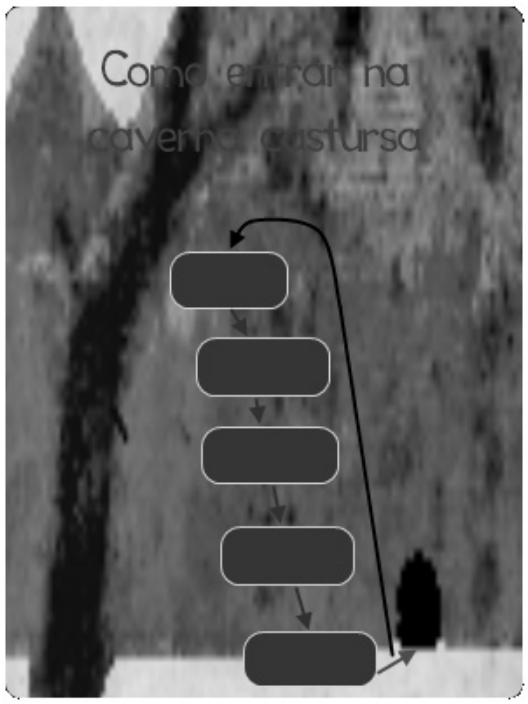
Cada fase possui uma tarefa a ser completada de modo que o jogador passe para a seguinte e o jogo prossiga. Para ajudar na compreensão da tarefa a ser realizada pelo jogador, o jogo possui narrativas que explicam o que deve ser feito, além de dar dicas sobre a melhor forma de conseguir prosseguir.

*A Fase da Montanha*

A primeira fase do jogo, a da montanha, possui cinco regiões distintas dispostas e interligadas sequencialmente, como mostra a Figura 1. Para atingir a última delas, o jogador deve passar pelas quatro anteriores.

Na primeira região, demonstrada pela Figura 2, o jogador pode perceber que há uma passagem secreta que pode ser aberta e revelada ao acionar uma roda de água. Há também um encanamento que desemboca na roda de água e que indica que sua origem está no alto da montanha. Há ainda uma peça de encanamento perdida em um extremo da região.

**Figura 1 -** Mapa com a disposição das regiões da primeira fase.



**Fonte:** autor

As próximas três regiões são similares entre si, e apresentam vários inimigos a serem batidos ou evitados. Apesar de serem semelhantes, alguns elementos, como a disposição de árvores, quantidade de inimigos e placa indicando sua altitude na montanha, mostram claramente que não são a mesma. Outro detalhe importante é que possuem um encanamento disposto de tal forma que sugere vir da região anterior e segue subindo a montanha, em direção à próxima.

**Figura 2 -** Fase da Montanha.


**Fonte:** autor

Já a última região é claramente distinta das outras, assim como a primeira, e apresenta também o encanamento que claramente vem das regiões anteriores e chega próximo a uma cachoeira, mas por pouco não a atinge. A ideia é indicar que está faltando uma peça para que o encanamento esteja no fluxo da cachoeira para que seja abastecido.

A tarefa a ser completada é a de revelar a passagem secreta e adentrá-la. Para tal, o jogador deve encontrar a peça de encanamento perdida na primeira região e instalá-la no início do encanamento, próximo à cachoeira na última região. Ao fazê-lo, o jogador descobre que o encanamento das outras regiões está defeituoso, com vazamentos, que devem ser consertados para permitir que o fluxo de água atinja a roda de água e abra a passagem para a próxima fase.

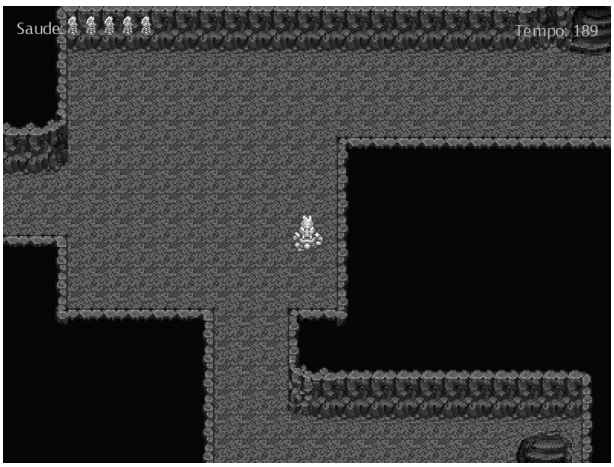
*A Fase do Labirinto*

A fase do labirinto é a segunda do jogo e é composta por uma quantidade aleatória de regiões, sendo gerada a cada vez que o jogador a atinge. O jogador deve encontrar a região final do labirinto, que possui uma saída para a próxima fase.

O processo de geração do labirinto segue certas regras que garantem que ele sempre possa ser representado por um grafo de árvore de ordem 2 e altura 8. Tal árvore possui 3 tipos de nós, que são a folha e os dois tipos de galho, que ligam a um ou dois nós filho.

Para cada tipo de nó o jogo atribui um tipo de região, que possui pelo menos uma entrada e quantas bifurcações for necessário para representar o nó corretamente. Por exemplo, a Figura 3 mostra o tipo de região que possui uma entrada e duas saídas, representando um galho que tem dois nós filhos. Há ainda dois tipos de regiões especiais, que são a inicial e a final, sendo que a primeira delas se distingue apenas por introduzir a fase, enquanto a segunda mostra claramente que o jogador atingiu o fim do labirinto.

**Figura 3 -** Fase da Montanha.




**Fonte:** autor

No início da fase, a narrativa indica que o jogador possui uma ferramenta extra para auxílio na tarefa de atravessar o labirinto. É uma placa marcadora que é posicionada na posição do personagem do jogador quando acionada, e possui um número que indica a qual altura do labirinto o está. A ideia é que ela seja utilizada tanto para indicar por quais caminhos o jogador já passou quanto para mostrar a profundidade percorrida. O correto uso da ferramenta depende totalmente do jogador, apesar de a narrativa dar dicas da melhor maneira de emprego.

*A Fase do Fantasma*

A fase do fantasma é a terceira e última do jogo. A disposição das regiões e a quantidade é a mesma da primeira fase, tendo como diferencial a presença de inimigos especiais, centrais à tarefa a ser resolvida pelo jogador.

Estes inimigos especiais estão presentes em cada uma das regiões, sendo apenas um em cada, possuem três ataques distintos e são invencíveis. Apenas o inimigo da última região, que é graficamente distinto dos das outras, pode ser vencido. Quando isso acontece, o inimigo da região anterior perde a invencibilidade e ganha um grau de transparência, que indica que este está mais fraco do que o inimigo batido. E quando este também é derrotado, o seu anterior sofre os mesmos efeitos. Esse processo segue em cadeia até que todos os inimigos sejam derrotados e a tarefa da fase completada.

*Metáforas Relacionadas à Teoria*

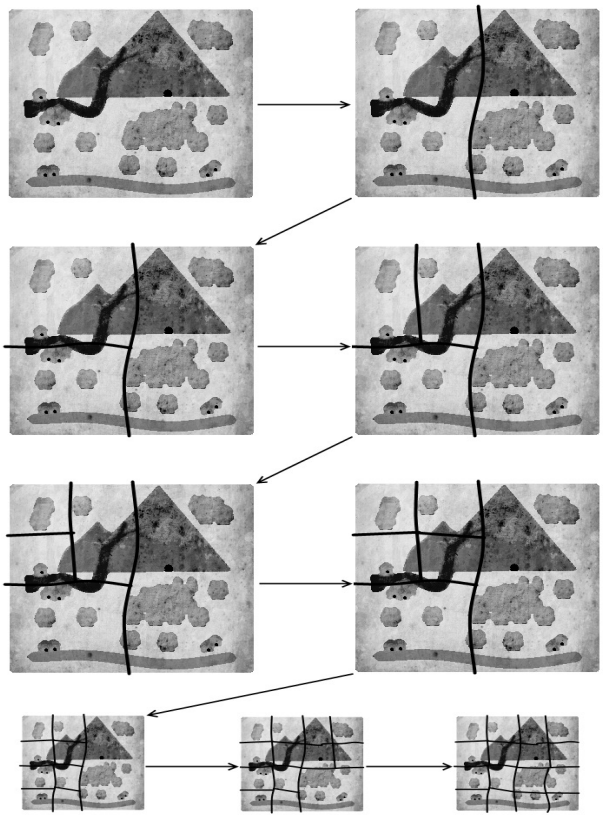
O jogo possui as seguintes metáforas em relação a algoritmos e processos recursivos:

1. A divisão do mapa na narrativa de introdução do jogo é um processo recursivo (Figura 4). O personagem lírico do jogador divide o mapa de uma região ao meio gerando duas sub-regiões, e repete o processo até que encontre uma sub-região de tamanho aceitável. Ao encontrá-la, ele volta às outras sub-regiões para dividi-las utilizando o mesmo processo. Ao final, tem-se o mapa dividido em regiões de tamanho similar;

2. A primeira fase metaforiza um processo recursivo linear em que cada região da montanha é como uma chamada à mesma função. Apesar de serem parecidas, elas não são a mesma, o que denota que são instâncias similares mas distintas. A disposição inicial dos elementos gráficos e dos inimigos é o maior indício disso. O topo da montanha consiste em um caso base, pois é a primeira região em que o problema da fase pôde ser resolvido. Na sequência, as outras regiões vão seguidamente podendo ser resolvidas também, até que todas sejam trabalhadas e o jogador é apresentado ao resultado final (abertura da passagem para a caverna);

3. A segunda fase metaforiza uma busca recursiva por um nó em uma árvore, que no jogo sempre é de ordem dois e altura oito. Ela é representada pelo labirinto que o jogador deve atravessar em busca do nó de saída. Com auxílio na metodologia de busca da saída, pode-se traçar um paralelo ao conhecido algoritmo *backtracking*, que serve também para resolução de labirintos. A placa marcadora que o jogador usa é justamente a marcadora de galhos e folhas já visitadas e da altura do nó. Ao encontrar a solução, basta ao jogador seguir as placas na ordem inversa, caso queira chegar ao início do labirinto;

**Figura 4 -** Passos para a divisão do mapa.



**Fonte:** autor

4. Na terceira fase, o processo é similar ao da primeira, a diferença fica por conta da clara exibição de como os resultados conseguidos em uma instância recursiva são carregados para as próximas. Isso se dá à característica do fantasma de ir ficando invisível quando os fantasmas anteriores vão morrendo juntamente com o fato de que ele se torna mortal quando o seu superior é derrotado (Figura 5);

Além das metáforas enumeradas, alguns nomes e elementos fictícios presentes na história do jogo remetem a conceitos que de alguma forma estão ligados à teoria dos algoritmos recursivos. O nome *Arquipélago de Fractalis* e a própria geografia remetem a fractais, o nome *Ilha de Binárvore* e a fase do labirinto remetem a árvores binárias, e o nome *Montanha de Edadidnuforp*, que é 'profundidade' ao contrário, remete ao algoritmo de busca em profundidade.

**Figura 5 -** Regressão da força dos fantasmas exibida pelo seu grau de transparência.


**Fonte:** autor

*Forma de Utilização Proposta*

O jogo foi projetado para prover experiências fictícias aos jogadores que facilite a realização de um paralelo com processos e algoritmos recursivos, suprindo assim a conhecida deficiência de experiências reais de recursividade. Dessa forma, a maneira ideal de sua utilização é a disponibilização por parte do professor para os alunos antes que as aulas sobre o assunto sejam lecionadas.

Se os alunos experimentarem o jogo, o professor pode então comentar sobre essas experiências e fazer o paralelo corretamente em relação à matéria. Dessa forma, garante-se a formação de metáforas que sejam corretas nas mentes dos alunos. Futuramente, quando eles se depararem com esse tipo de problema, poderão lembrar como foi a experiência de jogo para se guiarem à encontrar a solução, ou a melhor forma de implementar uma.

## Conclusão

Neste trabalho foi proposto e produzindo um jogo com capacidade de prover metáforas válidas em relação à teoria da recursividade. Ele tem potencial para ser atraente para os alunos dos cursos de Informática, devido às características audiovisuais e interativas. O produto do trabalho foi a implementação do jogo *Kobolds foram capturados!!!*.

Através de situações aparentemente normais de jogo, apresentam-se ao jogador quatro

metáforas, projetadas de tal forma a cobrir todas as especificidades da problemática da metaforização da recursividade na computação. Além disso, alguns elementos do jogo possuem nomes sugestivos que podem ajudar na fixação do aprendizado.

A capacidade de atração do jogo mostrou-se promissora, vários alunos se interessaram ao saber da existência e possibilidade de jogar em plena faculdade. Percebeu-se também a satisfação dos que puderam testar o jogo.

## Trabalhos Futuros

Em vista do que foi alcançado, é possível definir novos pontos de partida para melhorias sobre este trabalho. Dentre elas, a que desponta como mais importante e imediata é a aplicação de testes objetivos que avaliem a efetividade da ferramenta.

A ideia é criar um guia de utilização do jogo como ferramenta de ensino, e passa-lo juntamente com o jogo a um professor que fosse lecionar aulas de recursividade para a ferramenta em uma de suas turmas. A partir dessa aplicação, seria possível monitorar as notas das turmas nos exames de conhecimento, comparando as turmas que tiveram aulas convencionais com as que tiveram suporte da ferramenta.

## Referências


BARNES, T. et al. Game2learn: improving the motivation of cs1 students. In: INTERNATIONAL CONFERENCE ON GAME DEVELOPMENT IN COMPUTER SCIENCE EDUCATION, 3., 2008, New York. *Proceedings*... New York: ACM, 2008. p. 1–5.

BECKER, K. Teaching with games: the minesweeper and asteroids experience. *Journal of Computing Sciences in Colleges*, Shelbyville, v. 17, n. 2, p. 23-33, 2001.

CORMEN, T. H. et al. *Introduction to algorithms*. London: MIT Press, 2001.

EL-NASR, M. S.; SMITH, B. K. Learning through game modding. *ACM*: computers in entertainment, New York, v. 4, n. 1, p. 7, 2006.

GIGUETTE, R. Pre-games: games designed to introduce cs1 and cs2 programming assignments. ACM SIGCSE Bull, New York, v. 35, n. 1, p. 288-292, 2003.

GOLDWASSER, M.; LETSCHER, D. Teaching strategies for reinforcing structural recursion with lists. In: CONFERENCE ON OBJECT-ORIENTED PROGRAMMING SYSTEMS AND APPLICATIONS COMPANION, 22., 2007, New York. *Proceedings*... New York: ACM, 2007. p. 889–896.

GORDON, A. Teaching recursion using recursively-generated geometric designs. *Journal of Computing Sciences in Colleges*, Shelbyville, v. 22, n. 1, p. 124-130, 2006.

HABERMAN, B.; AVERBUCH, H. The case of base cases: why are they so difficult to recognize student difficulties with recursion. In: ANNUAL CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION, 7., 2002, New York. *Proceedings*... New York: ACM, 2002. p. 84-88.

HADEN, P. The incredible rainbow spitting chicken: teaching traditional programming skills through games programming. In: AUSTALIAN CONFERENCE ON COMPUTING EDUCATION, 8., 2006, Darlinghurst. *Proceedings*... Darlinghurst: Australian Computer Society, 2006. p. 81-89.

JIMÉNEZ-PERIS, R.; KHURI, S.; PATIÑO-MARTÍNEZ, M. Adding breadth to cs1 and cs2 courses through visual and interactive programming projects. In: SIGCSE TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 30., 1999, New York. *Proceedings*... New York: ACM, 1999. p. 252-256.

KAY, J. S. Using the force: how star wars can help you teach recursion. *Journal of Computing Sciences in Colleges*. Mahwah, v. 15, n. 5, p. 274-284, 2000.

SANDERS, I.; GALPIN, V.; GÖTSCHI, T. Mental models of recursion revisited. In: ANNUAL SIGCSE CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION, 11., 2006, New York. *Proceedings*... New York: ACM, 2006. p. 138-142.

SCHUYTEMA, P. Design de games: uma abordagem prática. [S.l.]: CENGAGE Learning, 2008.

SUNG, K. et al. Assessing game-themed programming assignments for cs1/2 courses. In: INTERNATIONAL CONFERENCE ON GAME DEVELOPMENT IN COMPUTER SCIENCE EDUCATION, 3., 2008, New York. *Proceedings*... New York: ACM, 2008. p. 51-55.

VELÁZQUEZ-ITURBIDE, J. N.; PÉREZ-

CARRASCO, A.; URQUIZA-FUENTES, J. Srec: an animation system of recursion for algorithm courses. In: ANNUAL CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION, 13., 2008, New York. *Proceedings...* New York: ACM, 2008. p. 225-229.

WIRTH, M. Introducing recursion by parking cars. ACM SIGCSE Bull, New York, v. 40, n. 4, p. 52-55, 2008.

WONG, W. L. et al. Serious video game effectiveness. In: INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTER ENTERTAINMENT TECHNOLOGY, 2007, New York. *Proceedings...* New York: ACM, 2007. p. 49-55.

XU, D.; BLANK, D.; KUMAR, D. Games, robots, and robot games: complementary contexts for introductory computing education. In: INTERNATIONAL CONFERENCE ON GAME DEVELOPMENT IN COMPUTER SCIENCE EDUCATION, 3., 2008, New York. *Proceedings...* New York: ACM, 2008. p. 66-70.

YANG, F.-J. Another outlook on linear recursion. ACM SIGCSE Bull, New York, v. 40, n. 4, p. 38-41, 2008.

*Recebido em 25 Julho 2011- Received on July 25, 2011.*
*Aceito em 25 Julho, 2012 - Accepted on July 25, 2012.*