Considerações sobre a Utilização de Linguagens Funcionais como Instrumento de Aproximação entre Educadores e Jogos Educativos

ANDRÉ WILSON BROTTO FURTADO
ANDRÉ LUÍS DE MEDEIROS SANTOS
ALEX SANDRO GOMES

Universidade Federal de Pernambuco (UFPE) – Centro de Informática (CIn)
Av. Prof. Luís Freire, s/n, Cidade Universitária, CEP 50740-540 – Recife/PE/Brazil
{awbf,alms,asg}@cin.ufpe.br

Resumo

Este artigo analisa a etapa de implementação de jogos educativos à luz das vantagens e limitações de linguagens funcionais. O objetivo é validar a combinação desses dois conceitos para diminuir a distância entre educadores e as tecnologias necessárias ao desenvolvimento de jogos educativos, permitindo a elaboração de um produto final de maior qualidade técnica e pedagógica.

Palavras-chave: *linguagens funcionais, jogos educativos, implementação.*

1 Introdução

Algumas décadas atrás, o surgimento de uma nova categoria de software, cujo objetivo era dar suporte a indivíduos ou grupos de indivíduos no processo de aquisição de conhecimento em relação a um domínio específico, promoveu o nascimento de um novo paradigma de desenvolvimento de software. Enquanto as abordagens tradicionais partiam do princípio de que a intenção do usuário, ao utilizar um software, era melhorar seu desempenho na execução de uma determinada tarefa, o surgimento de softwares educativos necessitou de abordagens que avaliassem os objetivos do usuário em termos de aprendizado, e não apenas de usabilidade ou melhoria de desempenho [1].

Não é exagero afirmar que, com o surgimento do conceito de software educativo, foi despertado um potencial capaz de promover um completo movimento de reforma educacional [2]. Uma instância particular de software educativo, conhecida como "jogos educativos", reforça essa idéia. Partindo do princípio de Hawthorn [6], que afirma que todo objeto em observação tende a mudar seu comportamento simplesmente pelo fato de estar sendo observado, muitos autores defendem a idéia de que a eficiência de um software educativo é maior quando o aprendiz não tem a noção de que está fazendo parte de um processo de aprendizado. Jogos educativos contribuem para a viabilização desse conceito na prática, pois possuem um objetivo inerente de criar mecanismos de imersão que possam envolver o jogador em um mundo à parte, focando-o na realização bem-sucedida dos desafios proporcionados pelo jogo.

Apesar da era da informação prover formidáveis subsídios tecnológicos para que a educação explore proveitosamente os benefícios oferecidos pela informática, como é o caso de jogos educativos, o cenário em que se apresenta a integração pedagógico-computacional é bastante diferente na prática. Na década de 90, Carraher [3] já apresentava uma visão pessimista em relação ao futuro do software educativo. Segundo o autor, "a produção de software de qualidade técnica e, mais ainda, de qualidade pedagógica, é mais complexa do que se imaginava, de tal modo que dificilmente surgirá, de forma rápida e espontânea uma quantidade de software de qualidade". Por trás de tal insucesso, esconde-se o fato de que muita ênfase é dada à avaliação técnica e computacional de software e jogos educativos,

enquanto a mesma deveria estar subordinada à avaliação educacional e pedagógica. Em outras palavras, a transformação do potencial pedagógico-computacional em aprendizado efetivo não pode ter como base apenas o avanço tecnológico.

Este artigo acredita que linguagens funcionais podem contribuir significativamente para o aumento da integração pedagógico-computacional, ao viabilizar uma agilização e simplificação da etapa de implementação de jogos educativos. Deste modo, educadores são mais bem abstraídos da tecnologia utilizada e podem concentrar seus esforços no conteúdo pedagógico do jogo. O trabalho revela que as vantagens oferecidas por linguagens funcionais adequam-se eficientemente às necessidades de implementação de jogos educativos, enquanto suas limitações não representam um desafio concreto para jogos deste tipo.

Este artigo, portanto, está estruturado da seguinte maneira: a seção 2 apresenta um estudo mais aprofundado sobre as atuais deficiências em jogos educativos. A seção 3, por sua vez, introduz o conceito de linguagens funcionais, realizando um estudo crítico sobre suas vantagens e limitações. A seção 4 confronta diferentes estudos com o objetivo de validar a utilização de linguagens funcionais no processo de implementação de jogos educativos. A seção 5, por fim, realiza conclusões acerca do trabalho apresentado.

2 Deficiências de Jogos Educativos

A definição de novas abordagens de implementação para jogos educativos necessita da identificação dos pontos críticos atualmente existentes no desenvolvimento de jogos deste tipo. Em outras palavras, procurou-se identificar quais os principais responsáveis pelo insucesso da integração pedagógico-computacional e como a fase de implementação poderia ser modificada para combater tais deficiências.

Mais de dez anos após os estudos de Carraher, apresentados na seção anterior, suas teorias são concretizadas por pesquisas recentes envolvendo o público-alvo de jogos educativos. Clua, Luca e Nabais [8] conduziram uma série de entrevistas com jogadores na faixa etária de 10 a 17 anos, com o objetivo de identificar o que eles pensam e esperam de jogos educativos. Os resultados retratam um cenário bastante pessimista em relação ao tema: nenhum dos jogadores entrevistados afirmou que existem jogos educativos "excelentes", ao passo que 68% deles consideraram todos os jogos educativos já experimentados como sendo "terríveis". Uma pequena parcela dos entrevistados (5%) se declararam indiferentes em relação ao tema em discussão e apenas 27% dos jogadores afirmaram que já jogaram pelo menos um jogo educativo que pudesse ser considerado como "bom". Os autores da pesquisa identificaram quatro motivos principais para explicar tal resultado:

1) Jogos educativos possuem deficiências em apresentar desafios complexos e interessantes;
2) O jogador de jogos educativos não é envolvido o suficiente por jogos deste tipo;
3) A maioria dos jogos educativos é desenvolvida apenas por pedagogos, que tomam cuidado excessivo com o tópico de aprendizagem e deixam a diversão do jogo em segundo plano;
4) Não há um investimento expressivo em projetos de jogos educativos, na maioria das vezes porque *publishers* não consideram esta área um negócio rentável.

As conclusões da pesquisa realizada deixam claro que a aceitação de jogos educativos é um problema bastante crítico. As conclusões 1, 2 e 3, quando consideradas em conjunto, revelam que educadores não possuem a intimidade tecnológica necessária para conseguir diferenciar um exercício pedagógico de um jogo propriamente dito. Tal visão também é compartilhada por Hubbard [6], que atrela a definição de jogos educativos à capacidade dos mesmos em proporcionar desafios imediatos e interessantes (ou seja, diversão), o que não é exigido para exercícios pedagógicos.

Outros estudos na área também convergem para a realidade de que uma das principais razões para o insucesso de jogos educativos consiste na distância que separa educadores da tecnologia utilizada no desenvolvimento de jogos educativos. Tal distância, ilustrada pela

Figura 1, promove relutância dos educadores à utilização de jogos como meio de aprendizado e, o que é pior, um sub-aproveitamento dos mesmos no próprio processo de desenvolvimento de jogos educativos. Tchounikine [4], por exemplo, ressalta a dificuldade de interação entre programadores e educadores, devido à não-trivialidade de compartilhar conceitos das diferentes áreas. Mandel [5], por sua vez, identifica problemas causados pela significativa diferença que *designers*, programadores e professores possuem em relação aos processos de ensino e aprendizagem.

Figura 1. Metáfora representando a distância entre educadores e jogos educativos.

Muitas abordagens têm sido utilizadas para diminuir a distância entre educadores e jogos educativos. Algumas destas abordagens, entretanto, dependem da capacidade do educador em enriquecer sua experiência e conhecimento tecnológico, de modo a atingir o patamar de complexidade exigido por um jogo educativo. Hadfield, Maddux e Love [9], por exemplo, constatam que a proliferação da tecnologia computacional nas escolas não foi acompanhada por um crescimento similar de mecanismos de instrução para o uso dessa tecnologia. Como resultado, mesmo quando a grande maioria dos professores tem acesso à tecnologia computacional em sala de aula, menos da metade deles integram o uso do computador no currículo escolar.

Abordagens deste tipo (ilustradas pela metáfora da Figura 2), portanto, não são as mais adequadas, pois dependem da disposição e competência do educador em se familiarizar com conhecimentos tecnológicos com os quais, geralmente, não possui intimidade. Em resumo, a opção de delegar ao educador a tarefa de adquirir o conhecimento necessário à utilização ou elaboração de jogos educativos é um sério entrave à qualidade final da integração pedagógico-computacional.

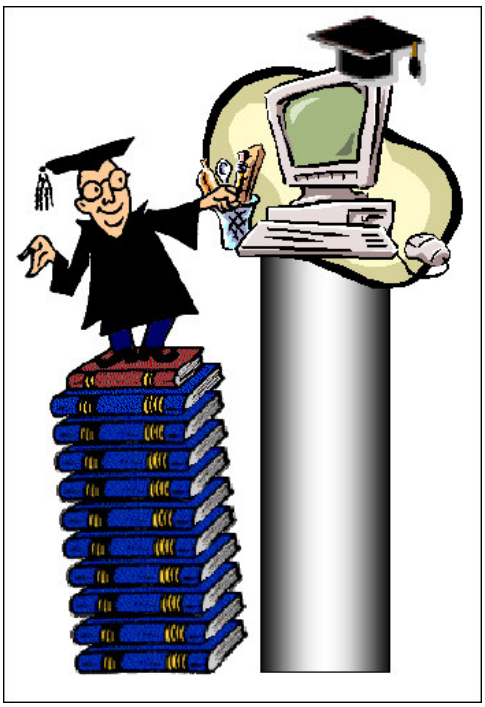

Figura 2. Redução da distância entre educadores e jogos educativos através do aumento do *expertise* tecnológico do educador.

Tendo em mente os problemas apresentados acima, este artigo procura seguir uma estratégia diferente: diminuir a distância (e a resistência) entre educadores e jogos educativos através da redução do patamar de complexidade tecnológica para a elaboração de jogos educativos, e não através do aumento do *expertise* tecnológico do educador. Este tipo de abordagem é ilustrado pela Figura 3.

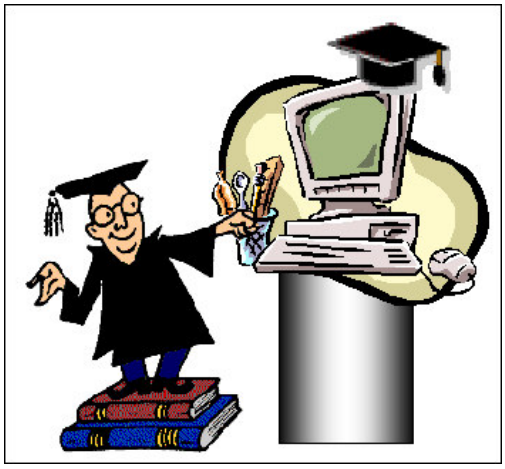

Figura 3. Redução da distância entre educadores e jogos educativos através da diminuição do patamar de complexidade tecnológica para a elaboração de jogos educativos.

Acredita-se que a etapa de implementação, por envolver tecnologias não triviais que exigem habilidades mais específicas (como linguagens de programação), é um dos principais responsáveis pela elevada dimensão do patamar de complexidade tecnológica na elaboração de um jogo educativo. Este artigo, portanto, visa agilizar e simplificar tal etapa, através da utilização de linguagens funcionais, diminuindo a necessidade do educador em adquirir conhecimentos tecnológicos complexos para a elaboração de um jogo educativo. Essa simplificação, entretanto, ainda permite que o educador possa inserir conteúdo pedagógico no jogo, garantindo assim a qualidade final no aprendizado do usuário.

Antes de apresentar as implicações na combinação entre linguagens funcionais e jogos educativos, é necessário compreender como está situado este tipo de linguagem em relação às linguagens de programação em geral. A seção a seguir apresenta uma introdução a linguagens funcionais, realizando um rápido estudo crítico entre linguagens funcionais e outros tipos de linguagem, ressaltando suas vantagens e limitações. A intenção final não é realizar uma comparação exaustiva entre linguagens funcionais e outros paradigmas, e sim apresentar fundamentos para embasar a viabilidade de linguagens funcionais para a implementação de jogos educativos.

3 Vantagens e Limitações de Linguagens Funcionais

C, Java[1] e Pascal são linguagens de programação *imperativas*, no sentido de que programas construídos nessas linguagens consistem em uma seqüência de comandos, executados estritamente um após o outro. Um programa feito em uma linguagem funcional, por outro lado, é uma única expressão. A execução de um programa funcional é na verdade a própria avaliação da expressão .

Uma planilha eletrônica, em que o valor de uma célula é computado em termos de outras células, exemplifica bem o conceito de linguagens funcionais [10]. Em programas deste tipo, não é especificada a ordem em que as células são calculadas; espera-se que a planilha eletrônica compute as células em uma ordem que respeite suas dependências. Além disso, não é dito ao programa como e quando a memória será alocada; espera-se que a planilha, apesar de exibir um plano aparentemente infinito de células, aloque memória apenas para as células em uso. Por fim, o valor de uma célula é especificado por uma expressão, e não por uma seqüência de comandos que computa o seu valor.

Uma das principais características de linguagens funcionais, portanto, consiste na abstração oferecida ao programador, que interage com a linguagem especificando em alto-nível "o que" deve ser feito, e não "como". Linguagens imperativas, por outro lado, procuram atenuar o problema da menor intuitividade de programação, provocada pela seqüencialização de comandos, através da introdução de novas palavras chaves (**while**, **for**, **foreach**, etc.) ou de bibliotecas. Entretanto, linguagens imperativas foram concebidas pela filosofia de seqüencialização de comandos, impedindo que esse conceito seja completamente eliminado destes tipos de linguagem. Desse modo, linguagens imperativas

[1] Alguns autores consideram linguagens orientadas a objeto, como Java e C#, um subconjunto das linguagens imperativas. Outros autores, entretanto, preferem separar os dois conceitos. Neste artigo, utilizaremos a primeira abordagem.

não conseguem atingir o mesmo nível de abstração proporcionado por linguagens funcionais [11].

Linguagens funcionais também oferecem as principais vantagens de linguagens orientadas a objeto. O mecanismo de encapsulamento de dados, por exemplo, é oferecido por linguagens funcionais através de tipos abstratos de dados (ADT ou *abstract data types*), enquanto o polimorfismo é proporcionado por "classes de tipo" (*class types*).

Por fim, as vantagens de linguagens funcionais podem ser resumidas em:

- Maior concisão: códigos feitos em linguagens funcionais são de 2 a 10 vezes menores do que códigos feitos em linguagens imperativas;
- Maior elegância e facilidade de compreensão: o *gap* semântico entre o programador e a linguagem é menor no caso de linguagens funcionais;
- Ausência de *core dumps*: a linguagem é "fortemente tipada", de modo que é impossível tratar um inteiro como um ponteiro ou haver referências nulas;
- Reuso de código (através de polimorfismo);
- *Lazy evaluation*: as expressões são avaliadas sob demanda, isto é, nenhuma parte desnecessária de código é avaliada durante a computação de um resultado;
- Alta abstração e modularização: o conceito de "funções de alta-ordem", por exemplo, permite que funções sejam passadas como argumentos para outras funções, retornadas como resultado de outras funções ou armazenadas como parte de uma estrutura de dados;
- Gerenciamento automático de memória (através do mecanismo de coleta de lixo, ou *garbage collection*).

Programas feitos em linguagens funcionais, entretanto, tendem a ter pior performance quando comparados a programas similares feitos em outros tipos de linguagens. Contudo, este problema não constitui um empecilho real para aplicações em que a performance não é absolutamente crítica. Wadler [12], em um artigo que, inclusive, defende o uso de C/C++ e questiona o uso de linguagens funcionais, cita que a questão da performance definitivamente não pode ser usada como argumento para justificar a não-utilização de linguagens funcionais. Adicionalmente, Furtado e Santos [13] constataram que as performances de duas versões de um mesmo jogo não muito complexo, uma desenvolvida na linguagem funcional Haskell e outra na linguagem C++, tendem a se igualar quando uma configuração razoável de *hardware*[2] está presente.

Entretanto, algumas deficiências em linguagens funcionais ainda restringem sua aplicação a outras áreas, como jogos educativos, por exemplo. O conjunto de bibliotecas que adicionam recursos a linguagens funcionais ainda é limitado. O processo de instalação de compiladores e outras ferramentas que dão suporte a este tipo de linguagem, por sua vez, precisa se tornar mais simplificado. Outro problema consiste à natureza isolacionista das linguagens funcionais, isto é, sua compatibilidade com outras tecnologias é de certa forma limitada. Atualmente, esforços estão sendo realizados com o objetivo de superar tal deficiência, como a integração de linguagens funcionais à tecnologia COM, por exemplo [14].

Adicionalmente, a própria popularidade das linguagens funcionais consiste em um problema, visto que elas são menos conhecidas do que linguagens como C, C++ e Java. O fato da sintaxe de linguagens funcionais ser bastante diferente do padrão apresentado por estas outras linguagens pode ter um impacto negativo na curva de aprendizado de programadores que desejem migrar de um paradigma para o outro.

Por fim, para aplicações cuja performance é crítica, outros tipos de linguagens são mais adequados do que linguagens funcionais. Por exemplo, a construção de uma aplicação de controle de tráfego aéreo, em tempo real, para uma região de movimentação muito intensa pode abrir mão da produtividade de linguagens funcionais em troca da garantia de um tempo mínimo de execução para suas rotinas.

---

[2] No caso, um Pentium III de 900Mhz e 256 MB de RAM.

4 Integrando Linguagens Funcionais a Jogos Educativos

Entendido o contexto no qual se situam linguagens funcionais, faz-se necessário analisar como maximizar sua aplicação no processo de desenvolvimento de jogos educativos.

O conceito de funções matemáticas, fundamento através do qual estão embasadas linguagens funcionais, é mais natural de se explicar, adquirir e utilizar, principalmente por indivíduos que não possuem intimidade com linguagens de programação, como é o caso de educadores. A noção de variáveis, orientação a objetos e até mesmo lógica exige um conhecimento um pouco maior e, portanto, é um tanto mais inacessível para educadores.

Entretanto, apesar da fundamentação da programação funcional ser mais simples e matemática, conseqüentemente mais acessível, usar apenas isso como argumento para reduzir o esforço da participação de educadores na criação de jogos educativos não é satisfatório. Mesmo para educadores da área de ciências exatas, o embasamento matemático de linguagens funcionais não é suficiente para que a distância entre o conhecimento do educador e o patamar de complexidade tecnológica da elaboração do jogo diminua, a não ser que o educador se especialize em linguagens funcionais. Uma abordagem desse tipo, portanto, iria cair no caso da Figura 2, em que educadores e jogos educativos são aproximados através do aumento do *expertise* do educador. Como apresentado anteriormente, abordagens deste tipo tendem a falhar devido à resistência do educador em ser obrigado a se familiarizar com tecnologias com as quais não possui intimidade.

Uma análise mais cautelosa, portanto, precisou ser realizada para validar abordagens de implementação que aumentem a tão referida integração pedagógico-computacional. Tal estudo procurou atacar o processo de desenvolvimento de jogos educativos à luz das vantagens oferecidas por linguagens funcionais.

Inicialmente, procurou-se identificar se as limitações de linguagens funcionais são obstáculos concretos para a utilização de linguagens deste tipo na implementação de jogos educativos, indiferentemente ao *expertise* do educador. Apenas quando esta dúvida fosse esclarecida seria possível partir para considerações finais que incorporassem o educador no processo de desenvolvimento.

Um estudo sobre alguns casos de sucesso de utilização de jogos educativos revelou que tais jogos podem atingir seus objetivos pedagógicos mesmo sendo bastante simples. Um trabalho recente realizado por Rosas et al. [16], por exemplo, registrou uma melhoria de aprendizado em crianças de escolas chilenas após a introdução, na sala de aula, de jogos bastante simples executados em aparelhos idênticos às primeiras versões do Gameboy da Nintendo [17]. Estes aparelhos possuíam um pequeno *display* LCD, de 160 x 144 pixels, e apenas 4 tons de cores. Um exemplo de jogo nesse aparelho é apresentado na Figura 4.

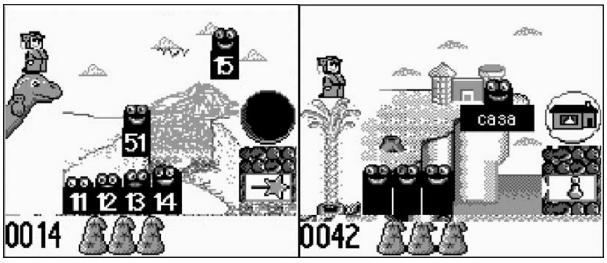

Figura 4. "Magalú", um dos jogos educativos do estudo de Rosas et al.

Apesar da simplicidade dos jogos educativos utilizados, como uma variação do "jogo da forca" (*hangman*), em que palavras devem ser construídas de acordo com os ícones que aparecem na tela, os resultados do trabalho revelaram uma significativa melhora dos alunos em Matemática, Leitura, Interpretação e Ortografia (*Spelling*).

Cruzando os resultados obtidos por Rosas et al. (2003) com os trabalhos de Furtado e Santos (2002) referentes à validação de linguagens funcionais para jogos não muito complexos, acredita-se que limitações de performance deste tipo de linguagem não constituem um desafio concreto para sua aplicação em jogos educativos.

Adicionalmente, linguagens funcionais já oferecem mecanismos direcionados especificamente à agilização do processo de desenvolvimento de jogos. A linguagem

funcional Clean [18], por exemplo, oferece a biblioteca CGL (*Clean Game Library*) [19], enquanto a linguagem funcional Haskell tem disponível o motor para jogos FunGEn (Functional Game Engine) [20]. O fato da CGL e do FunGEn estarem integrados a duas das mais bem conceituadas bibliotecas gráficas atualmente existentes (DirectX no caso da CGL e OpenGL no caso do FunGEn) revela que outros problemas relacionados a linguagens funcionais, como o isolacionismo e a pouca integração com tecnologias externas, estão realmente sendo combatidos na prática.

Resta, portanto, resolver um dos obstáculos mais críticos em relação à integração de linguagens funcionais com jogos educativos: como permitir que o educador, em suas limitações tecnológicas, possa tirar proveito das vantagens oferecidas por linguagens funcionais? Os autores deste artigo acreditam que linguagens de domínio específico (DSL ou *domain-specific languages*) e sua adequada integração com linguagens funcionais são a resposta para tal problema.

Ao contrário das linguagens de propósito geral (GPL ou *general-purpose languages*), como Haskell, C++ ou Java, que são capazes de criar diversos tipos diferentes de programas, linguagens de domínio específico são linguagens de programação utilizadas para abordar um problema ou domínio particular [23]. Por exemplo, os *shells* do Unix podem ser considerados DSLs, pois se utilizam de notações e abstrações (comandos, *streams* e operadores de *streams*, etc.) em um domínio bem-definido[3].

Assim como Reinke [24] afirma, sem boas notações para uma linguagem, provas se tornam complicadas, idéias permanecem inacessíveis, software permanecem não-implementados e computadores, inutilizados. DSLs provêem justamente um maior nível de abstração, notações e formulações declarativas, permitindo uma programação mais intuitiva, um reuso de código mais sistemático e uma maior facilidade para a verificação de propriedades de programas. Quando bem projetadas, DSLs podem se aproximar da própria linguagem natural. No caso de jogos educativos, a utilização de DSLs é uma alternativa que consegue reduzir o patamar de complexidade tecnológica na elaboração de jogos educativos, aproximando-o do educador, como apresentado na Figura 3.

Estudos do próprio Reinke [24] revelam que linguagens funcionais são uma excelente opção para a criação de DSLs, principalmente quando comparadas a outros tipos de linguagem. Algumas vantagens citadas pelo autor incluem:

- A abstração suportada por linguagens funcionais permite pensar a criação de uma DSL em termos de *composição de pequenas linguagens*;
- A linguagem nativa (funcional) geralmente não interfere no resultado final da DSL;
- Tipos algébricos de dados podem ser utilizados para gramáticas abstratas;
- Combinadores de *parsers* podem ser utilizados para gramáticas concretas;
- Interpretadores baseados em semântica facilitam o processo de geração da DSL;
- DSLs embutidas em linguagens funcionais herdam um rico *framework* de características de linguagens genéricas, como abstração, recursão, etc.

Peterson, Serjantov e Hudak [25] ressaltam, ainda, que a existência de funções de alta-ordem em linguagens funcionais permite a captura de abstrações complexas de maneira reusável, além do fato de que a natureza declarativa de linguagens funcionais torna intuitiva a tarefa de descrição de DSLs. Os autores utilizaram a linguagem funcional Haskell para desenvolver a linguagem "Frobocup", uma DSL para programar robôs jogadores de futebol.

Muitos outros trabalham validam o uso de linguagens funcionais para a criação produtiva de DSLs. Em especial, destacam-se a DSL "Fran" [26], para animações interativas envolvendo som e gráficos bi e tridimensionais, a "Pan" [27], para manipulação e síntese de imagens, a "FAL" (*Functional Animation*

[3] Exemplos de *streams* em *shells* do Unix são o "*stdin*" e o "*stdout*". Operadores nesses *streams*, por sua vez, incluem o *pipe* (|) e o operador de redirecionamento (>).

Language) [28] e a MDL (*Music Description Language*) [28].

Diante de tantos estudos que comprovam a aplicabilidade de linguagens funcionais na criação de DSLs de diversos domínios diferentes, acredita-se que DSLs específicas para jogos educativos podem agilizar a etapa de implementação dos mesmos. Isto envolve um trabalho adicional de elicitar, com educadores de diferentes áreas, suas necessidades e sua linguagem pedagógica, de modo a criar uma DSL que o deixe o mais confortável possível em sua utilização. A Figura 5 ilustra, portanto, a abstração apresentada ao educador com a introdução de DSLs no processo de implementação.

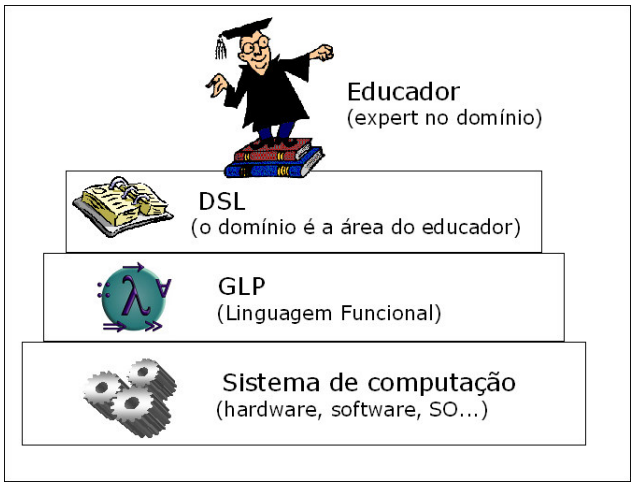

Figura 5. Abstraindo a complexidade de implementação do educador através de DSLs.

Obviamente, a construção da DSL não será função do educador, e sim de um cientista da computação. Isto vai de acordo com as idéias defendidas por Reinke:

- O *expert* no domínio deve focar em utilizar suas linguagens para lidar com seus problemas específicos;
- O cientista da computação deve focar na criação de linguagens de suporte através de sistemas de computação.

5 Conclusões

O trabalho apresentado neste artigo revelou que as vantagens oferecidas por linguagens funcionais atendem de maneira adequada às necessidades de implementação de jogos educativos. Entretanto, algumas limitações deste tipo de linguagem, como dificuldades no processo de instalação de compiladores e outras ferramentas, ainda não foram superadas e necessitarão de um cuidado extra no futuro.

Apesar de não apresentar uma abordagem completamente bem-definida para o desenvolvimento de jogos educativos através de linguagens funcionais, este artigo apresentou uma série de considerações que servirão como um guia para abordagens mais concretas.

Muitas possibilidades existem para o futuro. Uma sugestão é criar DSLs diretamente a partir do FunGEn, da CGL ou de outro mecanismo que já ofereça facilidades na implementação de jogos. Outra possibilidade consistiria em implementar um motor específico para jogos educativos, partindo apenas de uma linguagem funcional, como Haskell. Por fim, muitos trabalhos de interface homem-máquina podem ser realizados no sentido de criar uma DSL visual para o educador, facilitando ainda mais sua aceitação ao processo de desenvolvimento de jogos educativos.

6 Referências

1. Soloway, E., Guzdial, M., Hay, K.E. Learner-Centered Design: The Challenge for HCI in the 21st Century, Interactions, Vol.1, No.2, April, 36-48, 1994.

2. Ganss, A. D. A nation at risk: the crisis of the American educational mythos and the convergence of cynical political ideologies, Center For Education of Widener University, 1999.

3. Carraher, D.W. "O que esperamos do Software Educacional?" Acesso – Revista de Educação e Informática, Ano II, n° 3, jan./jun. 1990, issn 0103-0736, 1990.

4. Tchounikine P. "Pour une ingénierie des Environnements Informatiques pour l'apprentissage

humain", Information-Interaction-Intelligence, vol. 2, n. 1, 2002.

5. Mandel T., The Elements of user interface, John Wiley and Sons: New York, 1997.

6. The Hawthorne effect: a note, http://www.psy.gla.ac.uk/~steve/hawth.html (01/08/2003).

7. Hubbard, Evaluating computer games for language learning. *Simulation and Gaming* ,22 ,220 –223, 1991.

8. Clua, E. W. G., Luca Jr., C. L., Nabais, R. J. M. "Importance and Impacts of Educational Games in Actual Society", WJogos'2002.

9. Oakley D Hadfield, Cleborne D. Maddux, Glendel D. Love. Critical thinking ability and prior experiences as predictors of reduced computer aversion. Educational Administration Abstracts, Volume 34, Number 2, 1999.

10. Haskell.org: About Haskell, http://www.haskell.org/aboutHaskell.html (02/08/2003).

11. The Lambda Complex: Why does Haskell matter?, http://www.haskell.org/complex/why_does_haskell_matter.html (02/08/2003).

12. Wadler, P. Why no one uses functional languages. ACM SIGPLAN Notices, August 1998.

13. Furtado, A. W. B., Santos, A. L. M. FunGEn - A Game Engine for Haskell, WJogos'2002.

14. Peyton-Jones S., Meijer E., Leijen D. Scripting COM components in Haskell. IEEE Fifth International Conference on Software Reuse, Vancouver, BC, 1998.

15. Rahn, M., Waldmann J. The Leipzig autotool System for Grading Student Homework

16. Rosas R. et al. Beyond Nintendo:design and assessment of educational video games for first and second grade students. Computers & Education, ed. 40 pp. 71 –94, 2003.

17. Gameboy.com, http://www.gameboy.com/ (04/08/2003)

18. The Clean Programming Language, http://www.cs.kun.nl/~clean/ (05/08/03)

19. The Clean Game Library, http://cleangl.sourceforge.net/ (05/08/2003)

20. Functional Game Engine http://www.cin.ufpe.br/~haskell/fungen/ (05/08/2003)

21. DirectX Home Page, http://www.microsoft.com/directx/ (05/08/2003)

22. OpenGL – The Industry Foundation for High Performance Graphics, http://www.opengl.org/ (05/08/2003)

23. Domain-Specific Languages - An Overview, http://216.239.51.104/search?q=cache:t4uSaKg2kCAJ:compose.labri.fr/documentation/dsl/dsl_overview.php3&hl=pt&ie=UTF-8 (05/08/2003)

24. Reinke C., Domain-specific languages and functional programming. Computing Laboratory, University of Kent at Canterbury, 2003.

25. Peterson J., Serjantov A., Hudak P. Frobocup, a DSL for Robotic Soccer. Yale University, www.cl.cam.ac.uk/~aas23/talks_aas/ibm_talk3.ppt (05/08/2003).

26. Functional Reactive Animation, http://www.conal.net/fran/ (05/08/2003)

27. The Pan Home Page, http://www.conal.net/Pan/ (05/08/2003)

28. The Haskell School of Expression, http://haskell.cs.yale.edu/soe/ (05/08/2003)