JIndie: Uma Linha de Produto de Software para Jogos Educativos com Foco no Construcionismo

Carlos A. C. Lessa Filho[1], Arturo Hernández-Domínguez[1]

[1] Instituto de Computação - Universidade Federal de Alagoas (UFAL)
Maceió – AL – Brasil

carloswgama@gmail.com, arturohd@ic.ufal.br

Abstract. *The games in the teaching process have proved an interesting tool to contribute with the motivation and student learning. This paper aims to facilitate the development of these educational games. To do this, we developed a Software Product Line based on constructionist games. This Product Line was used to develop four games for its validation. Based on the results, it's possible to realize the advantages in using a Product Line in the development of educational games, such as, reduction in development time, simplification of code's complexity and directing of the creation of games.*

Resumo. *Os jogos no processo de ensino têm se demonstrado uma interessante ferramenta para contribuir com a motivação e aprendizado dos estudantes. Este trabalho tem como objeto facilitar o desenvolvimento desses jogos educativos. Para isto, foi desenvolvida uma Linha de Produto de Software com base em jogos construcionistas. Essa Linha de Produto foi utilizada na produção de quatro jogos. Baseando-se nos resultados, pode-se observar as vantagens no uso de uma Linha de Produto no desenvolvimento de jogos educativos como redução no tempo de desenvolvimento, simplificação da complexidade e direcionamento do processo de criação dos jogos.*

1. Introdução

Nos últimos anos, muitas pesquisas na educação têm sido realizadas com o foco em elevar a motivação dos estudantes para buscar novos conhecimentos. Várias dessas pesquisas têm apontado a diversão como um fator importante para alavancar a motivação dos alunos aos estudos [Yoon e Kim, 2015]. Atualmente, os jogos vêm ganhando importância na área de ensino, de forma que algumas instituições estão começando a adotá-los em sala de aula, com a intenção de despertar a motivação aos estudos que os estudantes não possuíam, como também possibilitar novas experiências, que antes eram difíceis sem o computador [Rezende et al 2013].

A abordagem tradicional de ensino entre professores e estudantes, onde apenas o professor transmite o conteúdo, enquanto os estudantes a decoram, também se demonstra defasada e desmotivadora, no qual cerca de 40% dos estudantes entre 15 e 17 anos que deixam os estudos, apontam a escola como algo desinteressante [Santos et al, 2015]. Papert e Harel (1991) desde antigamente defendiam a motivação no processo de ensino através do construcionismo, no qual o estudante para aprender um conteúdo por completo e ser motivado pelo conteúdo estudado, deve participar da construção de um material concreto. Papert ainda defende a importância dos jogos nesse processo, onde

DOI: 10.5753/cbie.sbie.2016.350

uma criança apresenta menos resistência ao conteúdo sendo apresentado em um jogo comparado ao conteúdo apresentado de forma tradicional em sala de aula.

Os jogos educativos podem contribuir bastante com o processo de resgate do interesse dos estudantes, de forma que o ideal seria que professores pudessem desenvolver seus próprios jogos [Tarouco et al, 2004]. Todavia, o processo de desenvolvimento de um jogo do zero é um processo demorado, caro, que pode envolver muitas pessoas. Para tentar contornar essa situação, o desenvolvedor pode optar pelo uso de uma Linha de Produto de Software (LPS). Uma LPS segundo Käköla e Leitner (2014) é uma metodologia validada industrialmente para o desenvolvimento de softwares visando menor custo, maior velocidade, qualidade e satisfação do usuário final.

Visando a importância desses jogos e a escassez de recursos para o desenvolvimento de jogos no contexto do construcionismo, este artigo apresenta uma proposta de uma LPS para o desenvolvimento de jogos construcionistas seguindo o modelo proposto por Pohl, Böckle e Linden (2005). A avaliação da LPS construída foi fundamentada no desenvolvimento de quatro jogos, sendo estes baseados em três jogos construcionistas.

2. Trabalhos Correlatos

Nessa seção iremos apresentar duas LPS, que foram selecionadas por também serem voltadas para o desenvolvimento de jogos ou aplicativos educativos.

A LPS Arcade Game Maker [Morin et al, 2008] foi criada pelo Software Engineering Institute (SEI) com o intuito de criar um exemplo de Linha de Produto para dar suporte a aprendizagem e experiência em Linha de Produtos de Software. A linha de produto conta com a presença de três jogos do gênero plataforma. Ao contrário da maioria dos jogos, os produtos do Arcade Game Maker não possuem foco na interface gráfica, pois o objetivo desta LPS é ser um exemplo abrangente. Seu material sofreu evoluções de pelo menos dois anos de produção e está sendo usado como exemplo ilustrativo em cursos.

A segunda LPS apresentada é a Linha de Produtos de Software para Módulos de Aprendizagem Interativa (iMA) [Dalmon e Brandão, 2013]. iMA são módulos educativos, como jogos, que são integrados a ambientes educacionais como o Moodle. A LPS de Dalmon e Brandão foi desenvolvida com a preocupação em reduzir as falhas e o tempo gasto para resolver tais problemas no desenvolvimento de iMA, que geram impactos diretos nos benefícios didáticos que esses módulos pretendiam fornecer. O desenvolvimento da LPS foi baseado nos iMA já existentes.

A análise dessas e outras LPS baseadas em aplicativos e jogos educativos auxiliaram no processo de definição dos recursos e métodos de implementação da LPS proposta. A LPS deste trabalho diferencia-se desses e outros trabalhos correlatos ao especializa-se nos recursos usados por jogos do domínio construcionista, que não dispõem de uma ferramenta especializada conhecida.

3. Construcionismo

O construcionismo de Seymour Papert foi baseado no construtivismo de Jean Piaget, que estudou as fases do conhecimento com uma visão biológica. Para Piaget, o

aprendizado ocorre de diferentes formas ao longo da vida do indivíduo, sendo dividida em quatro etapas: Sensório-motor (até 2 anos); Pré-operatório (até 7 anos); Operatório-concreto (até 12 anos); Operações formais (após os 12 anos) [Marlowe e Canestari, 2006].

Para Papert, o construcionismo não apenas compreende as fases do conhecimento proposto por Jean Piaget, como também considera que o sujeito deve construir um material concreto como, por exemplo, uma apresentação, um quadro ou um software. Assim, novos conhecimentos serão adquiridos através da construção. Para Papert o desenvolvimento de um material concreto é tão importante quanto o pensamento abstrato [Kafai, 2006]. Nesse processo de construção será feita uma conexão entre o concreto e o abstrato por meio de reflexões, possibilitando o sujeito pôr em prática suas ideias, teorias e hipóteses, que antes eram apenas trabalhadas de forma abstrata. Um conceito antigo no construcionismo está nos erros cometidos no processo de construção, no qual não terá mais um papel de punição, pois através dos erros será preciso compreender os equívocos cometidos, reformular seu processo de reflexão e criação para obter o resultado desejado [Lebow, 1993]. Apenas após todos esses procedimentos de hipóteses, prática e reflexão é que poderá chegar ao conhecimento real sobre um determinado assunto.

O papel do computador no construcionismo tem uma grande importância, por facilitar esse trabalho de construção, uma vez que em determinadas situações se tornaria inviável construir algo na vida real, além de possibilitar situações lúdicas que tornem a aprendizagem prazerosa segundo Papert [Papert e Freire, 2000].

4. Linha de Produto de Software

O conceito de Linha de Produto de Software teve início com a ideia de Linha de Produção de Henry Ford no setor automobilístico. A ideia de Linha de Produção consistia em criar uma sequencia de etapas repetidas e produzida pelas mesmas pessoas as tornando especialistas nessas atividades [Turi et al, 2015]. Todavia, o produto final seguia sempre o mesmo modelo nessa produção em massa. Com o tempo surgiu à necessidade de atender diferentes requisitos para cada cliente, como cores de automóvel e tamanho diferentes. Essa necessidade originou a customização em massa, onde inicialmente é criada uma plataforma em comum, que possui todas as características comuns daquele tipo de produto e apenas posteriormente eram realizadas as modificações exigidas por cada cliente para gerar o produto final finalizado. Dessa ideia surgiu a Linha de Produto de Software, onde se pretende criar uma plataforma em comum com todos os softwares de um determinado domínio, com pontos flexíveis. A plataforma comum possui as características chamadas comunalidades, que estão presentes em quase todos os softwares do domínio, e deve permitir a flexibilidade, chamada de variabilidade, que são os pontos onde o software pode ser customizado para atender a necessidade do cliente [Pohl et al, 2015].

O processo de desenvolvimento de um produto em uma LPS é dividido em duas partes [Blanes e Insfran, 2012]: Engenharia de Domínio – Estabelece a plataforma de reutilização e define as comunalidades e variabilidades; Engenharia de Aplicação – Utiliza-se a arquitetura e processo do domínio para gerar uma aplicação final usando as comunalidades e selecionando as variabilidades.

A reutilização no contexto de uma LPS não está apenas na reutilização de código e sim na reutilização do processo e da arquitetura. A reutilização do código ocorre quando há o reaproveitamento de um código já implementado anteriormente; reutilização do processo ocorre quando atividades, pessoas e papeis são reaproveitados; e a reutilização de arquitetura ocorre quando uma estrutura complexa é utilizada para resolver um determinado problema [Dalmon e Brandão, 2013].

Para possibilitar a implementação e seleção da variabilidade na plataforma comum, pode se adotar diversas técnicas de programação como agregação e delegação, parametrização, herança, *overloading*, compilação condicional, programação orientada a aspectos ou padrões de projetos [Gacek e Anastasopoules, 2001]. O desenvolvedor ainda pode contar com ferramentas para a produção de LPS como o *Colored Integrated Development Environment* (CIDE) [Feigenspan et al, 2010].

5. Metodologia

Inicialmente foi trabalhado o desenvolvimento de um jogo construcionista para então nesta nova etapa, iniciar o desenvolvimento de uma LPS voltada para esses jogos.

No contexto da LPS proposta nesse artigo, foi tido como base o modelo apresentado por Pohl, Böckle e Linden (2005). Esse modelo é um dos mais completos e utilizados em vários cursos. Entretanto, assim como é sugerido, esse modelo foi adaptado para atender as reais necessidades desse trabalho.

Para o trabalho voltado ao desenvolvimento da Engenharia de Domínio, foram selecionados 10 jogos construcionistas analisando tanto as funcionalidades do jogo, quanto os artefatos construídos nesses jogos. A seleção do número de jogos buscou atender diferentes tipos de jogos construcionistas sem repetições e que não comprometessem com o tempo disponível para a produção da LPS. Na implementação do código da plataforma comum, foram gerados dois frameworks, sendo um desenvolvido em PHP sem o uso de ferramentas de apoio e um segundo framework desenvolvido em Java utilizando a ferramenta de suporte à construção de LPS, CIDE. O desenvolvimento dos dois frameworks serviu para verificar se o modelo proposto está apto a realizar diferentes frameworks, avaliar diferentes técnicas de implementação da variabilidade e o processo realizado com e sem o uso de uma ferramenta de suporte a produção de LPS. Para a avaliação da LPS desenvolvida na Engenharia de Domínio, foram desenvolvidos quatro jogos na Engenharia de Aplicação, sendo 3 deles desenvolvidos na versão em PHP e um quarto na versão em Java. A avaliação do processo de desenvolvimento dos jogos foi baseada na verificação da viabilidade de construção de jogos, nas métricas de número de linhas de código, tempo de desenvolvimento, complexidade ciclomática e opinião do desenvolvedor dos jogos. Na avaliação também são realizadas comparações de um jogo desenvolvido com a LPS e sem a LPS desenvolvida, nomeada de JIndie, como também o desenvolvimento através da técnica de compilação condicional com a ferramenta CIDE e outra versão sem apoio de uma ferramenta de construção de LPS.

O desenvolvimento da LPS pode ser visto na seção 6 deste artigo e sua aplicação no desenvolvimento de jogos é apresentada na seção 7, seguida dos resultados na seção 8.

6. Uma Linha de Produto de Software proposta no Contexto de jogos construcionistas

Nesta seção é apresentada a LPS proposta para o desenvolvimento de jogos construcionistas. A LPS foi intitulada de JIndie, originada do termo Jogos Indie, normalmente utilizados para jogos desenvolvidos por pequenas empresas ou pessoas sem apoio financeiro. A abordagem nesta seção será da Engenharia do Domínio, passando pelas etapas de identificação e resolução dos problemas.

Um problema numa LPS é identificado como sendo os requisitos que os softwares necessitam, também conhecidos como *features* ou as comunalidades e variabilidades da LPS. Devido ao fato de não possuir *stakeholders* para levantar os requisitos, a ação realizada na etapa de levantamento dos requisitos foi semelhante a LPS iMA [Dalmon e Brandão, 2013], utilizando como base softwares e jogos já existentes. Nesta primeira etapa foram selecionados 10 jogos construcionistas e listados quais os requisitos presentes em cada um desses jogos tanto no contexto de um jogo construcionista de forma geral, como mais especificamente no artefato construído por esses jogos. Após a listagem dos requisitos, foram criadas duas tabelas de matriz de requisitos, uma para os requisitos do jogo e outra para os requisitos do artefato. A matriz de requisitos tem como finalidade filtrar e identificar quais desses recursos são requisitos exclusivos de um jogo, comunalidades ou variabilidades. Tendo filtrados e classificados os requisitos, estes foram organizados através do modelo *Feature Model,* que permite através de uma representação visual demonstrar as comunalidades, requisitos mandatórios, e variabilidades em forma de requisitos opcionais ou alternativos[Feigenspan et al, 2010]:

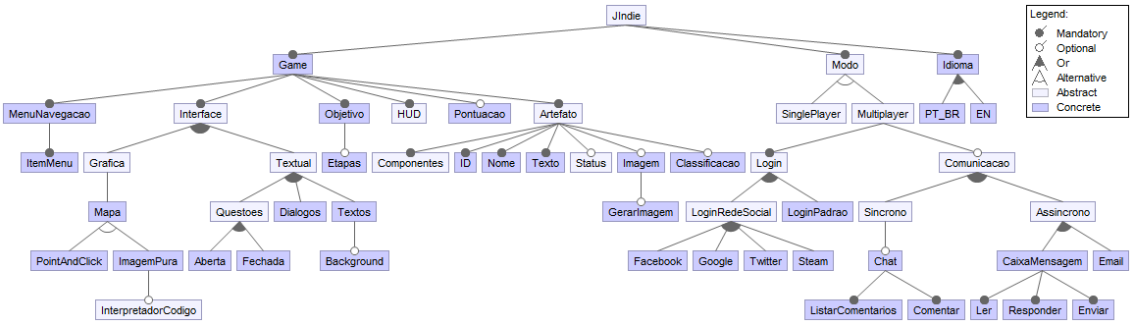

Figura 1. *Feature Model* da LPS JIndie

Entre os principais requisitos encontrados nos jogos construcionistas e representados na Figura 1, podemos citar a opção de criar imagens dinâmicas e interpretação de códigos inseridos pelo jogador como acontece no jogo Logo e RoboCode que serão apresentados na próxima seção, além de outros requisitos como comunicação entre jogadores, estilo *Multiplayer* ou *Single Player,* sistema de pontuação, escolha de idiomas e objetivos. Já no artefato o principal requisito identificado foi à necessidade de ter componentes que possam ser agregados e utilizados de forma fácil, para montar e finalizar o artefato sendo construído.

Para realizar essa integração foi planejado utilizar o Diagrama de Componentes, por facilitar a reusabilidade do artefato e seus componentes. Na Figura 2 podemos ver através do Diagrama de Componentes como quatro jogos distintos podem construir diferentes artefatos através do uso de interfaces. No primeiro bloco a direita (azul), podemos ver um jogo onde o artefato é um projeto de software ou uma empresa e o

componente é um funcionário que está sendo contratado pela empresa. No segundo bloco (vermelho), o artefato sendo construído se trata de um bolo e o componente pode ser um ingrediente deste bolo, como açúcar. No terceiro bloco a direita (verde), temos um jogo de construção de cidade, onde o componente adicionado é um prédio da cidade como hospital. Por fim no último bloco (laranja) temos um jogo de construção de imagens através de comandos, como acontece no jogo Logo. O componente Tile que aparece no diagrama representa blocos do desenho ou imagem sendo agregada junto com a adição do componente do artefato como o desenho de um prédio no mapa da cidade.

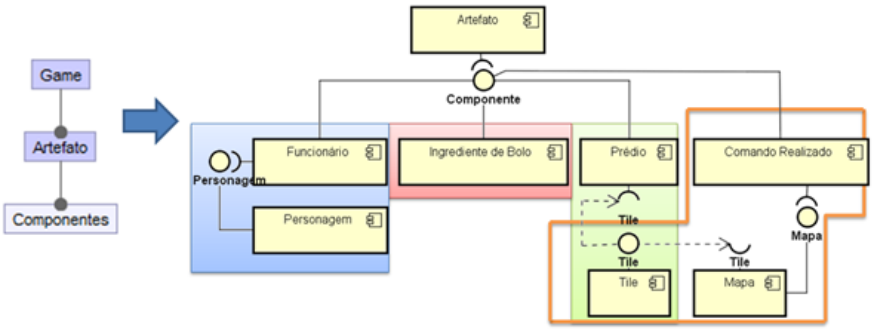

Figura 2. Diagrama de Componente do Artefato e seus componentes

Para a implementação da LPS foi adotada duas versões, na qual a primeira é composta por técnicas de implementação da variabilidade de agregação e delegação, parametrização, herança e técnicas de padrões de projetos com o *factory*. A segunda versão além de contar com essas técnicas citadas anteriormente, também foi desenvolvida com o suporte da ferramenta CIDE utilizando a técnica de implementação da variabilidade por compilação condicional. A escolha de duas implementações serviu para avaliar o impacto do uso de ferramentas de suporte a construção de LPS como o CIDE com a técnica de compilação condicional. A técnica de compilação condicional tem como finalidade definir quais trechos de códigos e arquivos pertencem a quais requisitos através do uso de *tags*, de forma que se um requisito não for necessário, todos os códigos e arquivos pertencentes a este, não estarão presentes na aplicação final.

7. Quatro jogos construídos a partir da LPS proposta

Para validar a viabilidade e qualidade do uso da LPS JIndie, foram desenvolvidos quatro jogos utilizando os dois frameworks da LPS. Os jogos desenvolvidos (Figura 3) foram baseados no Sim Investigador, RoboCode e Linguagem Logo que são jogos construcionistas que já foram utilizados em sala de aula.

Figura 3. Jogos desenvolvidos através da LPS JIndie

O primeiro jogo desenvolvido foi o Sim Investigador. Sim Investigador se trata de um jogo no qual o jogador assume o papel de detetive que deve solucionar casos envolvendo conteúdos de diversas disciplinas, como matemática, português, geografia ou história, através de um sistema de perguntas e respostas. Além de poder jogar um caso, o jogador também pode criar seus próprios casos assim como o professor também pode acompanhar os casos jogados criados pelos seus estudantes [Lessa Filho et al, 2015]. No caso do desenvolvimento do Sim Investigador foi possível ter acesso ao código original assim como a opinião do desenvolvedor comparando o desenvolvimento com e sem a LPS JIndie.

O segundo jogo desenvolvido foi baseado na Linguagem Logo bastante utilizada em colégios no processo de ensino de formas geométricas. Neste jogo construcionista, o jogador assume o controle de uma tartaruga através de comandos como "parafrente" ou "paraesquerda". A cada movimento realizado pela tartaruga na imagem, é deixando uma linha representando o caminho realizado, podendo assim criar figuras geométricas, num processo que busque a criatividade e reflexões de formas de figuras por parte do jogador [Fein et al, 2013]. O jogo desenvolvido com a LPS JIndie, além de contar com as funções de movimento, habilitar e desabilitar as linhas formadas ao mover a tartaruga, também apresenta a funcionalidade do jogador criar suas próprias funções como definir um método para criar um quadrado ou triangulo.

O terceiro e último jogo desenvolvido foi baseado no jogo RoboCode. O jogo RoboCode é um jogo voltado ao aprendizado de programação, permitindo aos jogadores através da linguagem Java ou C# criarem seus próprios robôs, para batalharem entre si [RoboCode, 2016]. Este jogo foi desenvolvido duas vezes com a LPS JIndie, sendo uma delas assim com os jogos citados anteriormente sem a ferramenta de suporte a LPS, o CIDE, e a outra versão utilizando o CIDE e a técnica de compilação condicional. As duas versões desenvolvidas permitiram o jogo *on-line* utilizando o *login* pela rede social Facebook, possibilitando que os jogadores amigos pudessem batalhar entre si. O *login* por redes sociais é um dos recursos já disponibilizados pela LPS JIndie.

8. Resultados e Discussão

Após realizar a produção dos jogos e observar que a LPS JIndie possibilita a construção de jogos construcionistas, foram analisados os jogos construídos através das métricas baseadas em trabalhos correlatos como número de linhas de códigos, opinião do desenvolvedor, necessidade de refatoração no domínio da aplicação, número de arquivos manipulados pelo desenvolvedor e a Complexidade Ciclomática (CC). A CC é uma métrica proposta por McCabe para avaliar a complexidade de softwares baseada nas linhas de códigos, de forma que através do fluxo do código é possível informar se este código apresenta baixo ou alto risco de apresentar falhas durante a sua manutenção [Watson et al, 1996].

No desenvolvimento do primeiro jogo, Sim Investigador, foi possível realizar uma comparação com o desenvolvimento do código original. A implementação do jogo com a LPS não apresentou dificuldades que impedissem o desenvolvimento, pelo contrário, foi possível obter ganho de desempenho devido aos diversos recursos extras que a LPS oferece, possibilitando que o desenvolvedor se focasse no desenvolvimento do núcleo do seu jogo. O jogo original contém 6.937 linhas de código que foram implementadas pelo desenvolvedor, enquanto a versão utilizando a LPS apresenta 20.210 linhas, das quais apenas 1.849 foram implementadas pelo desenvolvedor, sendo o restante

fornecido pela própria LPS, representando uma redução de 73,3% no esforço do desenvolvedor ao usar a LPS. O número maior de linhas no código está relacionado ao fato de que a arquitetura desenvolvida na LPS tende a ser mais genérica e completa para atender diversos tipos de jogos. Outros fatores que influenciam também no tamanho das linhas podem ser vistos em recursos de *logs*, filtros de segurança e *parses* de URL amigáveis, que não estavam presentes na versão original do jogo.

Ao avaliar a velocidade de carregamento de páginas entre a versão original e a versão com LPS, foi possível observar que a versão original é alguns milésimos mais rápida que a LPS, todavia esses milésimos são refletidos no fato da LPS possuir mais filtros e controles de segurança a cada solicitação. Em relação à análise da CC desses códigos foi possível observar um ganho considerável na qualidade dos códigos, de forma que os códigos com a LPS se tornaram mais limpos e menos complexos. Quanto menor a CC, mais fácil é de trabalhar no código e quanto maior, mais complexo esse código é. Recomenda-se que este valor esteja abaixo dos 20 pontos. No código original no cadastro dos usuários é possível ver uma CC com 14 pontos, enquanto na versão com a LPS JIndie esse valor caiu para apenas 1 ponto. No geral o desenvolvimento do Sim Investigador com a LPS se tornou mais organizado e robusto devido a padrões como MVC. Seu processo foi de fácil adaptação, obtendo um ganho de 45% do tempo usado na implementação do jogo. Para essa avaliação, foram desconsiderados o tempo de definição dos requisitos e criação da interface do jogo, uma vez que já haviam sido desenvolvidos e definidos no jogo original.

Na produção do jogo Logo foi possível realizar a conclusão do jogo com o desenvolvimento de apenas 409 linhas por parte do desenvolvedor (10,01% de todo código usado no jogo, sendo o restante disponibilizado pela LPS), uma vez que os recursos mais complexos como gerar imagem e interpretar códigos inseridos pelo jogador, já são disponibilizados pela própria LPS. Todos seus códigos apresentaram CC abaixo de 5 e sua produção foi possível ser realizada em apenas 16h. Todavia, a LPS apresentou uma limitação relacionada a imagem gerada. A classe responsável por gerar as imagens é baseada em um desenho formado por blocos como uma matriz, o que limitava a rotação da tartaruga em apenas 8 direções. Para contornar essa situação possibilitando formas mais arredondadas, é recomendado reduzir o tamanho das imagens dos blocos semelhantes ao que ocorre em desenhos na forma de *pixel art*.

A primeira versão do RoboCode em PHP desenvolvida sem a compilação condicional resultou em um código com 19.641 linhas, na qual apenas 1.194 foram desenvolvidas pelo programador. A segunda versão em Java utilizando o CIDE e a compilação condicional possibilitou um código com apenas 3.803 linhas, todavia 1457 linhas foram implementadas pelo desenvolvedor. Desta forma, é possível observar que o uso da compilação condicional reduz consideravelmente o número de linhas do projeto final, deixando apenas os conteúdos relevantes para aquele código, todavia a compilação condicional é um processo que ocorre antes do desenvolvimento da aplicação final, de forma que ela seja apenas aplicada nos códigos fornecidos pela LPS, não interferindo no código que o desenvolvedor precisa criar, não reduzindo o seu esforço. Porém, o uso da compilação condicional pode reduzir a CC dos códigos fornecidos pela LPS, como ocorre no script responsável por validação de formulários, que apresenta uma CC de 19 pontos sem a compilação condicional e apenas 5 pontos com o uso da compilação condicional.

Ao avaliar o CIDE como uma ferramenta de suporte ao desenvolvimento de LPS, foi possível ter resultados agradáveis devido ao fato que não há a necessidade de se utilizar as tags da compilação condicional. Todo o trabalho de compilação condicional na ferramenta CIDE é realizado selecionando trechos de códigos ou arquivos e informando a qual determinada *feature* esses dados pertencem. Ao pedir para gerar a estrutura que será usada na aplicação final, basta apenas escolher quais *features* o jogo que será desenvolvido vai usar e a ferramenta se encarrega de remover tudo o que não for utilizado. Entretanto, durante esse processo de remoção de trechos de código, a ferramenta CIDE apresentou algumas falhas ao não remover corretamente trechos de códigos como as chaves, sendo necessário remover manualmente na aplicação final. Numa visão de ensino e pesquisa, a ferramenta CIDE se mostrou bastante rica e promissora, porém numa visão mercadológica, apresentou uma alta dependência da LPS desenvolvida na ferramenta, devido à falta de interoperabilidade entre outras ferramentas.

9. Conclusão

O uso de jogos na educação tem se tornando uma ferramenta utilizada para atrair a atenção e motivação aos estudos. Os jogos construcionistas buscam a motivação dos estudantes e um maior aprofundamento nos conteúdos estudados através da construção de artefatos. Entretanto, o processo de desenvolvimento desses jogos pode se tornar complicado devido ao tempo e custo investido, além de poucos recursos voltados a esse gênero de jogos. Neste artigo é apresentada uma LPS com a finalidade de contribuir com o desenvolvimento de novos jogos construcionistas. Com o desenvolvimento da LPS proposta nesse trabalho, foi possível realizar a construção de novas versões de 3 jogos construcionistas usando a LPS. Durante o desenvolvimento dos jogos foi observado melhorias na redução da CC, redução de número de linhas desenvolvidas pelo programador, menor tempo investido, facilidade no desenvolvimento e uma comparação de um jogo usando a técnica de compilação condicional com a ferramenta de suporte CIDE e sem o uso desta técnica. Para trabalhos futuros pretende-se realizar melhorias nos componentes da LPS e criar um ambiente para disponibilizar toda documentação e códigos.

REFERENCIA

Blanes, D., Insfran, E. (2012). “A comparative study on model-driven requirements engineering for software product lines”. Revista de Sistemas e Computação, v. 2, n.1 p.3-13.

Dalmon, D. L., Brandão, L. O.. (2013) “Sobre o Desenvolvimento de Software Educacional: proposta de uma Linha de Produto de Software para Módulos de Aprendizagem Interativa”. In RBIE, v. 21, n. 3, p. 113-130.

Feigenspan, J., Kästner, C., Frisch, M., Dachselt, R., Apel, S. (2010) “Visual Support for Understanding Product Lines”. In Proceedings of the 18th International Conference on Program Comprehension (ICPC), Los Alamitos, CA: IEEE Computer Society, 2010, p. 34--35.

Fein, G. G., Scholnick, E. K. , Campbell, P. F., Schwartz, S. S., Frank, R. (2013) “Computer Space: A conceptual and Developmental Analysis of LOGO”. Constructivism in the Computer Age, Psychology Press.

Gacek, C., Anastasopoules, M. (2001). "Implementing product line variabilities". In ACM SIGSOFT Software Engineering Notes, v. 26, p.109-117.

Kafai, Y. B. (2006) "Constructionism". The Cambridge Handbook of The Learning Sciences, Cambridge University Press, p.35-46.

Käkölä, T., Leitner, A. (2014) "Introduction to Software Product Lines: Engineering, Service, and Management Minitrack". In 47th Hawaii International Conference on System Science. IEEE Computer Society. 2014.

Lebow. (1993) "Constructivist values for instructional systems design: Five principles toward a new mindset". Educational Technology Research and Development, vol. 41, n. 3, pp 4-16. 1993

Lessa Filho, C. A. C., Domínguez, A. H., Costa, F. P. D., Oliveira, P. V. T. A. (2015) "Um jogo digital baseado no construcionismo". In RBIE, v. 23, n.2, p.175-189, 2015.

Marlowe, B. A., Canestari, A. S. (2006) "Educational Psychology in Context Reading for future Teachers". Editora Sage Publications, India.

Morin, Brice et al. (2008) "A generic weaver for supporting product lines". In: Proceedings of the 13th international workshop on Early Aspects. ACM, 2008.

Papert, S. e Freire, P. (2000) "The Future of School", http://www.papert.org/articles/freire/freirePart1.html.

Papert, S e Harel, I. (1991) "Constructionism", publicado por Ablex Publishing Corporation.

Pohl, K., Böckle, G., Linden, F. J. (2005) "Software Product Line Engineering: Foundations, Principles, and Techniques". Springer, 1ª Edição.

Rezende, F. G. C., Nunes, M. M., Brancher, J. D., Sordi Junior, F.. (2013) "Jogo eletrônico e sua influência nas emoções do usuário: Uma análise sobre como os jogos podem estimular emoções relacionadas à aprendizagem". In SBIE 2013.

RoboCode. 2016. Robocode. Disponível em: http://robocode.sourceforge.net/. Acessado em jan 2016.

Santos, W. O., Silva Neto, S. R., Silva Junior, C. G., Bittencourt, I. I. (2015) "Avaliação de Jogos Educativos: Uma Abordagem no Ensino de Matemática". In SBIE 2015.

Tarouco, L. M. R., Konrath, M. L. P., Roland, l. C. (2004) "O professor como Desenvolvedor de seus Próprios Jogos Educacionais: Até Onde Isso é Possível?" In SBIE 2004.

Turi, A., Mocan, M., Ivaşcu, L., Goncalves, G., Maistor, S. (2015). "From Fordism to Lean management: Main shifts in automotive industry evolution within the last century". In MakeLearn International Scientific Conference on Management of Knowledge and Learning, pp. 25-27.

Yoon D. M., Kim K. J. (2015). "Challenges and Opportunities in Game Artificial Intelligence Education Using Angry Birds". In IEEE Access, v. 3, p.793-804.

Watson, a. H., McCabe, t. J., Wallace, D. R. (1996) "Structured testing: A testing methodology using the cyclomatic complexity metric". NIST special Publication, v. 500, n. 235, p. 1-114, 1996.