UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA MULTIDISCIPLINAR DE PÓS-GRADUAÇÃO DO MESTRADO
MODELAGEM COMPUTACIONAL DE CONHECIMENTO

CARLOS ALBERTO CORREIA LESSA FILHO

JINDIE: UMA ABORDAGEM BASEADA NO REUSO DE SOFTWARE E LINHA DE PRODUTO DE SOFTWARE PARA JOGOS CONSTRUCIONISTAS

MACEIÓ-AL
2016

CARLOS ALBERTO CORREIA LESSA FILHO

JINDIE: UMA ABORDAGEM BASEADA NO REUSO DE SOFTWARE E LINHA DE PRODUTO DE SOFTWARE PARA JOGOS CONSTRUCIONISTAS

Dissertação apresentada ao Curso de Mestrado em Modelagem Computacional do Conhecimento da Universidade Federal de Alagoas como requisito parcial para obtenção do grau de Mestre em Modelagem Computacional de Conhecimento.

Orientador: Prof. Dr. Arturo Hernández-Domínguez.

MACEIÓ-AL

2016

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária Responsável: Maria Helena Mendes Lessa

L638j Lessa Filho, Carlos Alberto Correia.
JIndie: uma abordagem baseada no reuso de software e linha de produto de software para jogos construcionistas / Carlos Alberto Correia Lessa Filho. – 2016.
154 f. : il.

Orientador: Arturo Hernández-Domínguez.
Dissertação (Mestrado em Modelagem Computacional de Conhecimento) – Universidade Federal de Alagoas. Instituto de Computação. Programa de Pós-Graduação em Modelagem Computacional de Conhecimento. Maceió, 2016.

Bibliografia: f. 113-119.
Apêndices: f. 120-154.

1. Software - Desenvolvimento. 2. Software – Jogos para computador. 3. Jogos educativos. 4. Construcionismo. 5. Educação – Meios auxiliares. I. Título.

CDU: 004.4:37

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Programa de Pós-Graduação em Modelagem Computacional de Conhecimento
Avenida Lourival Melo Mota, Km 14, Bloco 09, Cidade Universitária
CEP 57.072-900 – Maceió – AL – Brasil
Telefone: (082) 3214-1364

MODELAGEM COMPUTACIONAL DE CONHECIMENTO

Membros da Comissão Julgadora da Dissertação de Mestrado de Carlos Alberto Correia Lessa Filho, intitulada: "JIndie: Uma Abordagem Baseada no Reuso de *Software* e Linha de Produto de *Software* para Jogos Construcionistas", apresentada ao Programa de Pós-Graduação em Modelagem Computacional de Conhecimento da Universidade Federal de Alagoas, em 6 de setembro de 2016, às 9h00min, no auditório do Instituto de Computação da Ufal.

COMISSÃO JULGADORA

Prof. Dr. Arturo Hernández-Domínguez
Ufal – Instituto de Computação
Orientador

Prof. Dr. Fábio Paraguaçu Duarte da Costa
Ufal – Instituto de Computação
Examinador

Prof. Dr. Márcio de Medeiros Ribeiro
Ufal – Instituto de Computação
Examinador

Prof. Dr. Flávio Mota Medeiros
Ifal – Campus Rio Largo
Examinador

Maceió, setembro de 2016.

“Tudo o que um sonho precisa para ser realizado é
alguém que acredite que ele possa ser realizado.”

Roberto Shinyashiki

AGRADECIMENTOS

Primeiramente gostaria de agradecer a minha mãe pelo incentivo e ajuda para que eu pudesse entrar no mestrado.

Ao meu pai, minha irmã e tios pelos apoios durante a jornada.

A minha amada Mylana Dandara Pereira Gama por tornar os dias melhores.

Meus sogros, Girlane e Marizete, pelo carinho e incentivo.

A grande amiga, Laísa Bandeira, por sempre me escutar e por todas as conversas que tivemos.

A amiga de todas as horas, Priscila Pinheiro, por suas longas histórias e amizade deixando os dias mais alegres.

A amiga de quase uma década, Laís Barros pela amizade e por ajudar a superar as dificuldades em comuns.

Ao professor e amigo Mozart de Melo Alves Junior pelo apoio e incentivo para que eu completasse essa etapa.

Ao Professor Dr. Arturo Hernández-Domínguez pela paciência na orientação, direcionamento do conhecimento e por compartilhar os novos conhecimentos que permitiram a conclusão desta dissertação e publicações de trabalhos.

Agradeço a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior por acreditar e financiar a pesquisa, permitindo que esta fosse concluída.

Agradeço aos professores do curso por disponibilizar seus conhecimentos.

Aos companheiros de turma de mestrado e ao Vitor na secretária do curso por todas as explicações e companheirismo.

Aos amigos de trabalho Jonas Lucena, Eduardo Douglas, Marcel Almeida e Daniel Lima pelo apoio e aprendizado.

Aos estudantes da turma de Engenharia de Software da instituição UFAL do ano de 2016 por contribuírem com a minha evolução como professor.

RESUMO

A educação no Brasil apresenta uma deficiência na qualidade de ensino, de forma que muitos professores e empresários sentem que os estudantes, ao se formarem ainda não estão prontos para assumir seu papel na sociedade. Um dos fatores que contribuem para a baixa qualidade do ensino, segundo Papert, pode ser observado na quebra do incentivo a busca de novos conhecimentos, ocorrida quando o estudante entra no colégio. Uma estratégia adotada para motivar o estudante buscar a novos conhecimentos ocorre através do construcionismo, no qual, o estudante para aprender sobre um determinado conteúdo por completo, necessita criar conteúdo concreto sobre o assunto estudado. O desenvolvimento desses conteúdos concretos pode se tornar mais fácil através do uso do computador e com jogos que permitem representar um mundo virtual no qual o jogador tenha todas as ferramentas necessárias para ter a liberdade de criar de sua forma um artefato concreto sobre o assunto estudado. Para facilitar o desenvolvimento desses jogos, que muitas vezes não chegam a ser concluídos devido ao esforço investido, pode-se optar pelo uso de uma Linha de Produto de Software, que se trata de um sistema de produção intensivo de softwares de um determinado domínio. Este trabalho apresenta uma abordagem e produção de uma Linha de Produto de Software, JIndie, que tem como objetivo ser uma Linha de Produto de Software para produção de jogos construcionistas. Para a avaliação da Linha de Produto proposta no trabalho, um estudo de caso foi realizado com a produção de quatro jogos com a finalidade de avaliar a viabilidade de desenvolver jogos construcionistas através do JIndie, como também avaliar o desempenho e as ferramentas utilizadas no processo. Os resultados do estudo de caso revelaram um feito satisfatório na capacidade de desenvolvimento dos jogos, como também uma produção com baixa complexidade e esforço por parte do desenvolvedor dos jogos.

Palavras-chave:. Linha de Produto de Software. Jogos. Construcionismo. Educação.

ABSTRACT

The education in Brazil shows a deficiency in the quality of teaching, so that many teachers and businessmen realize that students that complete the basic education are not ready to assume their role in society yet. One of the factors contributing to the low quality of education, according Papert, can be seen in the break of the incentive to search for new knowledge, which usually occurred when the student starts to go to school. A strategy adopted to motivate the student to search for new knowledge occurs through of constructionism, in which to learn about a particular content, the student needs to create some concrete material. The development of these concrete material may become easier through the use of computers and games that provide a virtual world in which the player has all the tools necessary to create an artifact about the subject studied. To facilitate the development of these games, which often fail to be completed due to the effort invested, the developers can use a Software Product Line, which is a software intensive production system for a particular domain. This paper presents an approach and the JIndie Software Product Line that aims to be a Software Product Line for developing constructionist games. To validate Software Product Line proposed in this work, a case study was carried out with the production of four games to assess the feasibility to implement constructionist games using JIndie, as well as to evaluate the performance and tools used in the process. The case study reveals a satisfactory result in game development capabilities, as well as a production with low complexity and effort by the game developer.

Keywords: Software Product Line. Games. Constructionism. Education.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AGM	Arcade Game Maker
CC	Complexidade Ciclomática
CIDE	Colored Integrated Development Environment
DGE	Distributed Game Enviroment
EaD	Ensino a Distância
FODA	Feature Oriented Domain Analysis
FOSD	Feature-Oriented Software Development
GBL	Game-Based Learning
GQM	Goal Question Metric
HUD	Heads-Up-Display
IGDA	International Game Developers Association
iMA	Módulos de Aprendizagem Interativa
LMS	Learning Management System
LPS	Linha de Produto de Software
LSE	Levantamento da Situação Escolar
MVC	Model-View-Controller
OA	Objetos de Aprendizagem
ORM	Object/Relational Mapping

PBL	Problem-Based-Learning
PHP	Hypertext Preprocessor
POA	Programação Orientada Aspecto
SDK	Software Development Kit
SEI	Software Engineering Institute
UML	Unified Modeling Language
URL	Uniform Resource Locator

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho visa apresentar uma abordagem do uso de Linha de Produto de Software (LPS) para o desenvolvimento de jogos construcionistas, uma vez que esses jogos podem contribuir para melhorar a qualidade de ensino e ao mesmo tempo não dispõem de muitas ferramentas conhecidas para auxiliar a produção destes jogos.

A seguir será apresentada a motivação que levou a produção desta pesquisa.

1.1 Motivação

Os resultados dos últimos anos das principais avaliações do sistema de Levantamento da Situação Escolar (LSE), realizados pelo Ministério da Educação no Brasil desde a década de 90 revelam uma considerável preocupação na qualidade da educação dos estudantes das redes públicas que estão se formando (BORGES, 2014). Esta preocupação tem sido comum por diferentes partes envolvidas, desde os educadores presentes no processo de ensino, até mesmo aos empresários na hora de contratar, de forma que esses envolvidos se sentem bastante incomodados com a qualidade precária do conhecimento dos estudantes ao chegarem às universidades.

De acordo com um estudo realizado (BORGES, 2014), o maior problema na educação não é o acesso à educação básica obrigatória, como normalmente é investido pelo Governo com a ampliação do número de escolas, mas sim na qualidade de ensino.

A abordagem normalmente utilizada no ensino brasileiro, que ocorre através do professor ministrando o conteúdo de forma teórica, enquanto o estudante decora de forma passiva e sem participação, tem se demonstrada 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).

Para Piaget (MONTANGERO; NAVILLE, 2013) o lado cognitivo (conhecimento) deve ser considerado inseparavelmente do lado afetivo, que envolve as emoções primárias, secundárias e o estado de ânimo. Isso ocorre devido ao fato de que é necessário ter interesse e sentir-se motivado ao estudo, impulsionando a

busca de novos conhecimentos através de perguntas, para que este indivíduo se desenvolva por completo.

Por outro lado, o construcionismo de Seymour Papert (PAPERT; HAREL, 1991) defende que para obter o real conhecimento sobre um assunto é necessário realizar um procedimento de construção, como compor uma música ou realizar uma pintura nos estudos sobre arte, de forma que o estudante irá aprender fazendo.

O foco em elevar a motivação aos estudos é um dos principais pontos nas pesquisas voltadas à educação, devido ao fato de que a motivação leva o aprendiz aprimorar seu conhecimento (YOON; KIM, 2015). Consequentemente, muitas pesquisas buscaram elevar a motivação através do Problem-Based-Learning (PBL), outras buscaram elevar a motivação através da diversão. Com a busca da diversão e aprendizado, os jogos eletrônicos começaram a ganhar um papel importante na educação. Entretanto, nem sempre os jogos foram vistos como algo que pudesse somar valor ao processo de aprendizado. Todavia, pode-se observar que os jogos podem ser utilizados para capturar a atenção do jogador, com o intuito de desenvolver habilidades e destrezas, além de obter benefícios como motivação, facilitador de aprendizagem, desenvolvimento de habilidades cognitivas, aprendizado por descoberta, socialização ou coordenação motora (SAVI; DRA, 2008).

Embora os jogos sejam um dos maiores mercados financeiros (SAVI; DRA, 2008), ainda há um grande desafio para quem desejar construir seu próprio jogo de forma independente. Ao se falar sobre jogos independentes, costuma-se abordar no meio acadêmico as vantagens que esses jogos trazem para os diversos setores como entretenimento, educação e pesquisa, porém pouco se fala das dificuldades sobre desenvolvimento de jogos independentes. Ao realizar pesquisas em entrevistas com diferentes desenvolvedores como com Mike Roush (2014) criador do jogo *Binding of Isaac: Rebirth* e Yacht Club Games (2014) empresa responsável pelo jogo *Shovel Knight*, é possível observar diversos problemas como questões financeiras, tempo, bugs e softwares.

A realidade é que a 90% dos jogos construídos de forma independente não chega a ser concluídos (Cho, 2009). Segundo Kenta Cho (2009), a causa se dá principalmente a três fatores: Falta de ideia; Muito tempo gasto no desenvolvimento; O fato do jogo não se demonstrar ser tão interessante ao longo do desenvolvimento pelos desenvolvedores, como parecia no início. Ainda em pesquisa realizada no ano

de 2015 (ROCHA; BITTENCOURT; ISOTANI, 2015) os jogos voltados para abordagens sérias, conhecidos também como Jogos Sérios, apresentam algumas dificuldades durante o seu desenvolvimento, como: a qualidade da metodologia da criação do jogo; reusar e estender os jogos sérios e seus artefatos; e avaliar diferentes pontos do desenvolvimento do jogo.

Para suprir alguns desses problemas como as ideias de requisitos, estratégias, custos e tempo, pode-se optar por adotar estratégias de Linha de Produto de Software. Uma LPS, segundo Käköla e Leitner (2014), é uma metodologia validada industrialmente para o desenvolvimento intensivo de sistemas e serviços de softwares com menor custo, maior velocidade, maior qualidade e satisfação do usuário final. A escolha de uma LPS pode ser mais adequada a esta situação devido ao fato de não visualizar apenas a estrutura do código em si, mas também todo o planejamento e direcionamento da produção.

1.2 Problema da pesquisa

Diante da motivação apresentada, esta pesquisa tem como problemática:

“Como facilitar aos desenvolvedores de jogos construcionistas completarem seus projetos com menor tempo de desenvolvimento, falhas e custos?”

Uma possível hipótese adotada para tratar este problema foi:

“Construir uma Linha de Produto de Software com uma abordagem baseada em jogos com ênfase na aprendizagem baseada nos conceitos construcionistas”

1.3 Objetivo geral

O objetivo geral deste trabalho é analisar, especificar e implementar um modelo de Linha de Produto de Software voltada ao desenvolvimento de jogos construcionistas, que permita a construção de jogos com um menor custo, tempo, direcionamento no desenvolvimento e que possibilite a reusabilidade.

1.4 Objetivos específicos

Como objetivos específicos temos:

- Realizar um estudo sobre os jogos construcionistas e em suas caracteristicas;

- Analisar conceitos sobre a Linha de Produto de Software segundo Pohl, Böckle e Linden, e seu processo de implementação;
- Analisar formas de implementação dos pontos de variabilidade, assim como ferramentas que auxiliem esse processo;
- Implementar uma Linha de Produto de Software para jogos construcionistas;
- Avaliar a Linha de Produto de Software construída com base em três jogos existentes e avaliar as formas de implementação dos pontos de variabilidade com diferentes técnicas e ferramentas.

1.5 Metodologia da Pesquisa

A coleta e estudo das informações deste trabalho relacionados ao construcionismo e a Linha de Produto de Software foi realizado através de artigos, anais, sites e livros.

A construção da Linha de Produto de Software foi realizada através do modelo proposto por Pohl, Böckle e Linden (2005) com adapatações para o mercado de jogos construcionistas.

Para o estudo de caso foram implementados quatro jogos utilizando a LPS desenvolvida visando verificar se atendia a hipótese. Na avaliação foram utilizadas métricas como Complexidade Ciclomática, número de linhas de código, dificuldades e vantagens encontradas e implementação no domínio do jogo.

1.6 Contribuições da Pesquisa

- Esta dissertação gerou como contribuições cientificas:

  - Análise do domínio de jogos construcionistas;
  - Um modelo de uma Linha de Produto de Software para jogos construcionistas;
  - Publicação de quatro artigos sobre o construcionismo e Linha de Produto de Software;

- Esta dissertação também gerou como contribuições de engenharia:

  - Uma Linha de Produto de Software para jogos construcionistas;

- Quatros jogos construcionistas;

1.7 Estrutura da Dissertação

Além desta introdução, este trabalho contém mais 7 seções, sendo estes:

Na seção 2 são apresentados conceitos em relação ao Construcionismo, como também será apresentado sobre o inicio dos jogos eletrónicos até o seu uso na educação, como é o caso dos jogos construcionistas.

Na seção 3 será abordado o conceito e informações relacionadas a construção de uma Linha de Produto de Software, tanto no processo Engenharia do Domínio, que consiste na criação da LPS, como também na Engenharia de Aplicação que consiste na construção de uma aplicação com a LPS.

Na seção 4 será apresentado diferentes métodos de implementação dos pontos de variabilidades e ferramentas que possam auxiliar a implementação de uma LPS.

Na seção 5 serão apresentados trabalhos relacionados a Linha de Produto de Softwares voltada a produção de jogos e aplicações educativas.

Na seção 6 será apresentada a Linha de Produto de Software para Jogos Construcionista, JIndie. Nesse capítulo também será demonstrado a documentação dos pontos de variabilidade.

Na seção 7 será apresentado o Estudo de Caso realizado envolvendo a implementação de três jogos usando o framework da LPS na versão em PHP e um jogo na versão em Java. No final também será realizado uma avaliação sobre a LPS JIndie e as técnicas utilizadas.

Na seção 8, este trabalho será finalizado com a apresentação das considerações finais, contribuições e trabalhos futuros.

2 CONSTRUCIONISMO

Para compreender os conceitos relacionados ao construcionismo de Seymour Papert, é necessário antes entender os conceitos relacionados ao construtivismo desenvolvido por Jean Piaget, pois este serviu como base para o desenvolvimento do construcionismo (KAFAI, 2006).

Segundo Kitcher (1983), antigamente acreditava-se que o conhecimento poderia ser adquirido por meios aprioristas ou empiristas. O conhecimento adquirido por meios aprioristas é representado pelo conhecimento adquirido através das ideias inatas, que emergem do sujeito em processo de amadurecimento. Com esse entendimento, o sujeito é o único responsável por aflorar o conhecimento que este já possui. Por outro lado, o conhecimento adquirido por meios empiristas, ocorre pela experiência vivida através do meio ao qual o sujeito está presente, de forma que todo sujeito ao nascer é considerado um ser sem conhecimento, como uma tabula rasa.

Diante dessas concepções, poderíamos entender que no apriorismo todo o conhecimento adquirido seria exclusivamente mérito do sujeito, pois este conhecimento já viria com ele de outras vidas. Já no contexto do empirismo, o sujeito sendo um ser de tabua rasa, ou seja, uma pessoa sem nenhum conhecimento, não teria mérito algum no aprendizado sendo este exclusivo do meio. Portanto, o aluno apenas aprenderia no empirismo caso o professor o ensinasse, do contrário o aluno jamais seria capaz de aprender.

Seguindo um conceito diferente das ideias inatas do apriorismo e do comportamentalismo do empirismo, Jean Piaget desenvolveu o conceito do construtivismo, que apresenta uma nova forma de analisar a origem do conhecimento, não sendo de responsabilidade exclusiva do sujeito ou do meio como fonte do conhecimento, mas sim através de uma visão biológica do ser, que apresentaria a capacidade de adquirir novos conhecimentos ao atingir certa faixa etária (OVERTON, 2006). Estas faixas etárias ocorrem através da maturação biológica e são organizadas em quatro estágios do desenvolvimento do conhecimento, que podem ser organizados da seguinte forma (MARLOWE; CANESTARI, 2006):

Sensório-motor – Ocorrendo do nascimento até aproximadamente os dois anos, nesta fase a criança começa a ter a noção do mundo através da manipulação.

O conhecimento será adquirido através dos toques, movimentos, segurando e sentindo objetos, por isso que nesta fase é comum crianças levarem objetos à boca.

Pré-operatório – Inicia na transição dos dois anos e continua até aproximadamente os sete anos. É nesta fase onde se inicia o desenvolvimento da linguagem e da escrita, a aceleração dos pensamentos e sentimentos interindividuais.

Operações Concretas – Incide entre os sete e onze anos. É nesta fase onde se inicia o uso do raciocínio lógico e a capacidade de ter mais de um ponto de vista para uma situação. Nesta fase também começa o entendimento de operações concretas, como tempo, espaço e velocidade.

Operações formais – Inicia por volta dos doze anos. No último estágio do desenvolvimento do conhecimento, o sujeito já possui um pensamento lógico num nível maior de equilibração, considerado por Piaget o auge do desenvolvimento cognitivo. O sujeito agora é capaz de realizar pensamentos abstratos, reflexão, ter opiniões próprias e entender metáforas.

Durante esses estágios do desenvolvimento do conhecimento, o sujeito passa por diversas mudanças que podem ser consideradas “adaptação”, o que implica dizer que o sujeito sairá de um momento de desequilíbrio para um momento estável e equilibrado. Esse processo de adaptação é composto por duas etapas: assimilação, onde passa a compreender as novas ideias; acomodação onde aceita e passa a aplicar os novos esquemas aprendidos à realidade (KAFAI, 2006).

Por outro lado, Papert possuía uma visão um pouco diferente de Piaget, que é apresentada no construcionismo, na qual além dos estágios presentes no construtivismo, para gerar um conhecimento real, o sujeito deve construir algo concreto, ao contrário das formas abstratas e intangíveis de ensino.

Dentro do contexto do construcionismo, o sujeito irá desenvolver novos conhecimentos através da construção de um artefato. Após obter um conhecimento abstrato, o processo de construção estabelecerá uma relação entre o concreto e o abstrato por meio de reflexões, onde o sujeito poderá testar suas ideias, teorias e hipóteses. Enquanto realiza os seus testes, o sujeito passará a buscar por novos conhecimentos enquanto analisa os erros cometidos. Erros estes, que não mais terão o papel de punição, uma vez que auxiliarão na aprendizagem, já que será necessário compreender os equívocos cometidos para que se possa reformular seu processo de reflexão e depuração, chegando assim ao resultado desejado (LEBOW,

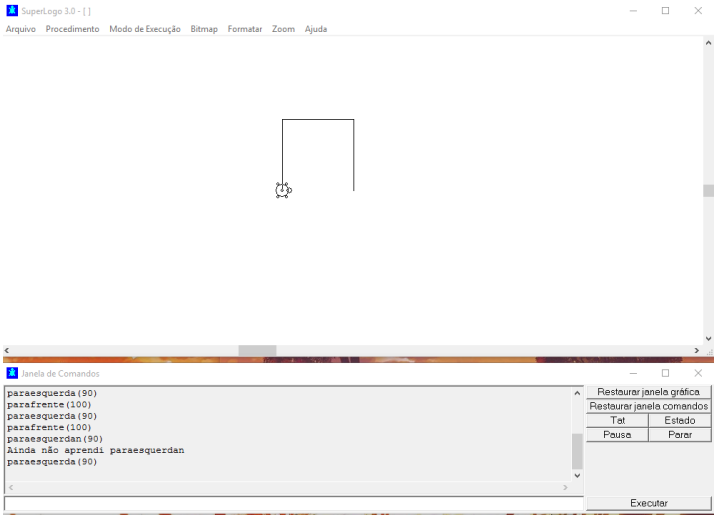
1993). Portanto, o estudante terá uma visão real de como o conteúdo ministrado funciona, conseguindo assim obter um conhecimento completo do conteúdo como Papert defendia.

Para Papert, a escola possui um aspecto negativo na vida do estudante, a partir do momento em que transmite os conteúdos programáticos de forma teórica, sem proporcionar ao estudante uma experiência real. Devido a isso, o estudante não irá adquirir o conhecimento completo sobre o assunto, pois para obtê-lo seria necessário o processo de construção de um material concreto (PAPERT; FEIRE, 2000). Exemplos de construções por parte do estudante poderia ser a elaboração de uma palestra, pintar um quadro, criar uma peça teatral ou desenvolver um programa de computador. Conteúdos que o estudante precise buscar e se aprofundar no conhecimento para que possa representá-lo de alguma forma.

Atualmente, já é possível perceber o uso do construcionismo presente em alguns colégios, como é o caso do uso da robótica em algumas disciplinas envolvendo conteúdos de programação, engenharia e física, onde os estudantes tanto do ensino superior quanto do ensino médio podem criam seus próprios robôs utilizando algumas plataformas abertas como o ARDUINO em competições ou eventos (ARLEGUI et al., 2010).

É neste ponto onde o uso do computador ganha um espaço importante na educação. Os computadores são ferramentas que permitem a construção de objetos com uma maior facilidade. A Linguagem Logo (FEIN et al., 2013), por exemplo, é usada com bastante frequência no Construcionismo, no qual o usuário terá o controle de uma tartaruga que forma linhas à medida que navega pela tela, possibilitando a construção de figuras geométricas (Figura 1). O jogo incentiva ao mesmo tempo a criatividade e a busca de novos conhecimentos, através de reflexões de diferentes meios para formar uma figura geométrica. Neste momento as crianças passavam a dar ordens ao computador e a programar, coisa que antes era apenas destinado a adultos.

Figura 1 - Linguagem Logo



Fonte: Elaborada pelo autor

Porém, o uso do computador nem sempre possui uma característica construcionista. O computador também pode ser utilizado como uma forma de seguir instruções já definidas, tendo papel instrucionista (PAPERT; HAREL, 1991). Os softwares tutoriais são um clássico exemplo disto, pois, os usuários não poderão questionar as ações, apenas executá-las, deixando de lado o papel de construção.

Diante desse entendimento do construcionismo, um software para ser considerado construcionista tem que possibilitar a construção de um artefato concreto de modo que o jogador aprenda durante o seu uso, como possibilitar a liberdade de criação e um *feedback* imediato de suas ações, o que leve a refletir sobre o conteúdo estudado.

2.1 Construcionismo e os jogos

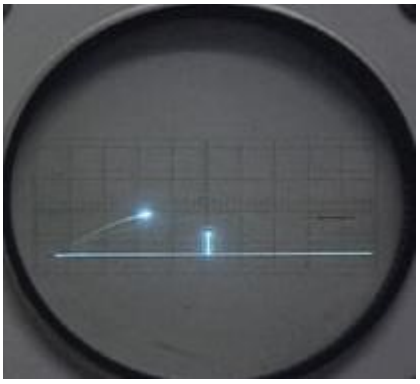
2.1.1 História dos Jogos educativos

Os jogos começaram a sua história por volta da década de 50 (ROGERS, 2010), onde apenas poucas pessoas tinham acesso. Os primeiros programadores eram estudantes de universidades como MIT ou funcionários de instalações militares com supercomputadores.

O primeiro jogo conhecido na história é o *Tennis for Two* (Figura 2) desenvolvido por William Higinbotham no Laboratório de Brookhaven (WOLF, 2008). O jogo possui uma interface gráfica muito simples representando um ponto de luz indo de um canto a outro da tela como um jogo de tênis. Porém, foi no ano de 1962

com o lançamento de *Spacewar* desenvolvido no MIT que os jogos começaram a ter maior visibilidade. Novas ideias de desenvolvimento para jogos foram surgindo, e no ano de 1971 surgiu a primeira máquina de Arcade, dando continuidade no ano seguinte com o lançamento do primeiro vídeo game de casa, Mangnavox Odyssey.

Figura 2 - Tennis for Two



Fonte: Belonio (2010)

Porém os jogos educativos não demoraram tanto para surgir dentro deste universo. Não há ao certo como saber qual foi o primeiro jogo educativo a ser criado, mas acredita-se que um dos primeiros jogos tenha sido o "*Logo Programming*", Linguagem Logo em português (ONLINE UNIVERSITIES, 2012), cuja sua importância no aprendizado baseado no construcionismo já foi citada.

Com a chegada do Apple II nas casas das pessoas, jogos como "*Lemonade Stand"* começaram a ganhar mais espaço. *Lemonade Stand* é um simples jogo de barraca de venda de limões, onde as crianças poderiam aprender mais sobre economia e negócios.

"Where in the World Is Carmen Sandiego?", lançado originalmente em 1983, serviu para trazer os primeiros jogos ás salas de aula, como uma ferramenta complementar ao ensino. Não muito distante, houve o lançamento do jogo *SimCity,* baseado na construção e administração de cidades.

Muitos desses jogos que fizeram parte dos primeiros 30 anos dos jogos eletrônicos serviram de inspiração para abrir espaço para outros jogos educativos, tanto por despertar interesse dos estudantes, quanto também por proporcionar meios diferentes de aprendizagem.

Na Seção 2.1.3, poderá ser observado alguns jogos construcionistas que surgiram graças a alguns desses jogos.

2.1.2 Jogos na Educação

A criação dos jogos educacionais tem sido iniciada há bastante tempo, porém apenas nos anos 2000 é que as pessoas começaram a usar o termo *Game-Based Learning* (GBL) (ARIFFIN et al., 2014). Devido a ser um método novo e por ser algo presente na vida de muitos estudantes, o método de aprendizagem baseado em jogos pôde despertar um interesse aos estudos com maior facilidade (LUO et al., 2010).

A vantagem dos jogos educativos digitais em comparação aos jogos educativos não digitais é a maior praticidade para realizar as demonstrações dos conteúdos de forma didática e lúdica. Segundo Rubel (1999), a união entre computador e jogos educativos possibilita novas formas de realizar e avaliar o aprendizado presencialmente, como também possibilita o Ensino a Distância (EaD), de forma que o professor, mesmo sem a presença física, tenha a condição de ensinar um determinado conteúdo e avaliar a performance dos seus estudantes.

Apesar das vantagens vistas nos jogos educativos, é possível observar que estes jogos educativos são focados em um tema específico, o que dificulta o seu uso em diferentes disciplinas ou conteúdos. Havendo assim a necessidade do professor ou estudante precisar criar seus próprios jogos para atender as suas necessidades.

Para suprir essas necessidades e a escassez de jogos educativos, os professores podem optar por realizar o desenvolvimento de um jogo por meio de kits que já possuem recursos prontos para o desenvolvimento de jogos. Alguns desses kits também chamados de *engines*.

As *engines* no contexto de jogos são plataformas que proporcionam a flexibilidade, extensão e reusabilidade de algoritmos, métodos e técnicas com grande eficiência em diferentes ambientes. As *engines* disponibilizam funções como renderização, física, detecção de colisão, além de eficientes sistemas de gerenciamento de memória (CADAVID et al., 2014).

Devido ao fato que algumas *engines* permitirem o desenvolvimento de jogos sem a necessidade de conhecimentos avançados em programação ou em lógica é comum observar essas ferramentas sendo utilizadas por pessoas de diversas idades com diferentes finalidades. Uma das práticas adotadas ocorre nas salas de aulas, aonde os professores convidam os estudantes a criarem seus próprios jogos relacionados a alguma disciplina, seguindo uma metodologia construtivista,

buscando a aprendizagem através do lúdico (CATETE et al., 2015).

Três engines bastante utilizadas na produção de jogos são:

- **Unity 3D** – Criado pelo Unity Technologies é uma poderosa ferramenta para construção de jogos 3D ou 2D. A ferramenta também conta com uma grande comunidade ativa, serviços extras e uma própria loja virtual de recursos (UNITY TECHNOLOGIES, 2016).
- **GameMaker** – GameMaker Studio é uma ferramenta desenvolvida pela YoYo Games com o intuito de desenvolver um kit para atender tanto profissional da área quanto pessoas sem grandes conhecimentos em programação (YOYO GAMES, 2016).
- **RPG Maker** – RPG Maker é uma ferramenta fácil e intuitiva, com foco em jogos de turno, conhecidos como *Role Playing Games* (RPG MAKER, 2016).

Embora estas engines facilitem a construção de um jogo, para pessoas que não sejam profissionais da área, ainda é necessário um conhecimento básico ou avançado em informática para seu desenvolvimento.

O problema no uso dessas ferramentas é o fato de se limitar apenas a etapa de desenvolvimento, não participando das etapas de levantamento de requisitos e planejamento do projeto, como uma LPS disponibiliza. Porém, tais ferramentas podem ser utilizadas junto de uma LPS na etapa de implementação.

2.1.3 Jogos construcionistas digitais

Os jogos construcionistas (LESSA FILHO et al., 2015) se diferenciam dos jogos tradicionais voltados a educação, por permitir uma maior liberdade ao jogador de tomar as ações que considerar necessárias para a construção de um artefato. O jogador irá passar pelas etapas de elaboração da hipótese, testes, erros e validações ao tomar essas próprias decisões.

Abaixo podemos observar dez jogos construcionistas que foram selecionados devido a já possuírem contribuições acadêmicas no ensino ou por serem jogos comerciais com grande publico e com contribuição ao ensino. Esses jogos serviram

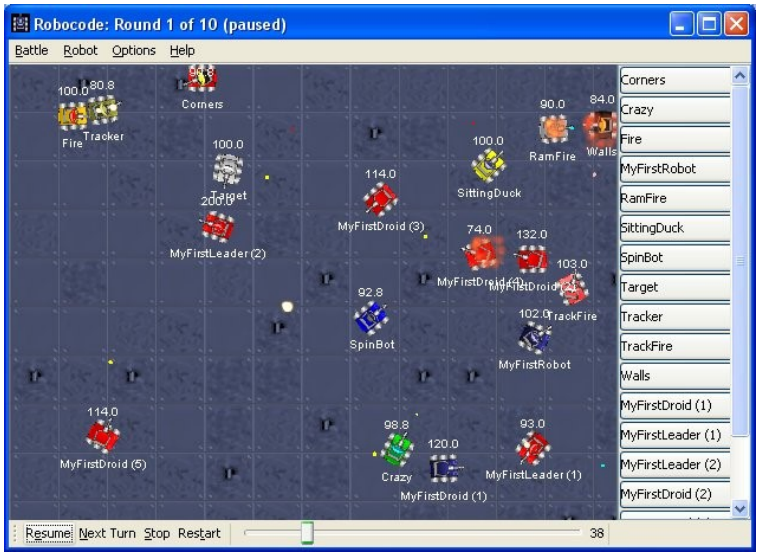
como base para o desenvolvimento da Linha de Produto de Software proposta neste trabalho:

Sim Investigador – Sim Investigador foi um jogo desenvolvido com a proposta construcionista com a intenção de auxiliar tanto os professores na hora de avaliar as dificuldades e desempenhos dos estudantes, quanto também motivar os estudantes a usarem seus conhecimentos adquiridos assumindo o papel de um detetive na resolução de casos. O diferencial do jogo é permitir que os casos do jogo fossem criados pelos próprios estudantes (LESSA FILHO et al., 2015).

Linguagem Logo – Linguagem Logo é um programa voltado à criação de figuras geométricas, onde o jogador terá o controle de uma tartaruga que deixa rastro enquanto navega pelo cenário (PAPERT; HAREL, 1991). O controle da tartaruga é realizado através de comandos como “parafrente” ou “paraesquerda”. Através dessa liberdade o jogador poderá aprender sobre estruturas de códigos, formatos e dimensões de figuras geométricas.

RoboCode – Robocode (Figura 3) é um jogo de programação, onde o objetivo é criar um tanque robô para batalhar contra outros jogadores nas linguagens Java ou .NET (ROBOCODE, 2015). Neste jogo, o jogador pode aprender programação a partir da criação de um robô próprio.

Figura 3 - Competição no Robocode



Fonte: Robocode (2015)

SimCity – SimCity (MAXIS, 2014) é um jogo de simulação de construção de cidades, de modo que o jogador terá como missão construir sua própria cidade, e a cada ação realizada pelo jogador, resultará em uma reação no mundo do jogo. Em

SimCity, o jogador será remetido várias vezes as diversas reflexões de como administrá-la, aprendendo com os erros cometidos.

Civilization – *Civilization* (Figura 4) é uma série de jogos desenvolvida pelo *game designer* Sid Meier (FIRAXIS GAMES, 2016). No jogo, o jogador terá o controle sobre uma nação desde os primórdios até tempo futuros, passando por descobertas de novas tecnologias, administração de exércitos, administração da nação e diplomacia. Além de poder tomar suas próprias decisões, que refletirão na forma como a nação evolui e sobrevive, o jogador também irá aprender sobre diferentes fatos históricos que aconteceram ao longo da história da humanidade.

Figura 4 - Tela do jogo Civilization II

Fonte: Firaxis Games, 2016

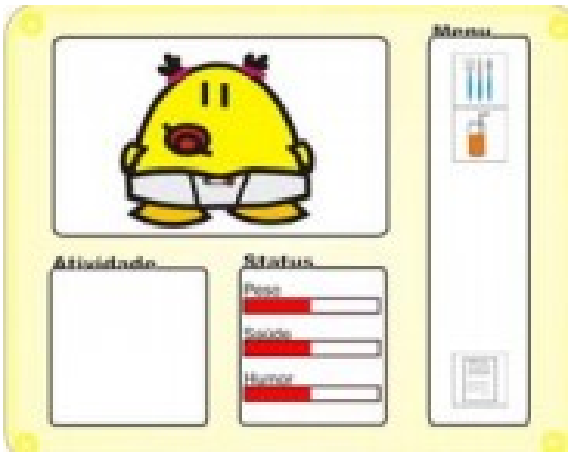
Business Tycoon Online – Em *Business Tycoon Online*, o jogador assumirá o papel de um empresário que acaba de criar sua própria empresa no mundo de *Liberty City* (DOVOGAME, 2010). No jogo, o jogador terá a liberdade de escolher qual será o seu tipo de empresa, quais funcionários contratar, demitir, promover funcionários, comprar ou fabricar recursos, expandir suas empresas em diferentes regiões da cidade, fazer parcerias com outras empresas, entre outras atividades comuns na administração de empresas.

Software Development Manager – Em *Software Development Manager* o jogador irá gerenciar uma equipe de empregados para o desenvolvimento de softwares conforme os contratos realizados com os clientes (KOHWALTER et al., 2014). A mecânica do jogo permite ao jogador decidir estratégias a serem tomadas no desenvolvimento do software, como também definir tarefas para cada membro da equipe. Durante o jogo, o estudante poderá aprender sobre o fluxo do

desenvolvimento de jogos e como certas escolhas podem impactar nos resultados finais.

Gene2 – Gene2 (Figura 5) é um jogo no qual o jogador irá assumir o papel de um cientista genético que cria geneticamente como será o seu “animal” de estimação, semelhante aos bichinhos virtuais (MORELATO et al., 2008). No jogo, os estudantes poderão aprender sobre genética, ao combinar diferentes genes e observarem como a combinação irá resultando na construção do seu bichinho virtual.

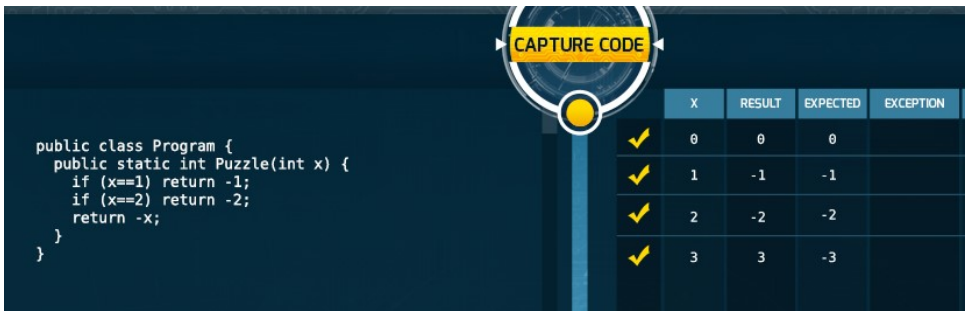
Figura 5 - Tela de Gene2



Fonte: Morelato et al., 2008

Code Hunter – Code Hunter (Figura 6) é um jogo educativo desenvolvido pelo setor de pesquisa da Microsoft, que tem o intuito de servir de auxilio para estudantes e professores de disciplinas de programação e Engenharia de Software (MICROSOFT RESEARCH, 2016). O jogo permite ao usuário a criar seus próprios códigos em Java ou C# de forma que o resultado final obtido no método criado atinja os critérios informados. O jogo não limita ao jogador criar um código especifico, dando total liberdade ao usuário implementar a sua própria resolução do problema.

Figura 6 - Verificando o código no Code Hunter



Fonte: Elaborada pelo autor

Geogebra - GeoGebra é um aplicativo de matemática dinâmica, que permite que o jogador possa interagir por meio de funções de forma que as representações possam ser ilustradas em figuras geométricas e algébricas (BARROS; STIVAM, 2012). Assim, o jogador ao adicionar uma função para realizar a construção de uma forma geométrica através de pontos, vetores, segmentos e retas, irá aprofundar seu conhecimento sobre o comportamento das funções e suas representações visuais.

Na análise desses jogos foi possível observar que os jogos construcionistas apresentam as seguintes características que os tornam construcionistas:

- Um problema ou objetivo a ser realizado;
- Liberdade de criar sua própria solução;
- Um artefato a ser moldado por várias ações;
- *Feedback* imediato das ações realizadas;
- Aprender um conteúdo através das ações e *feedbacks* proporcionados pelo jogo;

As informações sobre o construcionismo apresentadas nessa seção serviram como base para o desenvolvimento do domínio da Linha de Produto de Software proposta neste trabalho, uma vez que demonstram o que proporciona o aprendizado no construcionismo, características que representem um software construcionista, assim como exemplos de jogos construcionistas.

3 LINHA DE PRODUTO DE SOFTWARE

A construção de vários softwares para uma empresa pode ser um processo muito demorado e caro, entretanto muitas empresas trabalham com o desenvolvimento de softwares na mesma linha de domínio, de modo que se houver a possibilidade de reutilizar recursos e etapas no desenvolvimento, esse processo pode se tornar mais rápido e barato.

Henry Ford fez o mesmo no setor automobilístico com a Linha de Produção. Ford lançou por volta do século XIX o sistema de produção em massa que consistia em simplificar a montagem dos veículos reduzindo o esforço humano, aumentando a produtividade sem elevar os custos na produção e proporcionando que todos os veículos criados mantivessem o mesmo padrão (WOOD JUNIOR, 1992).

Entretanto, a produção em massa não permitia a customização dos veículos e nem todas as pessoas possuíam as mesmas necessidades, de forma que algumas pessoas precisavam de carros maiores, outras desejavam carros em outras cores. Diante dessa situação, surgiram as plataformas comuns (POHL et al., 2005). As plataformas comuns são plataformas que já possuem todas as características comuns daquele objeto que pretende ser criado, porém permitindo adicionar com facilidade outras características quem podem variar de forma a criar o produto final diferenciado para cada cliente.

As Linhas de Produtos de Softwares (LPS) seguem o mesmo conceito da produção em massa customizada, construindo uma plataforma comum para todos os softwares de um determinado domínio.

Northrop (2002) define uma LPS como um sistema de produção intensiva de software que compartilham características comuns e gerenciáveis para satisfazer um segmento particular do mercado. Os recursos do núcleo formam a base da LPS. Esses recursos incluem, mas não se limitam, a arquitetura, os componentes reusáveis, o modelo do domínio, documentação e especificação, planos de testes e descrição do processo.

O processo para a construção da Plataforma Comum na Linha de Produto de Software inicia-se durante a etapa da análise dos requisitos que os softwares do mesmo domínio possuem. Essa análise pode ser feita de diversas formas como analisar os produtos já existentes. Após analisar as características em comuns desses softwares, deve ser implementada na plataforma comum, todas as

características comuns, ou também como são chamadas: as comunalidades. Essa plataforma também deve ser baseada em componentes de forma a permitir a flexibilidade para durante a produção do produto final possam ser adicionadas novas customizações.

Enquanto as caraterísticas comuns e presentes em todos os produtos de uma LPS são conhecidas como comunalidades, as flexibilidades são chamadas de variabilidades, ou seja, os pontos pré-definidos que podem receber customizações no desenvolvimento do produto final. As variabilidades do software podem ser representadas nos artefatos por uma estrutura de decisão e estes pontos podem ser descritos como ponto de variação e variante. Os pontos de variação são os locais nos artefatos onde uma estrutura de decisão pode ser realizada e as variantes são as alternativas desses pontos (LOBO et al., 2007). As variações podem ocorrer na documentação, na arquitetura, código-fonte ou durante a execução do código (SILVA et al., 2011).

A representação das comunalidades e variabilidades podem ser expressas através de *feature model*, que é apresentada originalmente através do modelo *Feature Oriented Domain Analysis* (F.O.D.A.) desenvolvido pelo *Software Engineering Institute* e se tornando o método mais popular na década de 90 (KANG et al., 2002). As características presentes em um domínio sejam comunalidades ou variabilidades também podem ser representadas na literatura como *features*.

O *feature model* é representado por características do tipo:

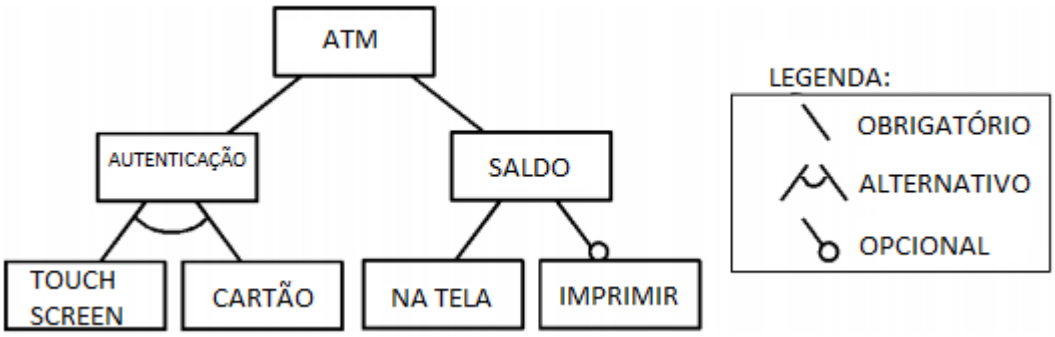
Obrigatória/Mandatórias – As características presentes em todos os produtos da LPS, por tanto são as comunalidades da LPS;

Opcional – As características que podem ou não estar presente nos produtos da LPS;

Alternativa – Conjunto de características quem podem ser escolhidas uma (XOR) ou mais (OR) para preencher um determinado componente.

Na Figura 7 é possível observar um simples exemplo de um sistema ATM que possui como característica alternativa a identificação do usuário por um sistema *Touch Screen* ou por leitor de cartão. Também é possível ver uma característica opcional em um ATM, a impressão do saldo o cliente:

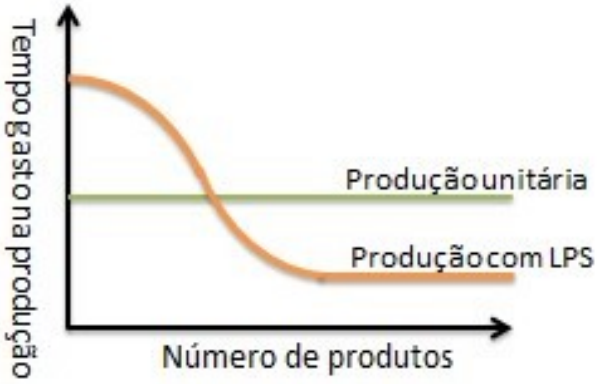
Figura 7 - Feature Model de uma LPS ATM



Fonte: Elaborada pelo autor.

Durante a construção inicial da LPS os custos e tempo investidos serão maiores comparados a construção de um único software, uma vez que será preciso criar uma estrutura flexível. Porém após aproximadamente o terceiro produto já será possível observar um ganho comparado a produção unitária, ou seja, sem o uso de uma Linha de Produto (MCGREGOR et al., 2002).

Figura 8 - Representação do tempo de produção



Fonte: Elaborada pelo autor.

Entre as vantagens do uso de uma LPS é possível observar (POHL et al., 2005): redução dos custos no desenvolvimento; aumento de qualidade do produto final; redução do tempo de produção.

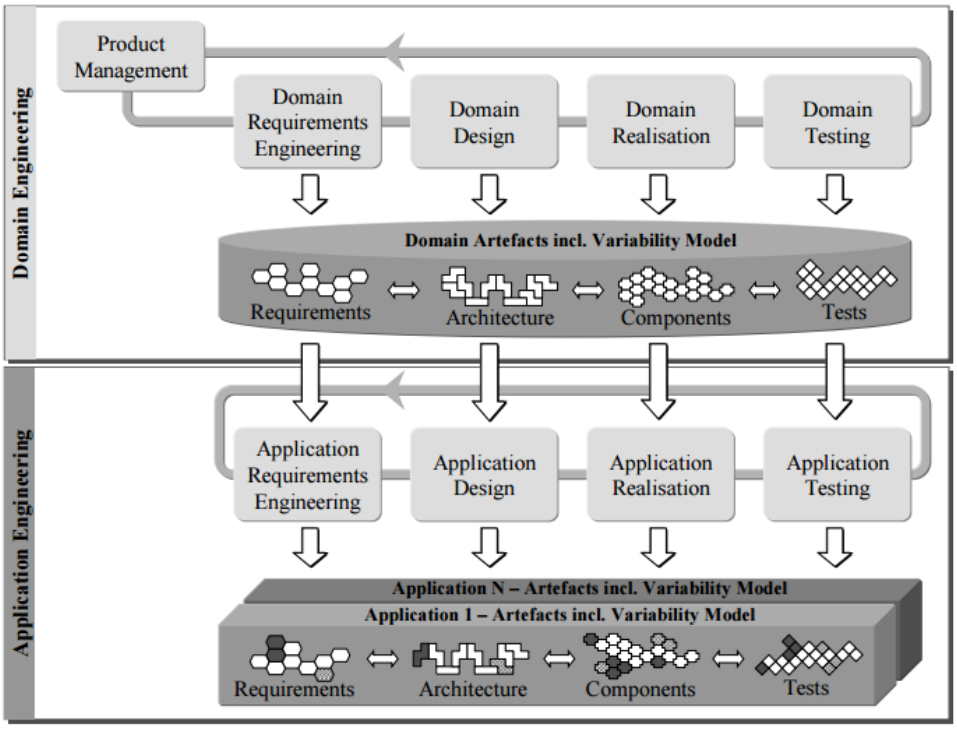
A facilidade do desenvolvimento de software através da LPS se dá, devido ao fato do processo ser dividido em duas etapas:

Domain Engineering **(Engenharia de Domínio)** – processo que estabelece as comunalidades e variabilidades do domínio. É nessa etapa onde é construída a plataforma comum. Esse processo é dividido em quatro subprocessos: *Domain Requirement Engineering (*Engenharia de Requisitos do Domínio*); Domain Design*

*(*Projeto do Domínio*); Domain Realisation (*Implementação do Domínio*); Domain Test (*Teste do Domínio*).*

***Application Enginnering* (Engenharia de Aplicação)** – Na engenharia da aplicação é possível encontrar os processos da construção do produto final utilizando os artefatos e a plataforma construídos no processo de engenharia do domínio. Esse processo é dividido em quatro subprocessos: *Application Requirements Engineering (*Engenharia de Requisitos da Aplicação*); Application Design (*Projeto da Aplicação*); Application Realisation (*Implementação da Aplicação*); Application Testing (*Testando a aplicação*).*

Figura 9 - Estrutura da LPS



Fonte: Pohl et al., 2005.

Não existe um modelo fixo para a construção de uma LPS, podendo este ser adaptado para atender as necessidades de uma empresa ou cliente. Este trabalho irá trabalhar em cima do modelo proposto por Pohl, Böckle e Linden.

Nas Seções 3.1 e 3.2, são apresentadas de forma mais detalhada as etapas presentes na LPS.

3.1 Engenharia do Domínio

O processo de *Domain Engineering* ou Engenharia do Domínio tem como finalidade identificar, construir, catalogar e disseminar conjuntos de componentes que possuam aplicabilidade em softwares existentes e futuros de um determinado domínio. Assim, engenheiros de software podem compartilhar componentes para reutilizá-los no desenvolvimento de sistemas futuros (PRESSMAN, 2011).

Para Pohl et al. (2005), a Engenharia do Domínio é formada para atender basicamente três objetivos:

- Definir as comunalidades e variabilidades da Linha de Produto de Software;
- Definir e construir artefatos reusáveis que atendam as variabilidades;
- Definir qual tipo de softwares (o domínio), do conjunto de softwares que será construído através da Linha de Produto.

Essa etapa de identificar, construir e validar a engenharia de domínio é composta por quatro atividades não sequenciais, ou seja, a qualquer momento da etapa da construção da Engenharia do Domínio, o engenheiro de software, poderá voltar a outras etapas para realizar melhorias constantes.

As etapas do processo de Engenharia de Domínio serão vistas nas subseções 3.1.1 a 3.1.4.

Durante o desenvolvendo destas etapas, também é recomendado a documentação das variabilidades nos diferentes artefatos. Para cada etapa, existe uma forma mais adequada para realizar a documentação.

3.1.1 Engenharia de Requisitos do Domínio

Para Nuseibeh e Easterbrook (2000), engenharia de requisitos é o processo de descobrir o objetivo e as necessidades dos *stakeholders*, realizar análises, documentar as informações para posteriormente realizar implementações. Este processo também é o responsável por realizar a separação dos problemas e as soluções.

Na Linha de Produto de Software, o *Domain Requirement Engineering* ou Engenharia de Requisitos do Domínio tem o papel de determinar as comunalidades

e variabilidades, assim como realizar a documentação. Embora seja a primeira etapa a ser realizada, ela será trabalhada ao longo de todo o desenvolvimento da linha de produto.

Durante o processo de busca dos requisitos, são elencados os problemas (as necessidades que o *software* precisa atender), que terão relação constante com Projeto do Domínio, que buscará a solução dos problemas identificados nesta etapa.

Após identificados os requisitos que serão atendidos, pode ser criada uma Matriz de Requisitos entre os requisitos identificados e as aplicações existentes, caso existam. Para cada combinação deve verificar se o requisito está presente ou não em cada aplicação conforme o exemplo da Figura 10:

Figura 10 – Modelo de uma Matriz de Requisitos

Requisitos/Aplicação	Aplicação 1	Aplicação 2	Aplicação 3	Aplicação 4
Requisito 1	✓	✓	✓	✓
Requisito 2	x	x	✓	x
Requisito 3	✓	✓	✓	x
Requisito 4	✓	✓	x	x

Fonte: Elaborada pelo autor

Diante dos valores obtidos com a Matriz de Requisitos é possível listar quais informações são características comuns a todos os softwares ou a maioria desde que não entre em conflito com outros requisitos (comunalidade); quais são flexíveis (variabilidade); e quais são exclusivas de um software.

Tendo identificado as comunalidades e variabilidades, o engenheiro de software pode representá-las através de uma *Feature Model*, como foi visto anteriormente.

Outra tarefa importante a ser realizada é documentar os pontos de variabilidade presentes na etapa de Engenharia de Requisitos do Domínio, que pode ser tanto de forma textual ou na linguagem de modelagem, *Unified Modeling Language* (UML) com diagramas. UML é uma linguagem visual para modelagem de sistemas de software orientados a objetos (BOOCH et al., 2005).

Na representação visual por UML o engenheiro de software pode optar por diagramas como Diagrama de Caso de Uso e de Sequência, que é mais fácil para os clientes que vão utilizar o sistema entenderem o funcionamento, como também pode optar por diagramas de estado, para facilitar o entendimento do comportamento de certos componentes pelo programador.

3.1.2 Projeto do Domínio

Pressman (2010) define uma arquitetura sendo um conjunto de componentes de uma construção de forma a coexistir como um todo. Já na visão de arquitetura em software ou sistemas se trata da estrutura que abrangem os componentes de softwares, propriedades externas visíveis e a relação entre eles.

Seguindo essa concepção, o Projeto do Domínio será trabalhado com os requisitos identificados no processo de Engenharia de Requisitos do Domínio, de forma a criar uma estrutura que solucione os problemas e que possam ser implementadas na etapa seguinte.

Durante a construção da arquitetura é interessante trabalhar com o uso de Diagrama de Componentes, pois este permite que peças possam ser retiradas e encaixadas com facilidade sem grandes impactos no sistema. Ao trabalhar com interfaces, permitirá que o sistema esteja apto a receber um componente futuro que ainda não tenha sido vislumbrado até o exato momento.

3.1.3 Implementação do Domínio

A etapa de Implementação do Domínio tem como finalidade implementar os artefatos reusáveis em formas de um framework com componentes e interfaces, que represente a arquitetura planejada.

Nesta etapa também é possível observar o nível de abstração das interfaces, de forma que uma abstração alta permite trabalhar diversos tipos de componentes, o que também pode ocasionar ao uso de componentes não desejados. Em contrapartida, uma abstração baixa, pode ter uma flexibilidade muito limitada.

A documentação da variabilidade dos artefatos na etapa de implementação ocorre diretamente com a demonstração do Diagrama de Classe, principalmente com o uso de interfaces para mostrar uma visão abstrata de como algo pode ser implementado de diferentes formas.

3.1.4 Teste do Domínio

No teste de domínio são realizados os testes que buscam tanto erros nas funcionalidades, algo básico de qualquer desenvolvimento de software, como

também das comunalidades e variabilidades, para verificar se estas estão atendendo de fato os requisitos dos clientes.

Quando mais cedo um erro for identificado menor será seu custo. Uma relação desses custos poderia ser representada de forma que um erro identificado na definição do projeto, tendo um custo médio de 1 dólar, ao ser identificado na produção do código teria um custo de 10 dólares e na etapa de teste seu custo subiria para 50 dólares (DUSTIN, 2002).

Por outro lado, um teste completo pode se tornar algo inviável numa LPS devido a quantidade de opções existentes. Desta forma o engenheiro pode optar por testes menos completos, como as seguintes estratégias (POHL et al., 2005):

- ***Sample Application Strategy –*** Essa estratégia consiste em criar algumas aplicações completas para verificar se a plataforma está atendendo algumas flexibilidades, porém devido ao fato de não testar todas as possibilidades, ainda há o risco de existir componentes e interfaces que não se adequem a diferentes softwares.
- ***Commonality and Reuse Strategy*** – Essa estratégia é semelhante ao *Sample Application*, porém o teste é focado em testar todas comunalidades e componentes reusáveis. Desta forma, ao invés de criar diversas aplicações para cada combinação, utiliza-se as mesmas aplicações já implementadas, modificando apenas as partes envolvendo os pontos de variação, para verificar se o ponto de variabilidade atende ou não a flexibilidade desejada.

A documentação dos pontos de variação no Teste do Domínio pode ser realizada em duas etapas: Planejamento e execução do Teste, por fim é avaliado através do relatório.

Ao completar as quatro etapas da Engenharia do Domínio (requisitos, projeto, implementação e teste), a plataforma comum que será utilizada na Engenharia de Aplicação deverá estar pronta para o uso.

3.2 Engenharia de Aplicação

No processo de Engenharia de Aplicação ocorre o desenvolvimento de um novo produto da LPS, utilizando os componentes reusáveis e explorando as

variabilidades que a Linha de Produto disponibiliza. Este processo é dividido em quatro etapas que se relacionam diretamente com as etapas da Engenharia do Domínio.

3.2.1 Engenharia de Requisitos da Aplicação

O objetivo da Engenharia de Requisitos da Aplicação consiste em identificar e documentar os requisitos de uma aplicação em particular, reutilizando os artefatos desenvolvidos na Engenharia do Domínio.

Uma atividade essencial durante esta etapa é a comunicação com os *stakeholders* em relação aos artefatos construídos na Engenharia de Requisitos do Domínio, uma vez que os gerentes de projetos e clientes estão relacionados diretamente com os requisitos da aplicação. Essa comunicação é essencial para identificar se os pontos de variabilidades atendem as necessidades dos *stakeholders* ou não. Se não atender, novas características exclusivas da aplicação devem ser adicionadas aos requisitos da aplicação que está sendo desenvolvida.

Os *stakeholders* além de poderem criar novos pontos de variação, também terão o poder de modificar os existentes, como remover variações tidas como obrigatórias e alternativas.

3.2.2 Projeto da Aplicação

Assim como a Engenharia de Requisitos da Aplicação tem relação direta com a Engenharia de Requisitos do Domínio, o Projeto da Aplicação também estará relacionado com o Projeto do Domínio.

Nesta etapa, é definida a arquitetura final da aplicação, aproveitando a arquitetura já desenvolvida na Engenharia de Domínio, sendo apenas necessário selecionar as variações escolhidas nos pontos de variação e adicionar novos componentes caso um novo requisito exclusivo da aplicação tenha sido adicionado.

A vantagem desse processo, é que nesta etapa será preciso apenas focar nas partes especificadas da aplicação, uma vez que ela já terá a arquitetura para as demais partes.

3.2.3 Implementação da Aplicação

Durante essa etapa é utilizado o framework criado durante a Implementação do Domínio para que se possa criar a aplicação final, que posteriormente será testada pelos envolvidos para saber se obteve a satisfação desejada.

Nesta etapa são utilizadas as interfaces e componentes reusáveis do framework, realizando as configurações necessárias.

As interfaces costumam ser reusadas sem a necessidade de nenhuma alteração, todavia nos componentes envolvendo os pontos de variações é comum haver a necessidade de ajustá-lo a aplicação final.

3.2.4 Testando a Aplicação

Nesta quarta etapa é verificada a qualidade da aplicação em si. Se esta alcançou os requisitos e atendeu as expectativas dos clientes. Durante essa etapa são testadas todas as outras fases do desenvolvimento da aplicação e seus testes normalmente são baseados nos testes do domínio.

Devido aos testes já realizados na Engenharia do Domínio, na Engenharia da Aplicação os testes são voltados aos pontos de variação. Ao longo do teste é necessário verificar se não existe uma variação que não deveria estar presente na aplicação, sendo necessário omití-la caso ocorra. Porém é indispensável ter o cuidado de verificar se a remoção de uma variante não causará impacto em uma comunalidade.

Com essas quatro etapas é possível realizar a construção do produto final da Linha de Produto de Software.

Na próxima seção será apresentado como é possível implementar a variabilidade do código na Engenharia do Domínio, para que este possa ser facilmente selecionado e trabalhado na Engenharia de Aplicação. Também será abordado ferramentas com foco no processo de implementação da variabilidade.

4 IMPLEMENTANDO A VARIABILIDADE

Como foi apresentado no capítulo anterior, os pontos de variação representam uma parte importante para permitir a flexibilidade dos softwares construídos através da Linha de Produto de Software.

Nesta seção serão apresentadas técnicas de implementação da variabilidade e ferramentas que auxiliem esse processo.

4.1 Métodos de implementação da variabilidade

Os métodos ou técnicas voltada para a implementação da variabilidade estão diretamente voltadas à forma como um código é programado para ser utilizado por outros desenvolvedores na produção do produto final.

Existem várias formas de realizar a implementação da variabilidade, de forma que nesta seção iremos apresentar sete técnicas.

4.1.1 Compilação condicional

A compilação condicional é amplamente utilizada nos projetos baseados em C e se trata de um mecanismo comum na implementação de LPS. A compilação condicional é um mecanismo de pre-processadores que utiliza anotações nos códigos como tags #ifdef [nome da variante] e #endif, para determinar que uma parte do código esteja relacionada a uma *feature* (característica) em especifico. Desta forma, nas Linhas de Produtos de Software ao criar um produto final e selecionar as *feature*s desejadas através do uso da compilação condicional, o código compilado retornará apenas os trechos dos códigos relacionados as *features* selecionadas (KÄSTNER et al., 2011). De tal modo é possível representar cada variante do código como uma *feature* isolada pelas anotações da compilação condicional, que apenas irá ser adicionada ao produto final, caso a variante tenha sido selecionada.

4.1.2 Overloading

Overloading se trata de uma técnica baseada em reutilizar nomes de métodos já existentes para executar tarefas diferentes. Portanto é possível existir dois métodos diferentes com o mesmo nome, porém com parâmetros, ações e retornos

diferentes. Um exemplo de overloading, poderia ser um método chamado “Soma” que recebe dois parâmetros para realizar a soma dos dois números, enquanto outro método com o mesmo nome “Soma”, recebe três parâmetros realizando a soma dos três valores. O uso overloading promove o reuso de códigos de forma que na LPS um ponto de variação poderia ser representado pelo o uso de métodos com técnica de overloading, enquanto a escolha da variação ocorreria através dos parâmetros recebidos pelos métodos. Todavia, os programadores que utilizam desse tipo de técnica, estão mais propensos a erros, uma vez que podem não saber quando usar a forma correta da técnica (ANASTASOPOULES; GACEK, 2001).

4.1.3 Delegação

No uso de técnica de Delegação, o programador retira do código todas as variabilidades das classes que implementam as comunalidades. Para cada variabilidade será criada uma nova classe que irá implementar essas funcionalidades. Desta forma, quando um código precisar executar uma variação seja ela opcional ou alternativa, a classe que recebeu a função de implementar a variação será chamada (MATOS JÚNIOR, 2008).

Como um exemplo, uma determinada classe com funcionalidades obrigatórias, poderá ter um determinado método que deverá executar uma variabilidade alternativa. Este método irá delegar a definição do que será executado a outra classe, que ficará responsável por escolher a variabilidade desejada. Posteriormente a classe principal irá executar o método já com a funcionalidade definida por outra classe.

4.1.4 Programação Orientada a Aspecto

A Programação Orientada a Aspecto (POA) é uma técnica efetiva para dar suporte a variabilidade e estabilizar a arquitetura de uma Linha de Produto de Software, sendo bastante utilizada no AspectJ, uma extensão da linguagem Java voltada a POA (THÜM et al., 2014). A POA é uma metodologia que permite separar funcionalidades secundárias dos módulos principais do código, as transformando em aspectos. Na LPS a POA tem por objetivo trabalhar com o encapsulamento de *feature*s transversais em novas unidades modulares, sendo os aspectos. A intenção é fazer com que a variação de *features* transversais se torne mais modulares e

evoluíeis quando comparado com mecanismos de variabilidade convencionais, onde o código principal e os pontos de variação ficam unidos (FIGUEIREDO et al., 2008).

4.1.5 Programação Parametrizada

A Programação Parametrizada possui a ideia básica de maximizar o reuso no programa através de armazenamento dos códigos na forma mais genérica possível (GOGUEN, 1984). É comum ser observado técnicas de parametrização em classes de acesso a banco de dados através de técnica como *Object/Relational Mapping* (ORM), que consiste em representar uma classe como uma tabela no banco. Sendo assim, para não ser preciso criar uma classe de acesso ao banco para cada classe representada no banco de dados, é criada uma única classe genérica utilizando técnica de parametrização. A parametrização pode melhorar a usabilidade em uma linha de produtos e também facilitar a rastreabilidade para projetar decisões. No entanto, centralizar código acessado por parametrização que são usados muitas vezes é uma tarefa muito complexa, se não impossível (ANASTASOPOULES; GACEK, 2001).

4.1.6 Herança

A herança é um mecanismo de linguagens orientadas a objetos. Na herança uma classe mãe contem as características comuns entre diversas classes semelhantes, de forma que as classes filhas fiquem responsáveis apenas pelos comportamentos especializados de um grupo. Diante disso, na herança é possível utilizar uma classe mãe para implementar as comunalidades, enquanto as classes filhas ficariam responsáveis pelas variações da LPS. Sendo assim, no produto final o desenvolvedor pode optar por usar apenas a classe filha que atenda a variação desejada no produto final (MATOS JÚNIOR, 2008).

4.1.7 Padrões de Projeto

Os Padrões de Projetos podem ser explorados em um contexto linha de produtos já que muitos deles identificam os aspectos do sistema que podem variar e fornecer soluções para o gerenciamento da variação (ANASTASOPOULES; GACEK, 2001). Nos padrões de projetos podemos ver técnicas como *Factory, Builder, Adapter, Composite, Observer, Strategy* entre outros.

4.1.8 Dependency Injection

A técnica *Dependency Injection* ou Injeção de Dependência é uma técnica que visa diminuir o acoplamento de componentes entre classes, de modo que uma classe não seja mais responsável por buscar ou criar os seus objetos dependentes. Segundo Ekstrand e Ludwig (2016) o uso de Dependency Injection permite que componentes estejam livres de conhecer a implementação de suas dependências; possam reconfigurar mudanças em suas dependências sem modificar os componentes; realizar testes com facilidade.

O processo de injeção de dependências pode ocorrer através do construtor, getters and setters e implementação de interfaces. Deste modo é possível permitir ao desenvolvedor da aplicação final decidir qual será o tipo da propriedade que será usada na dependência da classe.

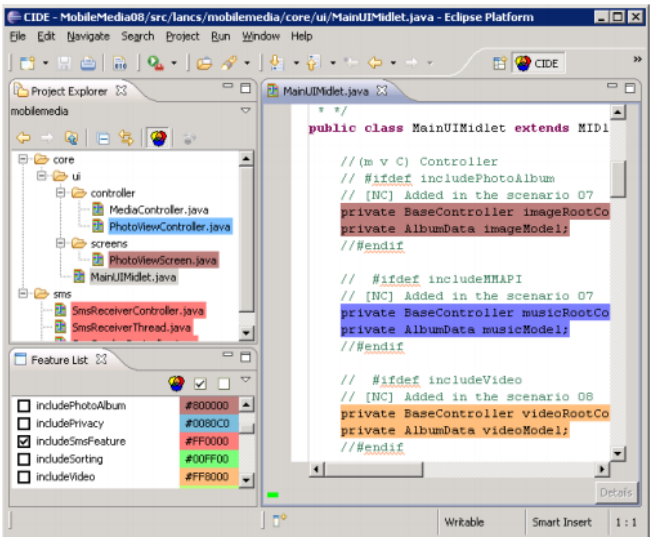
4.2 Ferramentas para implementação da variabilidade

Para trabalhar com técnicas de implementação como as citadas na subseção anterior, o programador pode optar por usar algumas ferramentas que facilitem o processo. Nesta seção serão apresentadas três ferramentas voltadas a produção de Linha de Produtos de Software, que facilitam a implementação da variabilidade através de técnicas citadas anteriormente.

4.2.1 CIDE

Colored Integrated Development Environment (CIDE) é um plugin que dá suporte ao desenvolvimento de LPS baseados na técnica de Compilação Condicional. Ao contrário da Compilação Condicional tradicional baseada em anotações, o CIDE apresenta um editor especializado (Figura 11) que representa as anotações através de cores (FEIGENSPAN et al., 2010).

Figura 11 - Editor visual do CIDE



Fonte: Feigenspan et al, 2010.

Além de disponibilizar um editor visual que torna o processo de desenvolvimento mais fácil e prático, o CIDE também permite realizar as condições de compilação para arquivos e diretórios. No programa é possível realizar a criação de uma *Feature Model*, que servirá como base para definir quais partes do projeto estão relacionadas a quais variações da LPS.

4.2.2 FeatureIDE

Outra ferramenta utilizada é a FeatureIDE. FeatureIDE é um framework *open-source* baseado no Eclipse que dá suporte ao *Feature-Oriented Software Development* (FOSD). O foco principal do FeatureIDE é de cobrir todo o processo de desenvolvimento e incorporar ferramentas que auxiliem a implementação de LPSs em um ambiente de desenvolvimento integrado. A arquitetura da FeatureIDE facilita o desenvolvimento de ferramentas de suporte para FOSD através de linguagens existentes e até mesmo para novas linguagens, reduzindo assim o esforço para tentar adicionar novas linguagens e conceitos. Atualmente a ferramenta é mais voltada à pesquisa e ensino de disciplinas de Engenharia de Software. (THÜM et al., 2014).

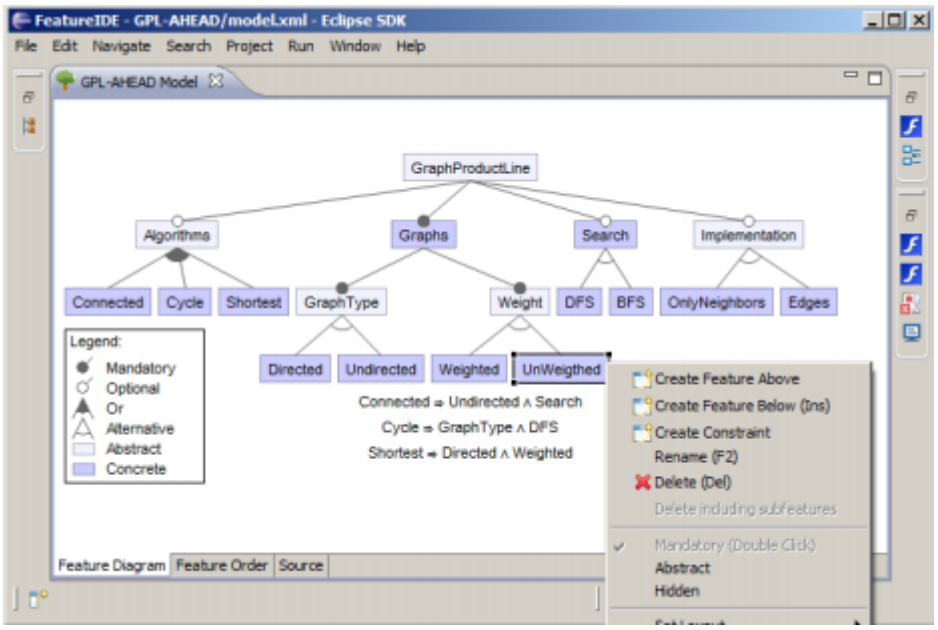
FeatureIDE já conta com a implementação de diversas técnicas voltadas aos processos envolvendo as etapas das LPS. Entre elas é possível citar:

- Programação Orientada a Aspecto

- Compilação Condicional
- Programação Orientada a *Features*
- Programação Orientada a Delta

A ferramenta assim como o CIDE também conta com editor visual do *Feature Model* (Figura 12).

Figura 12 - Editor visual do Feature Model na FeatureIDEE



Fonte: Thüm et al., 2014

4.2.3 Colligens

Uma terceira ferramenta que pode ser utilizada para facilitar a implementação da variabilidade numa Linha de Produto de Software é a Colligens.

Colligens é uma ferramenta baseada no ambiente Eclipse para o desenvolvimento de LPS na linguagem C integrado com a funcionalidade de pre-processadores. Deste modo, os desenvolvedores podem criar modelos de features, analisar estatisticamente a LPS e gerar produtos automaticamente (Medeiros et al, 2016).

Esta ferramenta se assemelha ao CIDE, porém dando ênfase a verificação de ausências de produtos inválidos, visando a redução de falhas relacionadas aos erros de sintaxes relacionados a anotações incompletas.

Outras ferramentas como Pure::Variant, Gear, S.P.L.O.T. ou Holmes podem ser vistas no estudo *Software Product Line: Survey of Tools* de Munir e Shahid (2010).

Na próxima seção serão apresentandos trabalhos que desenvolveram Linhas de Produtos de Softwares.

5 TRABALHOS RELACIONADOS

A LPS JIndie é uma LPS voltada a produção de jogos construcionistas, todavia além da LPS proposta nesse trabalho, também existe outras LPS para o desenvolvimento de jogos ou materiais educativos, entretanto não direcionados ao construcionismo, que é o diferencial buscado pela LPS JIndie.

Nessa seção serão apresentadas algumas Linhas de Produto de Software, selecionadas por fazerem parte do desenvolvimento de jogos ou softwares educativos.

5.1 Uma Linha de Produto de Softwares para Módulos de Aprendizagem Interativa

A LPS desenvolvida por Dalmon et al (2012) tem como intenção solucionar os problemas encontrados durante o desenvolvimento de softwares educativos conhecidos como Módulos de Aprendizagem Interativa (iMA), que se trata de uma família de sistemas para atividades interativas que podem ser integrados a Sistemas de Gerenciamento de Cursos como o Moodle.

iMA são softwares educacionais que tratam de recursos interativos e as principais características presentes são: execução em navegadores *web*, apresentar ferramentas de autoria, promover atribuições interatividade e proporcionar a comunicação com os Sistemas de Gestão de Aprendizagem (DALMON et al., 2012).

O desenvolvimento da LPS foi baseado no estudo de cinco softwares da família iMA anteriormente desenvolvidos pelo grupo de pesquisa da Universidade de São Paulo: iGraf, um sistema para estudo de funções matemáticas e gráficos; iCG, um sistema de emulação de computador com um compilador interno; iComb, um sistema de ensino de combinatória; iGeom, um sistema de Geometria Interativa; e iVProg, um sistema de programação visual.

O processo de avaliação da LPS contou com a refatoração dos softwares utilizando a LPS e entrevistas com os desenvolvedores. Durante o estudo realizado, foi possível observar uma reutilização do código, diminuindo em cerca de 35% do código do IGeom e 45% do IComb. Também foi possível observar pelos programadores que a arquitetura fornecida levou a redução de tempo gasto pensando em como o projeto deve ser organizado. O processo de desenvolvimento

com a LPS também permitiu uma melhoria na qualidade na fase inicial do desenvolvimento e tornar mais fácil a manutenção.

5.2 Uma Linha de Produto de Softwares para jogos mobile

A LPS desenvolvida por Nascimento et al. (2008) possui o foco no desenvolvimento de jogos para mobile. O desenvolvimento de jogos para *mobile* apresentam complexos desafios relacionados a variedade de aparelhos, fabricantes, restrições de plataformas e diferentes configurações de hardwares. Devido a esses fatores a escolha de uma LPS pode ser uma opção conveniente (NASCIMENTO et al., 2008).

Devido a essas limitações encontradas no domínio de jogos eletrônicos para mobile, os pesquisadores propuseram uma LPS que desse suporte ao máximo de detalhes possíveis em nível de código. A LPS desenvolvida buscou realizar a análise através de um estudo de caso envolvendo o desenvolvimento de três jogos de aventura já presentes no mercado e o desenvolvimento de um quarto jogo explorando as características comuns entre eles.

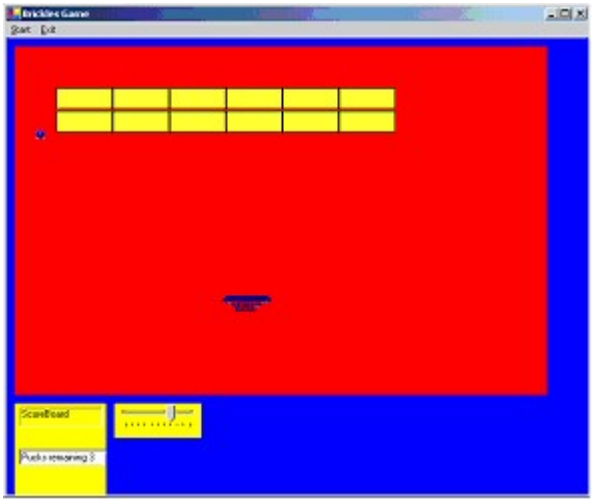
Durante a implementação do jogo foram utilizadas técnicas de compilação condicional para a seleção dos requisitos de cada jogo. Na avaliação foram definidas métricas mínimas envolvendo a complexidade do código, restrições envolvendo o domínio e rastreabilidade dos artefatos que deveriam ser atendidas pelos quatro jogos desenvolvidos.

5.3 Uma Linha de Produto de Software para Jogos no ensino de Linha de Produtos de Software

A LPS Arcade Game Maker (AGM) foi criada pelo Software Engineering Institute (SEI) com o intuito de criar um exemplo de Linha de Produto para dar suporte ao processo de aprendizagem e experiência no uso de Linha de Produtos de Software (SOFTWARE ENGINEERING INSTITUTE, 2009). A Linha de Produto conta com a presença de três simples jogos de plataforma. Ao contrário da maioria dos jogos, os produtos do Arcade Game Maker não possuem foco na interface gráfica, pois o real objetivo desta LPS é ser um exemplo abrangente, de forma a ser um exemplo simples, porém completo.

Seu material passou por um processo de evolução de pelo menos dois anos. Os jogos disponibilizados junto com a LPS para o processo de ensino são: *Brickles*, *Pong* e *Bowling*. *Brickles* (Figura 13) é um jogo no qual o jogador deve controlar uma nave para mover uma bola pelo cenário que deve destruir todos os tijolos, sem cair no chão. *Pong* é o segundo jogo baseando no jogo *Pong* dos arcades. Em *Pong* o jogador terá o controle de duas naves nas extremidades horizontais e terá o papel de impedir que a bola acerte tais extremidades. No jogo *Bowling* o jogador participará de uma partida de boliche e deverá acertar o máximo de pinos que conseguir com a bola.

Figura 13 - Interface do jogo *Brickles*



Fonte: Software Engineering Institute, 2009.

O material disponibilizado é focado em duas partes distintas: A primeira relacionada aos recursos e produtos da LPS como arquitetura, requerimentos, planos de testes e os códigos dos jogos. A segunda parte é voltada para o setor pedagógico com sugestões de exercícios usando os conteúdos da Linha de Produto.

Os jogos produzidos com esta Linha de Produto contam com sistemas voltados a jogos single player; oferecer um sistema gráfico; possuir animações; criação de objetos em movimentação e estáticos. Os jogos criados podem ser multiplayer, porém apenas um jogador pode executar a ação por vez localmente. Quanto às variabilidades da LPS estão presentes: as regras de cada jogo; os tipos e números de peças envolvidas; os comportamentos das peças; e o ambiente físico aonde o jogo irá operar (SOFTWARE ENGINEERING INSTITUTE, 2003).

Para o desenvolvimento de um novo jogo utilizando AGM é analisado previamente uma tabela matriz envolvendo as características de recursos gráficos,

interação, objetos do jogo e pontuação. Através dessa análise é possível observar se o jogo pretendido pertence ao grupo de jogos comerciais para PC, comerciais online ou jogos arcades *freeware,* sendo este último grupo apto a ser desenvolvido através da LPS.

5.4 Uma Linha de Produto de Software para aplicações *e-Learning*

Atualmente a internet tem se mostrado uma ferramenta que traz muita vantagem para o aprendizado. Desta forma, muito tem se gasto com treinamentos virtuais, aprendizagem virtual ou aprendizagem eletrônica do inglês *e-Learning* (GUENDOUZ; BENNOURAR, 2013). Muitas dessas aplicações e-Learning têm sido propostas, sendo as mais conhecidas os Sistemas de Gestão de Aprendizagem, tradução de *Learning Management System* (LMS). Entretanto, um estudo realizado com 113 instituições na Europa revelou que as maiorias das LMS são desenvolvidas por empresas locais ou programadores da própria instituição ao invés de usar LMS existentes. Revelando que as instituições tendem a criar seu próprio LMS para atender seus requisitos específicos.

A partir desse estudo foi desenvolvida uma LPS para desenvolver aplicações e-Learning facilitando a produção de novas LMS, uma vez que essas compartilham elementos em comuns e certas variações. O domínio da LPS envolve os ambientes voltados para escolas, universidades e empresas que gerem treinamento online para seus funcionários. Para o desenvolvimento do *Feature Model*, de forma a identificar os requisitos, houve uma divisão de interesses, sendo uma voltada aos recursos relacionados à lógica de um e-Learning como exercícios, avaliação, gerenciamento de grupos e estatísticas, e o outro relacionado aos recursos que são utilizados na implementação do código como padrões de Objetos de Aprendizagem (OA), tipo de banco de dados e interface.

Para a documentação da variabilidade presente na LPS, foi utilizado o modelo com representação visual (apresentado na seção 3.1), onde um trecho da variabilidade é representado através dos artefatos utilizados lado a lado ao Diagrama de Componente.

5.5 Uma Linha de Produto de Software para Educação e Pesquisa

A LPS foi desenvolvida para trabalhar com ambientes de jogos conhecidos como Distributed Game Enviroment (DGE), com a função de desenvolver ambientes de jogos para dois jogadores baseado em turno com fins educativos e de pesquisa (QUAN, 2013).

Para a coleta dos requisitos foi trabalhado com *stakeholders* para investigar o domínio da aplicação, serviços que a LPS deveria providenciar, restrições e requisitos de qualidade dos ambientes. O desenvolvimento da LPS foi dividido em ciclos, de forma que ao final de cada ciclo era verificado com os *stakeholders* os resultados, problemas e feedbacks para melhorar o ambiente.

Durante os ciclos foram identificados os seguintes requisitos núcleos do domínio:

- Capacidade de jogar remotamente entre os jogadores através da LAN;
- Capacidade de enviar mensagens de textos remotamente;
- Capacidade do manter uma lista de jogadores disponíveis;
- Capacidade de gerenciar os jogadores disponíveis;
- Capacidade de controlar o status do ambiente do jogador;
- Capacidade de acompanhar as atividades existentes no ambiente;

Além dos requisitos núcleos, outros requisitos relacionados a qualidade como robustez, capacidade de atualização e segurança foram definidos junto aos *stakeholders*.

A LPS DGE ainda conta com componentes reusáveis com responsabilidade de fornecer serviços de infraestrutura como persistência de dados, logs, comunicação, segurança e gerenciamento de sessão.

Ao final de sua produção, era possível construir jogos como: *Five in a Row*; Jogos da Velha; e jogos de cartas.

5.6 Análise dos Trabalhos Relacionados de LPS

Após análise dos trabalhos relacionaos de LPS (Seção 5.1 até 5.6), serão apresentados a principal contribuição e o produto gerado de cada trabalho correlato.

Tabela 1 – Contribuições dos trabalhos relacionados

LPS	Principal contribuição	Produtos
iMA (DALMON et al., 2012)	Facilitar a produção de Módulos de Aprendizagem Interativa com integração a ambientes de aprendizagem	Módulos de Aprendizagem Interativa
Para jogos mobile (NASCIMENTO et al., 2008)	Facilitar a produção de jogos atendendo as diferentes características de celulares como resolução da tela	Jogos para dispositivos moveis
AGM (SOFTWARE ENGINEERING INSTITUTE, 2009)	Criar um exemplo de Linha de Produto de jogos arcades para dar suporte ao processo de aprendizagem e experiência em LPS	Jogos de gênero árcade
LMS (GUENDOUZ; BENNOURAR, 2013)	Facilitar a produção de Sistemas de Gestão de Aprendizagem para aprendizagem e treino virtual	Sistemas de Gestão de Aprendizagem
DGE (QUAN, 2013)	Facilitar a produção de jogos de tabuleiros baseado em turnos, multiplayer para dois jogadores, voltado ao ensino	Jogos de turno para dois jogadores

Fonte: Elaborada autor.

A LPS deste trabalho diferencia-se dos trabalhos relacionados a LPS ao disponibilizar recursos usados por jogos do domínio construcionista.

5.7 Análise dos Trabalhos Relacionados de Jogos Construcionistas

Além da análise dos trabalhos relacionados sobre LPS na Seção anterior, será apresentado nessa Seção a presença do problema, do artefato e do feedback dos jogos apresentados na Seção 2.1.3, relacionando-os com as características que definem um jogo construcionista.

Tabela 2 - Jogos construcionistas e suas caracteristicas

Jogo	Problema	Artefato	*Feedback*
Sim Investigador (LESSA FILHO et al., 2015)	Representar uma história envolvendo conteúdos de disciplinas do ensino fundamental ou médio	Narrações/Casos/Histórias criadas pelo jogador	*Feedback* do avaliador que irá liberar ou não a história para os demais jogadores
Linguagem Logo (PAPERT; HAREL, 1991)	Criar figuras geométricas	Desenhos e formas geométricas	Desenho formado pelo comando inserido
RoboCode (ROBOCODE, 2015)	Criar um robô para batalhas	Robôs	Comportamento do Rôbo baseado nos comandos inseridos pelo usuário
SimCity (MAXIS, 2014)	Administrar uma cidade do inicio.	Cidades	Satisfação da população e saldo financeiro
Civilization (FIRAXIS GAMES, 2016)	Administrar uma civilização ao longo das gerações	Nações	Impacto dos relacionamentos com outras nações
Business Tycoon Online (DOVOGAME, 2010)	Administrar uma empresa desde sua origem	Empresa	Situação financeira da empresa
Software Developer Manager (KOHWALTER et al., 2014)	Produzir um software para um cliente	Projeto de Software	Satisfação do cliente
Gene2 (MORELATO et al., 2008)	Criar e cuidar de um bichinho virtual	Bichinho Virtual	Caracteristicas visuais e de saúde do bichinho virtual criado
Code Hunter (MICROSOFT RESEARCH, 2016)	Criar um código que atenda um pedido	Códigos	Alerta de falha na execução do código criado
Geogebra (BARROS; STIVAM, 2012)	Criar figuras baseados em fórmulas de matemática	Gráficos Geométricos	Representação visual do gráfico

Fonte: Elaborada pelo autor.

Com isto, é possível observar que existem jogos construcionistas sendo criados, entretando sem o uso de uma LPS que ofereça esse suporte.

Nas próximas seções será trabalhado o desenvolvimento da LPS proposta nesse trabalho, utilizando os conceitos apresentados nas seções expostas até o momento.

6 JINDIE

Nesta seção será apresentada a Linha de Produto de Software JIndie, desenvolvida para ser uma ferramenta de suporte para desenvolvedores independentes no desenvolvimento de jogos construcionistas.

O nome JIndie se origina do termo “Jogos *Indie”.* O termo “*Indie”* usado nos jogos não se refere ao gênero do jogo como RPG, *Shooter,* plataforma ou Puzzles. O termo “*Indie”* ou Jogos Independentes são mais comumente utilizados para a forma como o jogo se originou, mais especificamente as condições em que o jogo foi desenvolvido e distribuído (SIMON, 2012). Jogos *Indie* costumam ser desenvolvidos por pequenos grupos de pessoas ou até mesmo por uma única pessoa sem a condição de grandes investimentos e na maioria dos casos sem apoio financeiro de patrocinadores.

As *engines* apresentadas na Seção 2.1.2 como o Unity, são ferramentas muito poderosas utilizadas tanto por pequenas empresas quanto por grandes empresas no ramo de desenvolvimento de jogos, porém além de não darem suporte às etapas de planejamento, testes e levantamento de requisitos, as *engines* também não se tratam de ferramentas especializadas no domínio de educação e jogos construcionistas. O JIndie além de apresentar as facilidades que uma LPS possui como o reuso de componentes do código, artefatos e uma plataforma comum, ainda conta com a especialização no domínio de jogos construcionistas, que ainda não conta com uma ferramenta com destaque e especializada no suporte do desenvolvimento desses jogos.

6.1 Uma Linha de Produto de Software

Nas subseções abaixo serão apresentadas as etapas do desenvolvimento da LPS dividida nas quatro etapas da Linha de Produto de Software segundo o modelo proposto por Pohl, Böckle e Linden (2005).

O modelo de usar uma LPS para jogos construcionistas foi escolhido por possibilitar a facilidade no desenvolvimento e direcionar o desenvolvedor da aplicação, quais etapas devem ser realizadas.

6.1.1 Engenharia de Requisitos do Domínio

O primeiro processo realizado durante o desenvolvimento da LPS é realizar a análise do domínio estudado, identificando quais as características são requisitos essenciais para o desenvolvimento de jogos nesse gênero.

As características que pretendem se identificar nessa etapa são classificadas como obrigatórias, opcionais ou alternativas. Para identificar essas características foram selecionados dez jogos do nicho construcionistas, que são apresentadas na seção 2.1.3 deste trabalho. Torna-se difícil definir o tamanho real de quantos jogos construcionistas existem, principalmente pelo fato de todo dia surgirem diversos novos jogos e muitos desses não chegarem ao conhecimento da maioria dos jogadores. Deste modo, a seleção do número de jogos buscou atender diferentes tipos de jogos construcionistas sem repetições, como ocorre com os diversos jogos semelhantes ao *Sim City*, e que não comprometessem com o tempo disponível para a produção da LPS. Os jogos selecionados foram escolhidos através de suas contribuições em artigos ou devido a sua grande utilização comercialmente e ainda proporcionarem um processo de aprendizagem pelo construcionismo.

Tendo selecionado um grupo de jogos, foi realizada a análise de quais características estão presentes em cada jogo. A análise foi realizada em dois momentos. No primeiro momento, pretendia-se buscar as características presentes no jogo de forma geral. No segundo momento foi realizada a análise das características dos artefatos construídos em cada um desses jogos. Tratando-se de jogos construcionistas, em cada um desses jogos, o jogador tem a liberdade de construir um objeto, que o identificamos como o artefato construído através do jogo. Abaixo segue uma lista com os jogos e seus artefatos:

- **Sim Investigador**
  - Artefato: Narrações/Casos/Histórias criadas pelo jogador
- **Linguagem Logo**
  - Artefato: Desenhos e formas geometricas
- **RoboCode**
  - Artefato: Robôs
- **SimCity**
  - Artefato: Cidades

- **Civilization**
  - Artefato: Nações
- **Business Tycoon Online**
  - Artefato: Empresa
- **Software Development Manager**
  - Artefato: Projeto de Software
- **Gene2**
  - Artefato: Bichinho Virtual
- **Code Hunter**
  - Artefato: Códigos
- **Geogebra**
  - Artefato: Gráficos Geométricos

Diante da análise desses jogos foram identificados os seguintes requisitos em relação ao jogo:

1. Jogo *Multiplayer*;
2. Jogo *Single Player*;
3. Inserção de código no jogo;
4. Realizar *login*;
   a. Realizar *login* por rede social;
5. Realizar comunicação entre jogadores;
6. Inserção de formulários;
7. Classificação por *rank*/posição;
8. Uso de Pontuação;
9. Uso de Questionários;
10. Jogabilidade baseada em texto;
11. Jogabilidade baseada em gráficos;
12. Multi-idioma;
13. Uso de personagens;
14. Menu de navegação;
15. Apresentar objetivo;
16. Uso de mapa;
    a. Point and Click;

17. Uso de HUD (*Heads-Up-Display*);
18. Formação de equipe/time;
    a. Com diferentes funções/Cargos;
19. Uso de número de tentativas (Contagem de Erros);
20. Contagem de atributo. (Exemplo: Dinheiro/População/Estamina/Moral...);
21. Uso de nível/experiência.

Para que pudessem ser identificadas quais dessas características são obrigatórias, opcionais ou alternativas foi realizada uma matriz de requisitos que pode ser acompanhada no Apêndice A.

Com a análise da matriz de requisitos, foi possível listar os requisitos ignorados. Entre eles, estão: a inserção de formulário (Requisito 6) por ser importante apenas em seu jogo; uso de sistema de nível (Requisito 21) e Contagem de atributos devido ao baixo uso; Classificação por rank (Requisito 7) e Uso de número de tentativas (Requisito 19), por serem características secundárias em seus jogos e não possuírem usabilidade nos demais jogos.

Entre as características comuns, podemos listar as que tiveram mais presentes nos jogos e que não entraram em conflito com outras características: Uso de Personagem (Requisito 13); Menu de Navegação (Requisito 14); Objetivo (Requisito 15); e HUD (Requisito 17). Também pode ser considerada uma comunalidade de acordo com o modelo seguido nesse trabalho, as características que são importantes em alguns jogos e que não causem conflitos em outros, assim como características presentes em alguns jogos e que no futuro podem se tornar importantes ou obrigatórias. Com isso o requisito Multi-idiomas (Requisito 12), pode entrar na lista, uma vez que a escolha de um idioma fixo tem funcionalidade semelhante a um jogo com um único idioma, e esta é uma característica que tem se tornado cada vez mais comum nos jogos atuais.

Por fim, foram selecionadas as variabilidades do jogo, que são as características que são muito importantes em uns jogos e não em outros, ou que entrem em conflito com outras características.

Os requisitos 1 e 2, são relacionados ao estilo de jogo, *Multiplayer* ou *Single Player*. A escolha das duas ao mesmo tempo, gera um conflito, de modo que essa característica necessita ser uma alternativa. O Requisito Comunicação será uma característica opcional aos jogos *Multiplayer*, uma vez que nem todos os jogos

permitem essa comunicação entre os jogadores, por outro lado o requisito de *login* é obrigatório a todos os jogos *Multiplayer* para que possa ser identificado o jogador. Uma característica que tem se tornado tendência atualmente é realizar o *login* através das redes sociais, gerando como alternativa o *login* padrão com email e senha ou por rede social.

Outras características opcionais são a inserção de código (Requisito 3), pontuação (Requisito 8), questionário (Requisito 9), uso de mapa (Requisito 16) e construção de equipes (Requisito 18), devido a grande importância que possuem em alguns jogos, porém sem relevância em outros.

As características de Jogabilidade baseada em textos ou gráficos (Requisitos 10 e 11), também são paralelas, tornando-as características alternativas.

Após essa primeira análise nos jogos, foi realizada a análise nos artefatos de cada jogo, para verificar as características que cada um possuía, possibilitando identificar as características obrigatórias, opcionais e alternativas do artefato.

Entre as características e atributos presentes nos artefatos construídos, puderam ser identificadas:

1. Identificado (ID);
2. Nome;
3. Saúde*;
4. Imagem;
5. Dinheiro*;
6. População;
7. Cliente;
8. Texto/Descrição;
9. Status;
10. Peso;
11. Humor*;
12. Período em que o Artefato está;
13. Felicidade* ;
14. Classificação do Artefato;
15. Gerador de Imagem do Artefato;

* As atribuições de características humanas atribuídas ao artefato estão relacionadas as características identificadas do objeto que é construído no jogo

16. Uso de Componentes do Artefato.

Os resultados da matriz de requisitos dos artefatos dos jogos Construcionistas podem ser observados no Apêndice B.

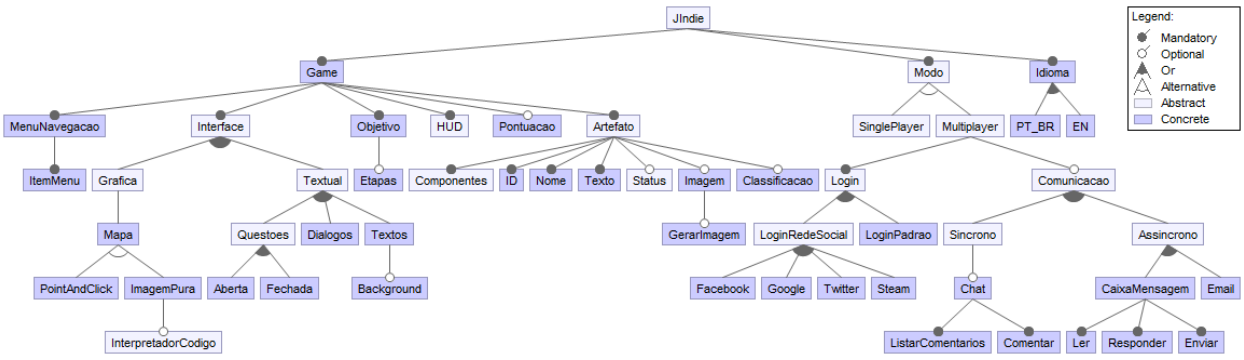
As características que apresentaram pouco uso ou que se demonstraram irrelevantes para outros jogos foram ignoradas. Essas características são listadas a seguir: Saúde (Requisito 3); Dinheiro (Requisito 5); População (Requisito 6); Cliente (Requisito 7); Peso (Requisito 10); Humor (Requisito 11); Período (Requisito 12); Felicidade (Requisito 13).

Entre as comunalidades estão as características mais presentes nos artefatos dos jogos e que não entram em conflitos com outros requisitos, como: Identificador (Requisito 1); Nome (Requisito 2); Texto (Requisito 8); Componentes (Requisito 16).

Como variabilidades dos artefatos foram identificadas as características que são importantes em alguns jogos, mas irrelevantes em outros: Imagem do artefato (Requisito 4); Status do artefato (Requisito 9); Gerador de imagem do artefato (Requisito 15).

Tendo identificado todas as comunalidades e variabilidades presentes que devem ser atendidas na LPS construída, foi possível criar um *feature model* (Figura 14) para representar os requisitos identificado:

Figura 14 - *Feature Model* da LPS JIndie



Fonte: Elaborada pelo autor.

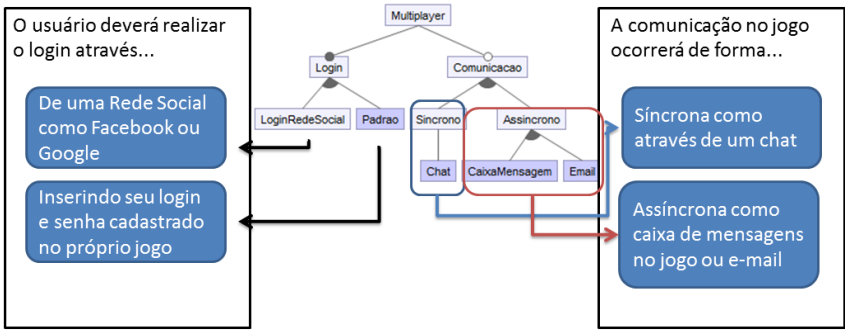
Posteriormente durante o processo de Implementação do Domínio, novas funcionalidades foram adicionadas ao *Feature model* da JIndie adicionando funcionalidades de apoio ao desenvolvedor. A nova versão da *Feature model* pode ser vista no Apêndice C.

Após ter a análise dos requisitos realizada, é interessante por parte do desenvolver da LPS, que também seja realizado a documentação da variabilidade,

tornando mais fácil e explícito o uso da LPS. Portanto, exemplificaremos abaixo algumas documentações da variabilidade realizadas na etapa de Engenharia do Requisito do Domínio.

Como vimos ao longo da Seção 3, a documentação da variabilidade pode ocorrer em forma de texto, relacionando-os aos pontos de variação. Na Figura 15 é documentada as variabilidades que ocorrem em um jogo *multiplayer*:

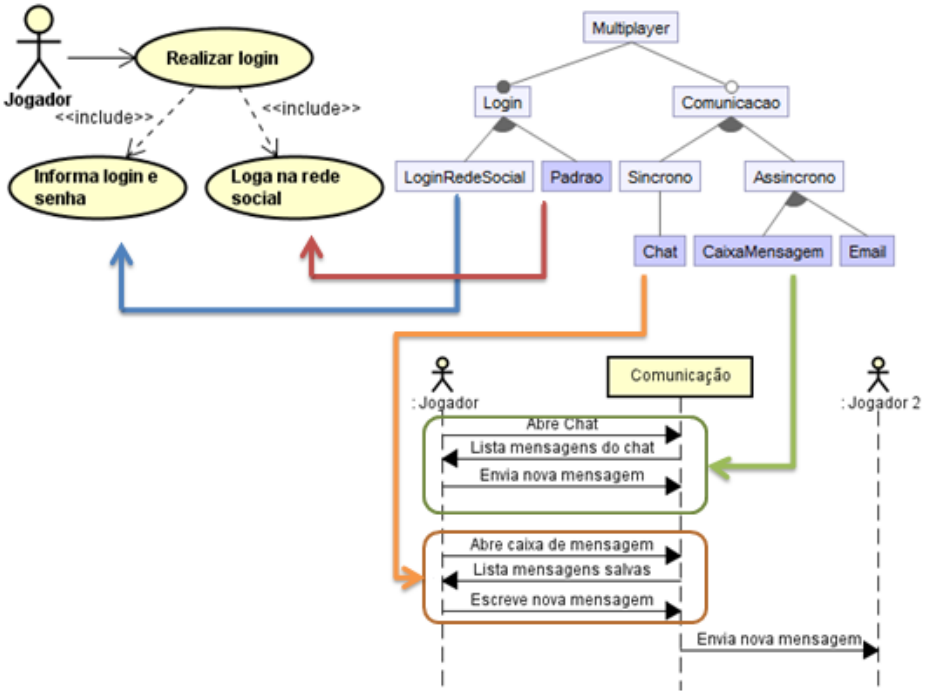
Figura 15 - Documentação Textual do jogo Multiplayer



Fonte: Elaborada pelo autor.

A mesma documentação, também pode ser realizada através de gráficos UML (Figura 16), demonstrando os processos realizados pelos *stackeholders*.

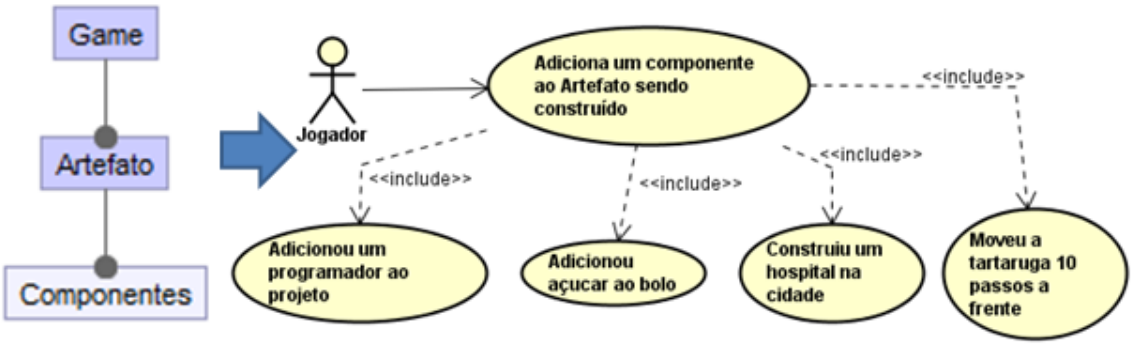
Figura 16 - Documentação por UML do jogo Multiplayer



Fonte: Elaborada pelo autor.

Na documentação abaixo, podemos ver através de um UML de Caso de Uso, como os componentes do artefato podem ser adicionados, em quatro diferentes tipos de jogos:

Figura 17 - Documentação do Componente do Artefato



Fonte: Elaborada pelo autor.

Na primeira situação, temos um jogo onde o artefato é um projeto de software e o componente pode ser um programador adicionado a equipe de desenvolvimento do projeto. No segundo jogo temos um bolo como artefato e os ingredientes como componentes. No terceiro jogo temos uma cidade como artefato, e as construções como os componentes. No quarto e último exemplo, temos o jogo Logo, onde o artefato é o desenho gerado e os componentes são os comandos inseridos pelo jogador.

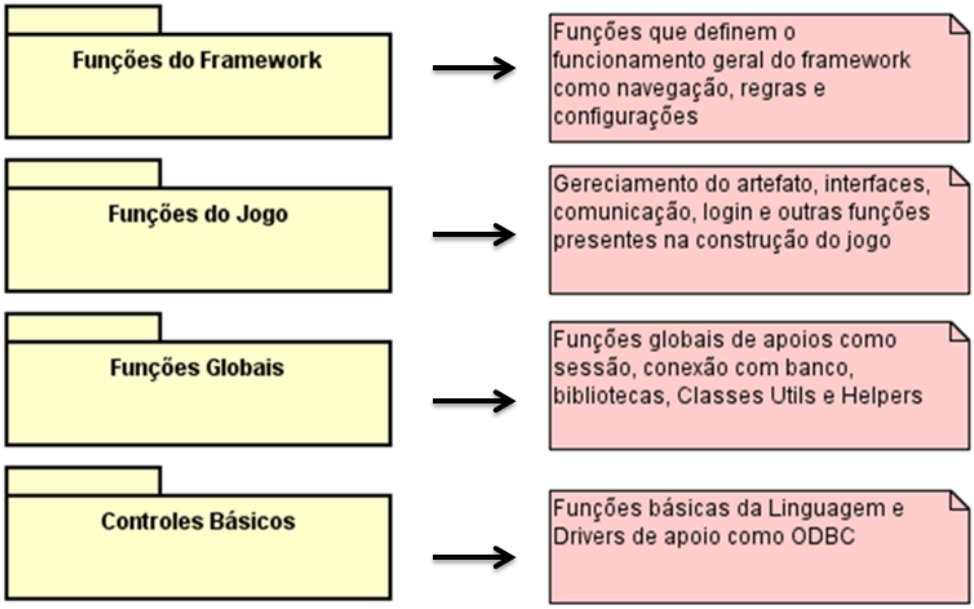
Na próxima Seção será apresentado o processo realizado na etapa do Projeto do Domínio.

6.1.2 Projeto do Domínio

No Projeto do Domínio tenta-se resolver os problemas identificados através dos requisitos listados na Engenharia de Requisitos do Domínio. Esta segunda parte da LPS permite uma visão abstrata da implementação do projeto. É a parte responsável por definir a estrutura da Linha de Produto de Software.

Para criar a arquitetura presente na LPS foram definidas quatro camadas representadas por pacotes conforme apresenta a Figura 18.

Figura 18 - Camadas da LPS JIndie



Fonte: Elaborada pelo autor.

Na camada de Framework ficará todas as funcionalidades responsáveis pelas configurações e regras do funcionamento de como o jogo deve fluir quanto a uma aplicação. Imaginando uma aplicação *web*, nesta camada ficariam todas as funcionalidades relacionadas à navegação entre uma página e outra, tal como qual *script* deve ser executado e em qual ordem, regras de segurança, tratamento de erros, entre outras funções. Basicamente esta camada será a responsável por gerenciar e iniciar as demais.

Na camada das Funções do Jogo encontram-se todas as funcionalidades relacionadas ao funcionamento do jogo. Enquanto na camada do Framework ficam os scripts relacionados à execução de códigos, regras de navegação e comportamento da aplicação. Na camada de Funções do Jogo estão os scripts que se relacionam com o domínio do jogo. Portanto, será nessa camada que estarão presentes os scripts relacionados ao artefato construído pelo jogador, regras do jogo, e escolhas como estilo de interface textual ou gráfica, objetivos e sistema de pontuação.

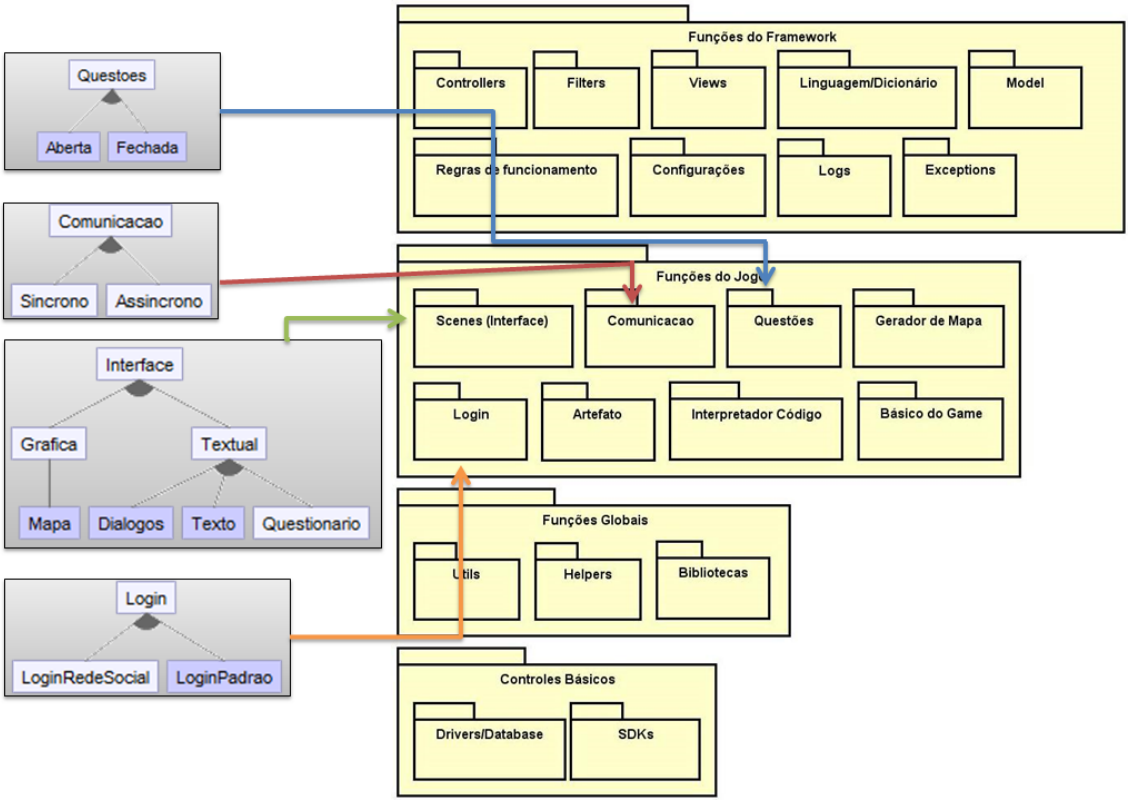
Na camada de Funções Globais estão os scripts mais simples que não fazem parte do funcionamento geral da aplicação e nem do domínio da aplicação, mas que ainda apresentam papel importante como dar suporte às duas camadas citadas anteriormente. Nessa camada será possível encontrar *utils, helpers* e bibliotecas.

Na camada de Controles Básicos estão presentes as funcionalidades ofertadas pela própria linguagem de programação, independente da aplicação.

Com as camadas definidas, o passo seguinte foi apresentar uma arquitetura baseada na visão de desenvolvimento (*Development View*). Através do Diagrama de Pacotes (Apêndice D) foi definida uma estrutura que estivesse apta a atender as necessidades dos requisitos da LPS.

Na Figura 19 é possível identificar em quais pacotes estariam certos pontos de variação da LPS JIndie:

Figura 19 - Diagrama de Pacotes e Pontos de Variação

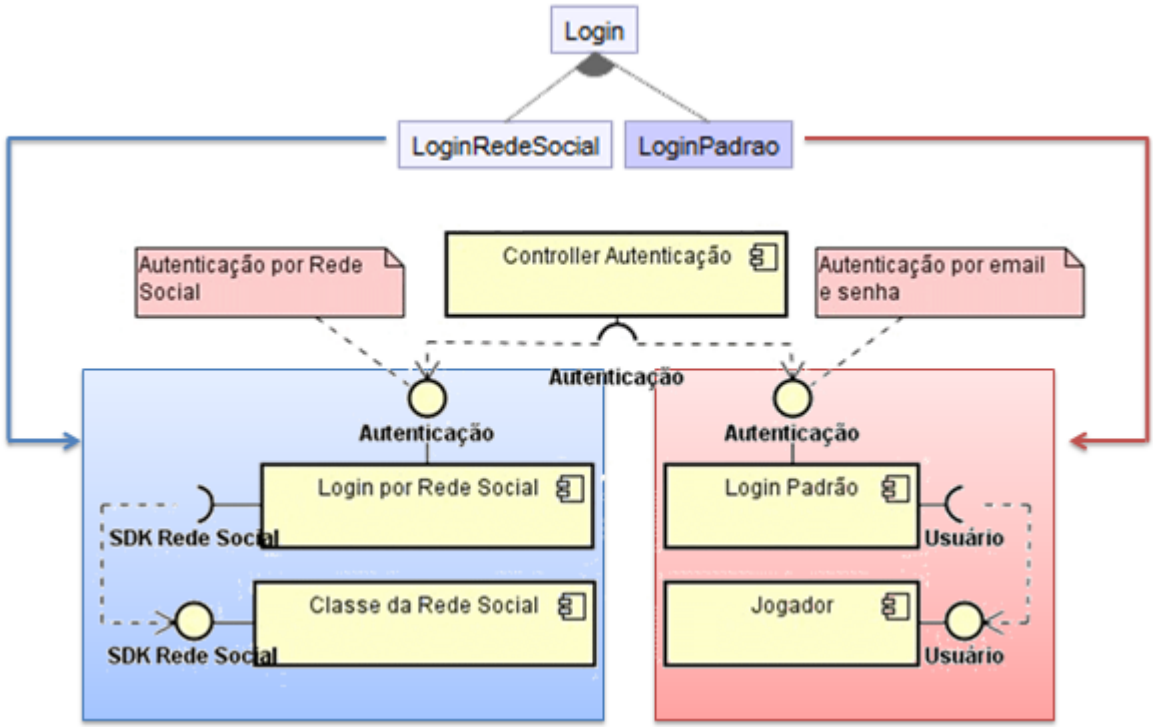


Fonte: Elaborada pelo autor.

Além dos pontos de variação ilustrados nas imagens acima, ainda há relações entre esses pacotes e outros pontos de variação.

O ponto de variação do *login*, por exemplo, além de estar relacionado com o pacote envolvendo as funções de *login*, também possui relação com os pacotes de Banco de Dados (*Database*) para realizar um *login* padrão através do e-mail e senha, como também pode estar relacionado com o pacote de SDK's (*Software Development Kit*), onde terá bibliotecas de conexão com as redes sociais (Figura 20).

Figura 20 – Componentes da autenticação do usuário



Fonte: Elaborada pelo autor.

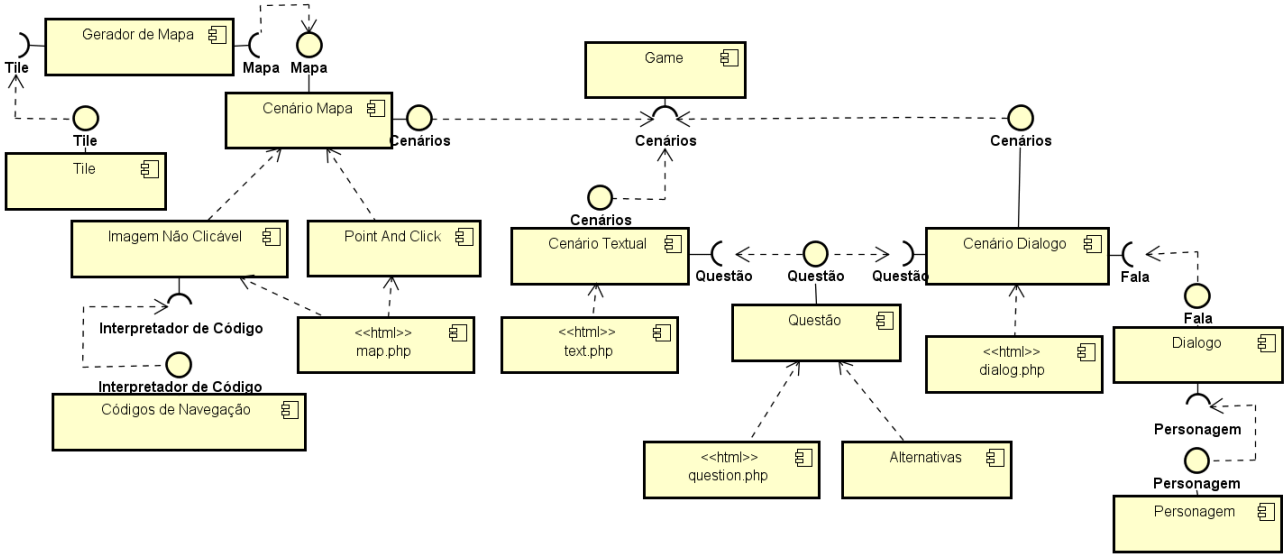
No Projeto do Domínio, também é possível representar a estrutura que irá ser implementada para atender os requisitos, através do Diagrama de Componentes. Como vimos na Seção 3, o desenvolvimento do Projeto de Domínio apresenta vantagens ao ser desenvolvido baseado em componentes e interfaces, uma vez torna possível o processo de reuso e troca de componentes.

Para desenvolver a arquitetura para a construção de jogos construcionistas atendendo as necessidades, flexibilidade e reuso foram criados componentes e interfaces na arquitetura. No componente principal das funções do jogo, chamada de Game, foram estabelecidas as conexões existentes entre outros componentes através do uso de interfaces. Esse processo facilitou a seleção dos pontos de variação, como é o caso das características opcionais, onde um componente poderia ser removido sem causar problema à estrutura, uma vez que as funcionalidades estão em componentes separados. Outro fator que facilita a construção da arquitetura através de componentes pode ser observado nas características alternativas como o caso da jogabilidade baseada em textos ou gráfico, nos tipos de *login*, escolha de idioma ou na comunicação, que por contar com interfaces apropriadas para cada característica, torna-se fácil a troca de um componente por outro.

Para facilitar o processo de desenvolvimento do programador, componentes como geradores de mapas baseados em blocos (*tiles*) e interpretadores de comandos, que lêem, interpretam e executam o código do jogador foram adicionados.

Na Figura 21, podemos ver o exemplo do Diagrama de Componentes construído para atender os requisitos do tipo de jogabilidade, que conforme a Figura 19 pode ser um cenário baseado em um mapa, diálogos, textos ou questões:

Figura 21 - Diagrama de Componentes da Interface da LPS JIndie



Fonte: Elaborada pelo autor.

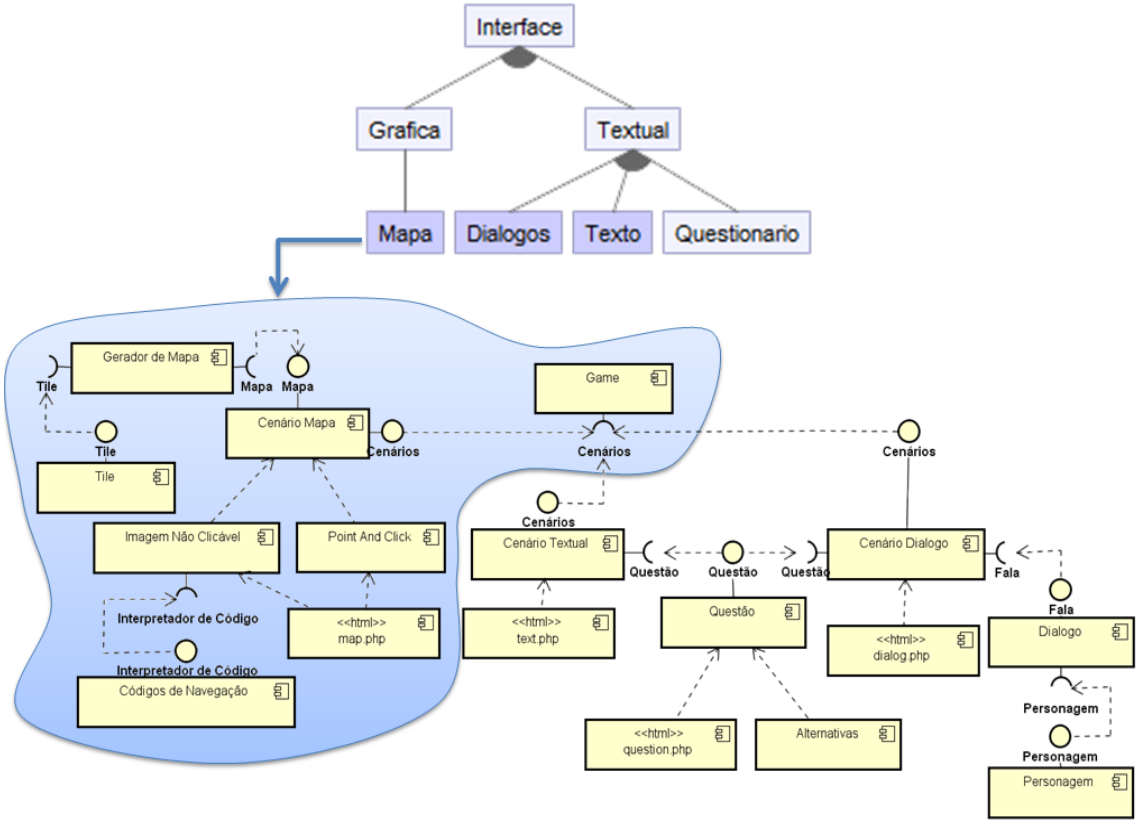
A visão do Diagrama de Componentes pode causar certo mau entendimento pelo o programador da aplicação, ao tentar saber quais componentes estão relacionadas à quais características, de forma que a documentação da variabilidade pode suprir tais dúvidas.

Na documentação abaixo, podemos ver quais os componentes estão relacionados a construção de um cenário baseado em um mapa ou desenho.

O requisito de construção de cenário com mapa (Figura 22) envolve componentes que podem ser uma “Imagem Não Clicável” ou um Mapa *Point-And-Click*. No caso do componente “Imagem Não Clicável”, a imagem pode ser gerada através de um código inserido pelo jogador, como é o caso da Linguagem Logo. Vale ressaltar que a sintaxe do código e regras dos comandos que o usuário pode executar também é um componente que pode ser trocado ou reusado, uma vez que nem todos os jogos terão as mesmas regras de código a ser interpretadas. Já no mapa *Point-And-Click*, a interação ocorrerá através do mouse clicando em partes da

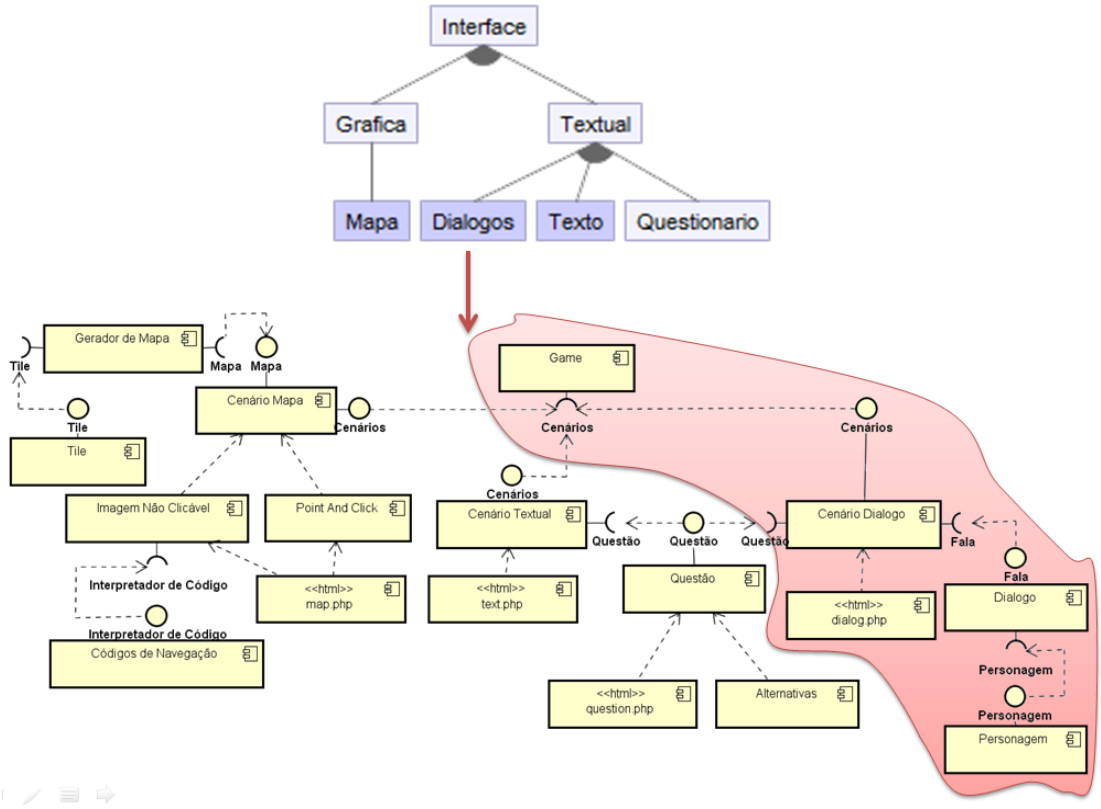
imagem criada, como é o caso de *SimCity*. Todavia, para ambos os casos, a construção do Mapa contará com um componente Gerador de Mapas baseados em blocos chamados de *Tile*.

Figura 22 – Componentes de Cenário Baseado em Mapa



Fonte: Elaborada pelo autor.

Figura 23 - Componentes de Cenário Baseado em Diálogos



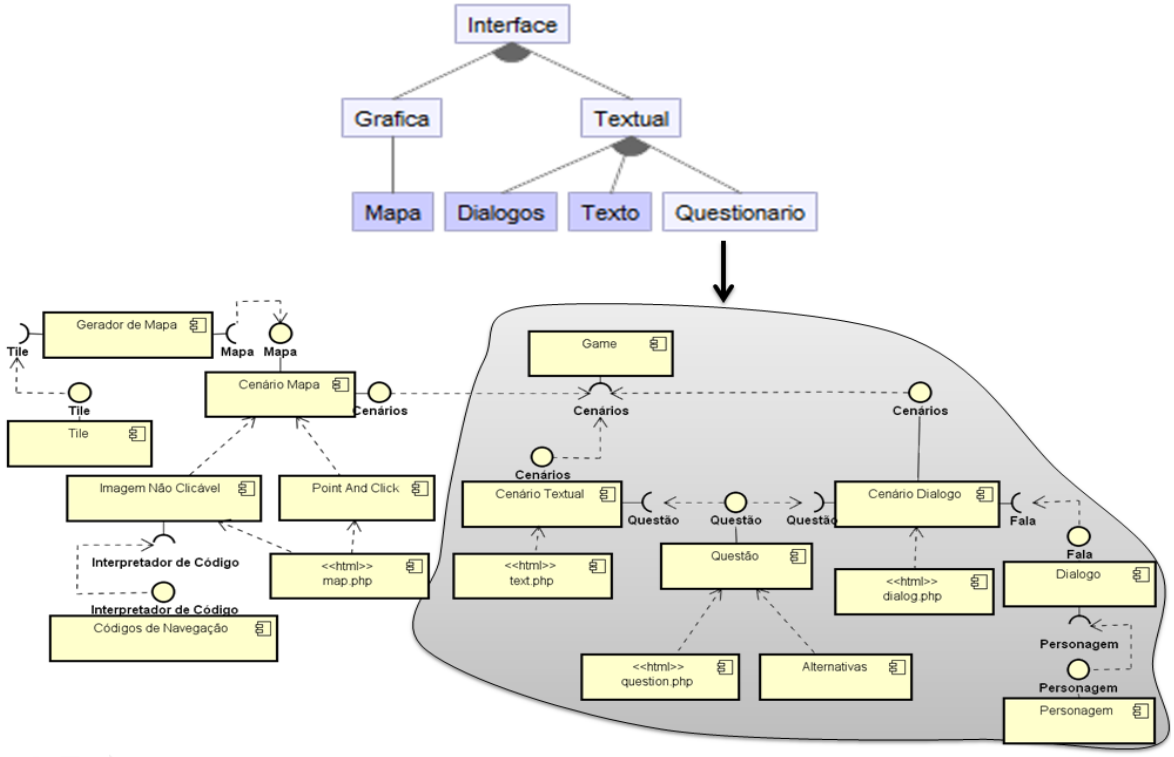
Fonte: Elaborada pelo autor.

A construção do cenário baseado em diálogos (Figura 23), contará com um componente chamado de Dialogo, responsável pela fala de algum personagem do jogo. Esse tipo de cenário pode ser observado no jogo Sim Investigador, onde são apresentadas ao jogador as falas dos personagens na narração de um caso. Os componentes envolvidos nesse tipo de requisito podem ser identificados através da figura abaixo:

Na construção de um cenário baseado em textos, por se tratar de algo mais simples, os componentes envolvidos relacionam-se apenas com o componente Textual e seu arquivo html.

Na ocasião de um cenário fazer o uso de questionários, este poderá envolver outros componentes textuais como o diálogo ou texto:

Figura 24 - Componentes de Cenário Baseado em Questionário

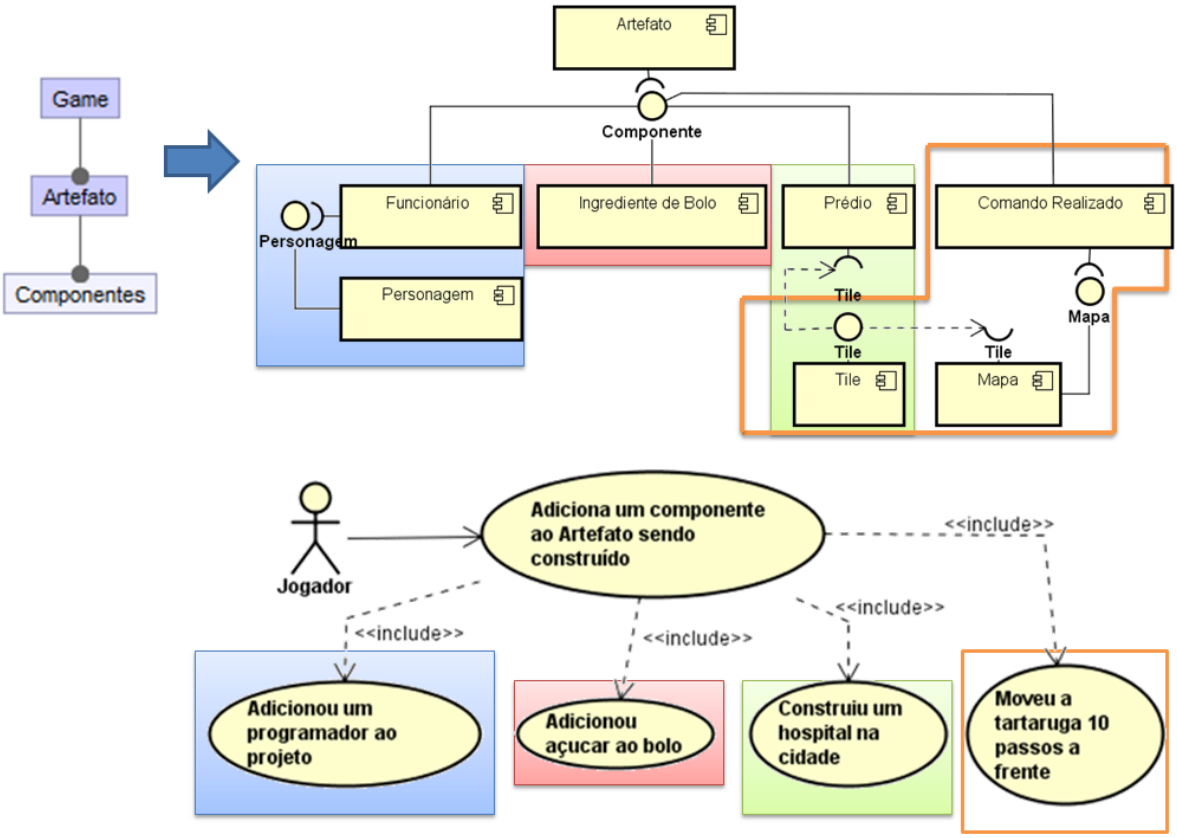


Fonte: Elaborada pelo autor.

A questão apresentada ao jogador, ainda poderá ser tanto aberta com uma resposta discursiva, como também fechada com alternativas. Esse tipo de situação pode ser observado nos jogos como Sim Investigador e Gene2.

A adição de um componente ao artefato que está sendo construído no jogo pode ocorrer facilmente através do uso de interfaces. No exemplo abaixo, podemos ver como seria a adição dos componentes ao artefato em quatro diferentes tipos de jogos. O diagrama de Caso de Uso apresentado junto à imagem tem como finalidade identificar qual componente do artefato está relacionada à qual ação do jogo. No caso do componente “Prédio”, semelhante ao jogo SimCity, é adicionado junto a esse componente, um *Tile* que representa um bloco do mapa a ser adicionado a cidade que está sendo construída. O mesmo ocorre no componente “Comando Realizado”, nos jogos semelhante a Linguagem Logo, onde o desenho formado através da interpretação dos comandos é representado através do Gerador de Mapas com o uso de vários *tiles*.

Figura 25 - Diagrama de Componentes do Artefato



Fonte: Elaborada pelo autor.

Na próxima seção será apresentado o processo de desenvolvimento do *framework* a ser utilizado na construção dos jogos.

6.1.3 Implementação do Domínio

Na Implementação do Domínio é realizado um framework, arcabouço ou protótipo que sirva para a implementação dos produtos finais baseado na arquitetura planejada na etapa anterior.

Neste trabalho foram realizadas as implementações de dois frameworks. Foram realizadas duas implementações para que fosse possível avaliar a LPS em duas situações diferentes, a primeira quanto a um jogo desenvolvido em PHP e outra em relação à técnica de compilação condicional em Java.

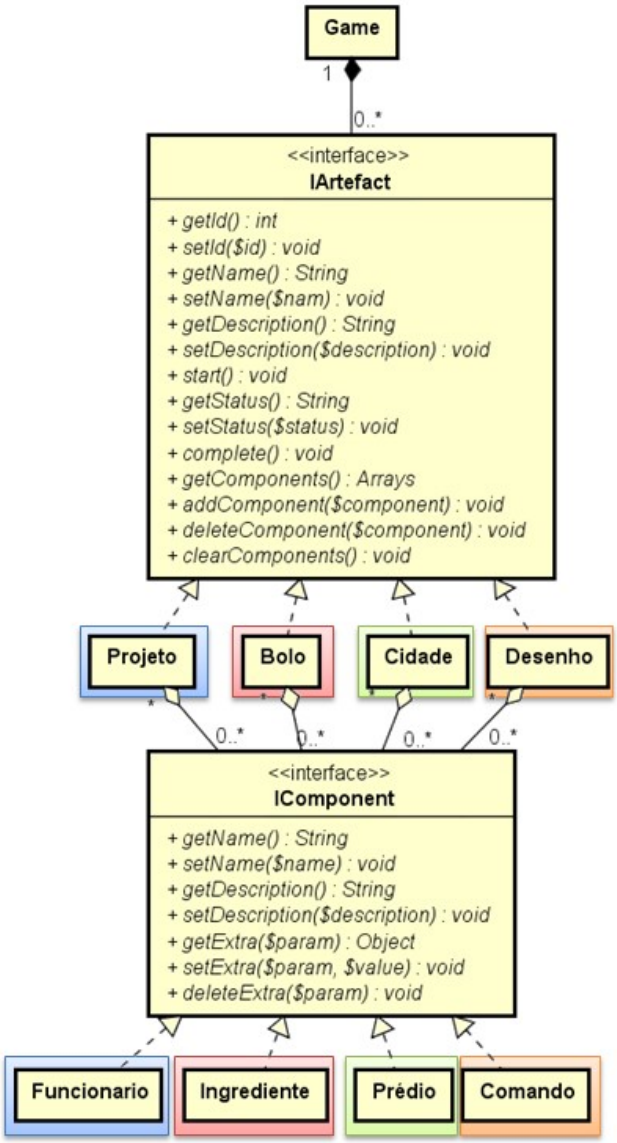
A primeira implementação desenvolvida, utilizou a linguagem *Hypertext Preprocessor* (PHP), que é uma linguagem amplamente utilizada no desenvolvimento *web* e que permite desenvolvedores escreverem páginas que serão geradas dinamicamente de forma rápida, e com um fácil processo de

aprendizagem (THE PHP GROUP, 2016). A escolha da linguagem está relacionada à praticidade de implementar um conteúdo *Multiplayer* e devido a alguns jogos rodarem diretamente pelo browser. Essa escolha da linguagem também sofreu influência relacionada a produção de um dos jogos, o Sim Investigador, devido ao fato do desenvolver do jogo original poder participar da avaliação do desenvolvimento do jogo com o uso da LPS, possibilitando comparações relacionadas a tempo de desenvolvimento, complexidade, número de linhas e esforço para a implementação do jogo.

Nesta primeira implementação foram utilizadas técnicas de implementação da variabilidade como agregação e delegação, parametrização, herança e técnicas de padrões de projetos com o *factory*.

Para a implementação do artefato e seus componentes foi utilizada a técnica de delegação, no qual a função de cada componente estaria separada do artefato, possibilitando assim a flexibilidade. No exemplo da Figura 26, podemos verificar como quatro artefatos de diferentes jogos podem trabalhar com quatro componentes diferentes através da técnica de delegação.

Figura 26 - Delegação dos Componentes do Artefato



Fonte: Elaborada pelo autor.

Com a separação das funcionalidades que um componente possui em relação ao artefato utilizando a interface IComponent, torna-se possível a adição de diferentes componentes aos artefatos.

Um exemplo da implementação da variabilidade através dos Padrões de Projetos pode ser observado no *login*, através de um *factory*. A técnica *factory* permite a um cliente poder instanciar uma classe conforme uma interface em particular ou protocolo sem saber precisamente a qual classe que está sendo obtida (ELLIS et al., 2007).

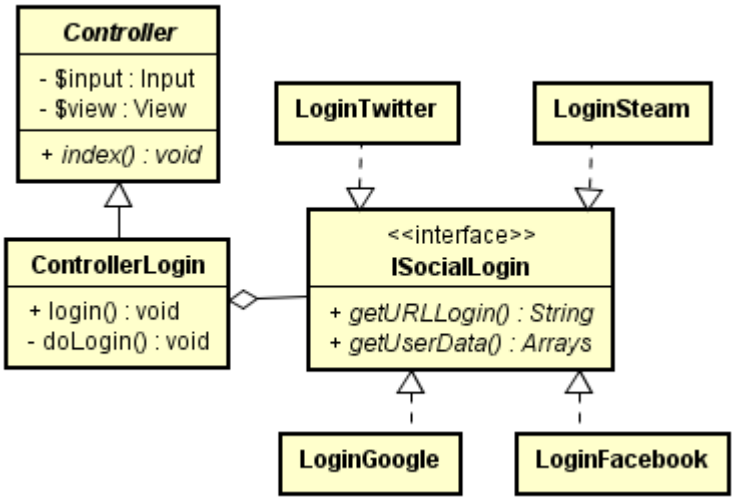
No exemplo abaixo, temos uma classe do tipo Controller que solicita um login através da rede social twitter:

Figura 27 - Seleção da rede social para o login

```php
<?php

class ControllerLogin extends Controller{

    public function __construct() {
        parent::__construct();

        $this->loadLibrary('loginTwitter');
    }

    public function login() {...}

    public function doLogin() {...}

}
```

Fonte: Elaborada pelo autor

Figura 28 - Representação em Diagrama de Classe do Login por rede social

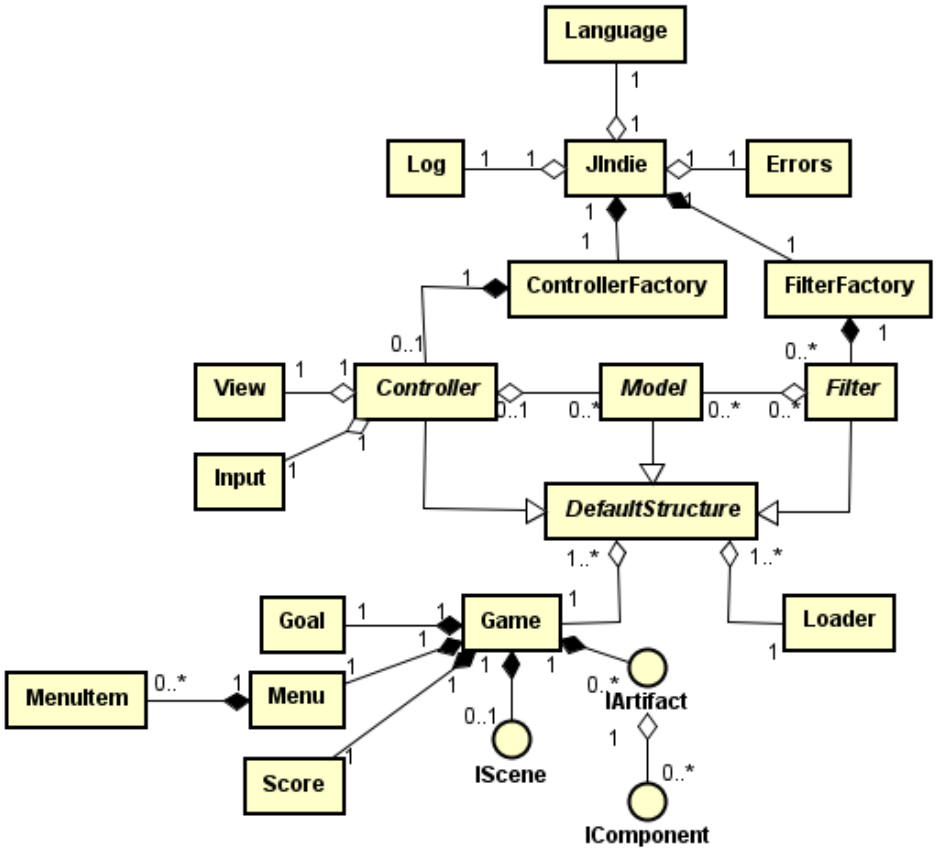


Fonte: Elaborada pelo autor.

A plataforma comum desenvolvida além de possuir diversas funcionalidades completas previstas nos requisitos, necessitando apenas uma configuração prévia, como o caso do *login* e *email*, onde são necessárias apenas as credências de autenticação utilizadas na aplicação final, também conta com diversas funcionalidades de apoio como acesso a banco de dados, filtros de acesso de segurança, tratamento de erros e gerador de logs.

A estrutura básica da plataforma gerada pode ser observada através do Diagrama de Classes a seguir:

Figura 29 - Estrutura básica da implementação JIndie



Fonte: Elaborada pelo autor.

Todo o controle do sistema será realizado através de uma classe central, nomeada de JIndie. Está classe terá o suporte das classes de tratamento de erros, gerador de log e controle do idioma (Language).

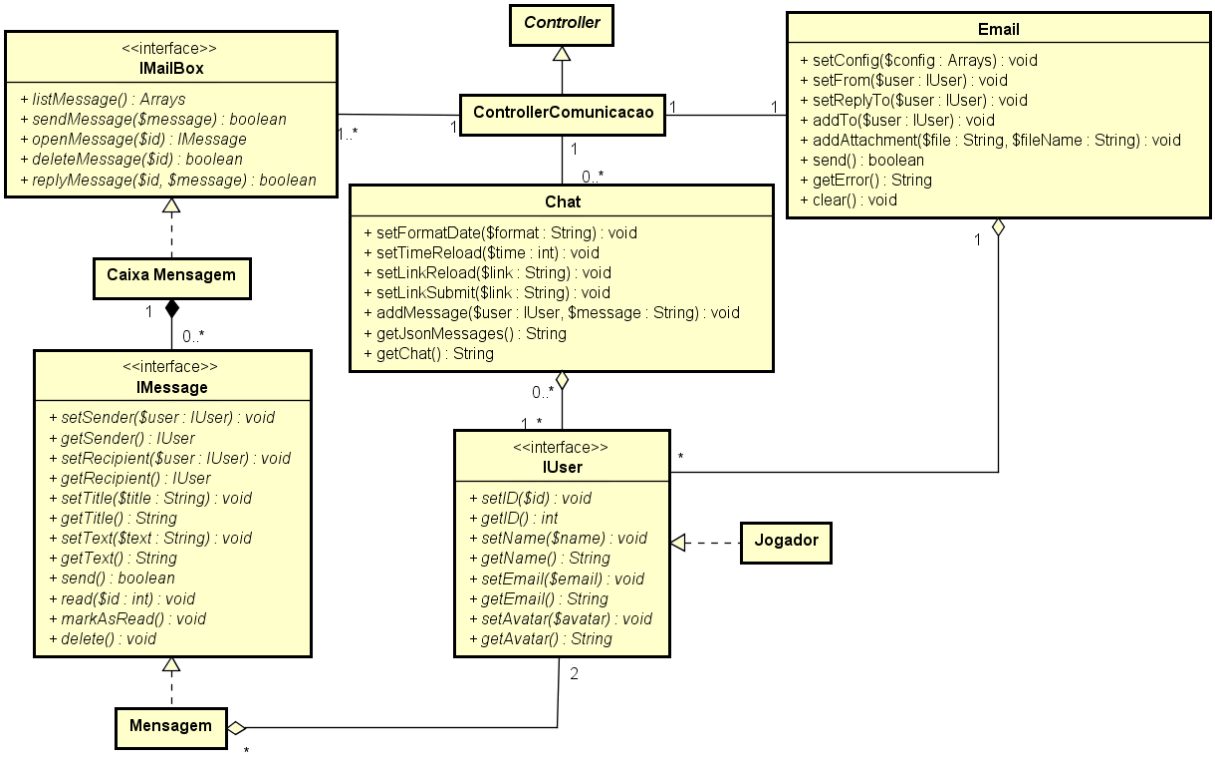
O desenvolvedor da aplicação final poderá controlar os acessos dos usuários através de filtros (Filter), que são verificados a cada novo acesso pelo FilterFactory. Após a execução do filtro de segurança, caso o acesso seja liberado, é iniciada uma classe Controller, gerada pelo ControllerFactory, de acordo com o local que o usuário esteja tentando acessar.

As classes Controllers, Model e Filter têm uma estrutura em comum (DefaultStructure), que possibilita carregar bibliotecas, configurações, *helpers*, outros models e administrar o comportamento do jogo através da classe Game.

Através da classe Loader é possível acessar outras classes que possuam funcionalidades relacionadas aos requisitos do domínio, ou que sirvam de apoio para criar a estrutura básica do jogo. Essas funcionalidades são:

- Chat;
- E-mail;
- Gerador de Imagem;
- Interpretador de Códigos;
- Login por rede social;
- Validador de formulário;
- Acesso a banco de dados;
- Controle de paginação;
- Controle de Cache;
- Controle de cookies e sessão do usuário;
- Controle de questões abertas e fechadas;

Figura 30 - Diagrama de Classe dos componentes de comunicação



Fonte: Elaborada pelo autor.

A documentação da variabilidade durante a etapa de implementação do domínio foi realizada através do uso do Diagrama de Classes. Na Figura 30 é possível ver as classes e interfaces relacionadas à *feature* envolvendo a comunicação do jogo. Um exemplo da documentação realizada nesta etapa envolvendo as classes de comunicação pode ser vista no Apêndice E. Nesta

documentação é possível observar quais classes e interfaces se relacionam com quais requisitos do projeto.

Os apêndices F, G, H e I mostram os códigos desenvolvidos nas principais classes utilizadas durante o processo de desenvolvimento dos jogos no estudo de caso. Os códigos envolvem a classe principal do controle do jogo (Game), a classe responsável por interpretar os comandos inseridos pelos usuários (CodeReader), a classe responsável por gerar os desenhos e mapas dos jogos (MapGenerator) e a interface usada para a construção do Artefato (IArtefact).

6.1.3.1 LPS implementada no CIDE

A segunda implementação da arquitetura foi realizada através da linguagem Java. Java é uma tecnologia que permite o desenvolvimento de sistemas web, executar jogos, fazer upload de fotos, bater papo on-line, fazer *tours* virtuais e usar serviços, como treinamento on-line, transações bancárias on-line e mapas interativos (ORACLE, 2016).

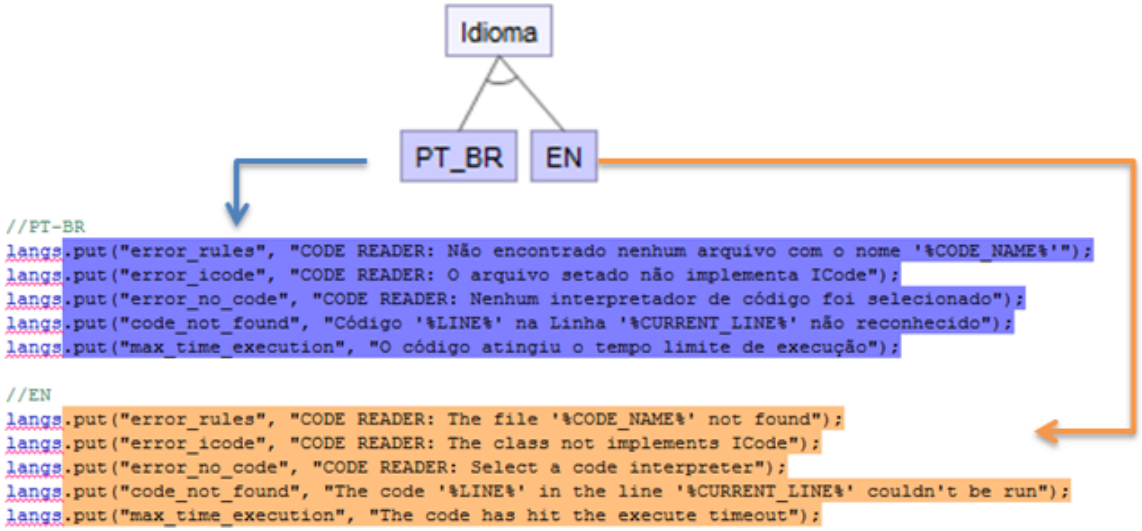
Após a produção e validação da primeira versão da LPS JIndie em PHP, resolveu-se desenvolver uma segunda versão em Java. A segunda versão desenvolvida teve como finalidade analisar a técnica de compilação condicional unida a uma ferramenta de suporte ao desenvolvimento de LPS. A ferramenta escolhida foi o CIDE, apresentado na Seção 4.2.1, por ser uma ferramenta que possui um foco maior na técnica de compilação condicional.

Tanto a versão em PHP quanto a versão em Java foram baseadas na arquitetura desenvolvida no Projeto do Domínio. Deste modo é admissível que independente da linguagem escolhida é possível desenvolver uma plataforma com a arquitetura planejada.

A versão em Java também segue características semelhantes à versão em PHP, assim como a sua estrutura de classes padrão. As principais modificações ficaram nos requisitos opcionais e alternativos, de forma que o código pode se tornar mais limpo de acordo com as necessidades do produto final.

Na Figura 31 é possível observar o uso da compilação condicional através do CIDE na *feature* de idioma. Nela os desenvolvedores dos produtos finais ao escolherem quais idiomas não estarão presentes nos seus jogos, estarão removendo todo o código que não for necessário.

Figura 31 - Escolha do Idioma



Fonte: Elaborada pelo autor.

O uso da compilação condicional além de reduzir linhas de códigos e arquivos que são processados sem utilidade para o jogo produzido, também pode reduzir a complexidade do código reduzindo o número de estruturas de decisão no código.

A comparação entre utilizar uma ferramenta de suporte ao desenvolvimento de LPS e não a utilizar, assim como as vantagens observadas ao optar pelo uso da compilação condicional, pode ser observada na Seção 7.4 deste trabalho.

6.1.4 Teste do Domínio

Os testes realizados nessa etapa têm como finalidade verificar se a LPS está atendendo aos requisitos listados na etapa de Engenharia de Requisitos do Domínio.

Os artefatos da LPS e a plataforma desenvolvida nas etapas anteriores permitem certa variedade de como o jogo pode ser implementado, podendo o produto final ser um jogo de construção de cidade, desenhos, narrações entre outras opções. Deste modo a utilização de teste por força bruta se tornaria um processo lento e inviável, uma vez que seria necessário criar um produto para cada combinação possível para se obter a conclusão sobre o teste. Sendo assim, a estratégia inicialmente adotada nesse trabalho foi o *Commonality and Reuse Strategy*, que visa criar pequenas aplicações para testar todas as comunalidades e reaproveitando as mesmas aplicações já criadas para validar as variabilidades, apenas trocando partes do código onde ocorrem as variações.

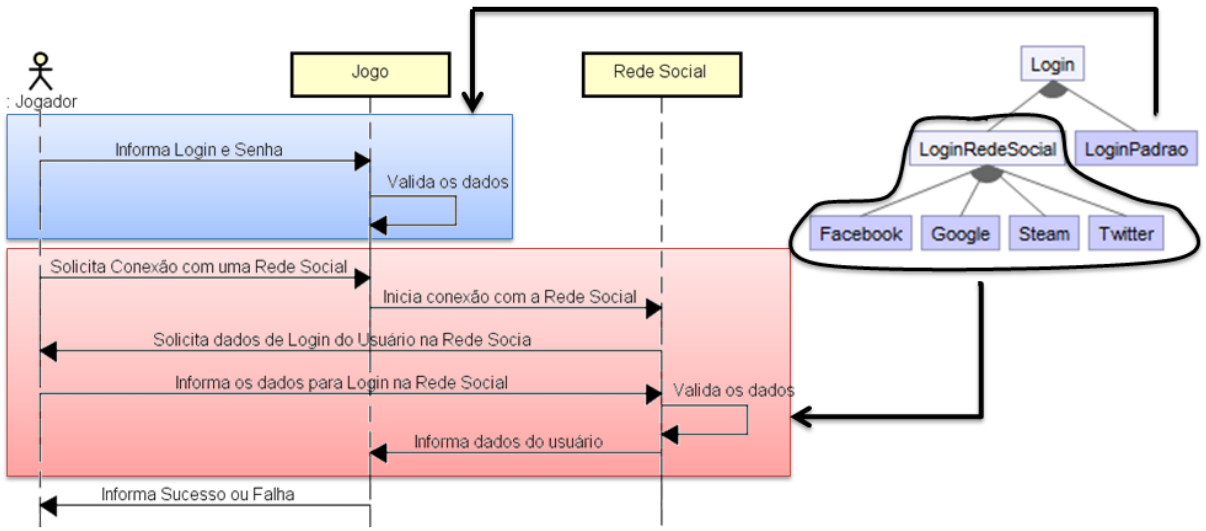
No exemplo do *Commonality and Reuse Strategy,* podemos exemplificar com o caso do *login*, no qual o teste ocorreu pegando as informações do usuário através do *login* padrão e posteriormente através das redes sociais.

Durante o planejamento do teste foi verificado um maior esforço e cuidado durantes os testes envolvendo as redes sociais. Ao realizar os testes com *login* por rede social existem algumas preocupações relacionadas às dependências de SDK's externas.

No *login* padrão, o usuário insere seu *login* (ou e-mail) e senha, que serão verificados e trabalhados por um Model ou Controller, retornando os dados do usuário caso o *login* tenha ocorrido com sucesso. Portanto, sua complexidade é baixa, sendo apenas uma simples relação entre script e banco de dados, que será implementada no produto final. Por outro lado, o *login* através de rede social exige o cuidado de deixar previamente implementada as funcionalidades de *login* por Facebook, Google, Steam e Twitter de forma que a única preocupação do desenvolvedor da aplicação final será gerar as credências exigidas pela API da rede social que pretende utilizar. Além de verificar se o *login* está sendo realizado de forma correta entre as quatro redes sociais já disponibilizadas, ainda há a preocupação de verificar a viabilidade de adicionar novas redes sociais futuras utilizando a Interface.

A Figura 32 apresenta o caso de teste realizado no *login* representando o cenário através do Diagrama de Sequência:

Figura 32 - Diagrama de Sequência do Teste de Login



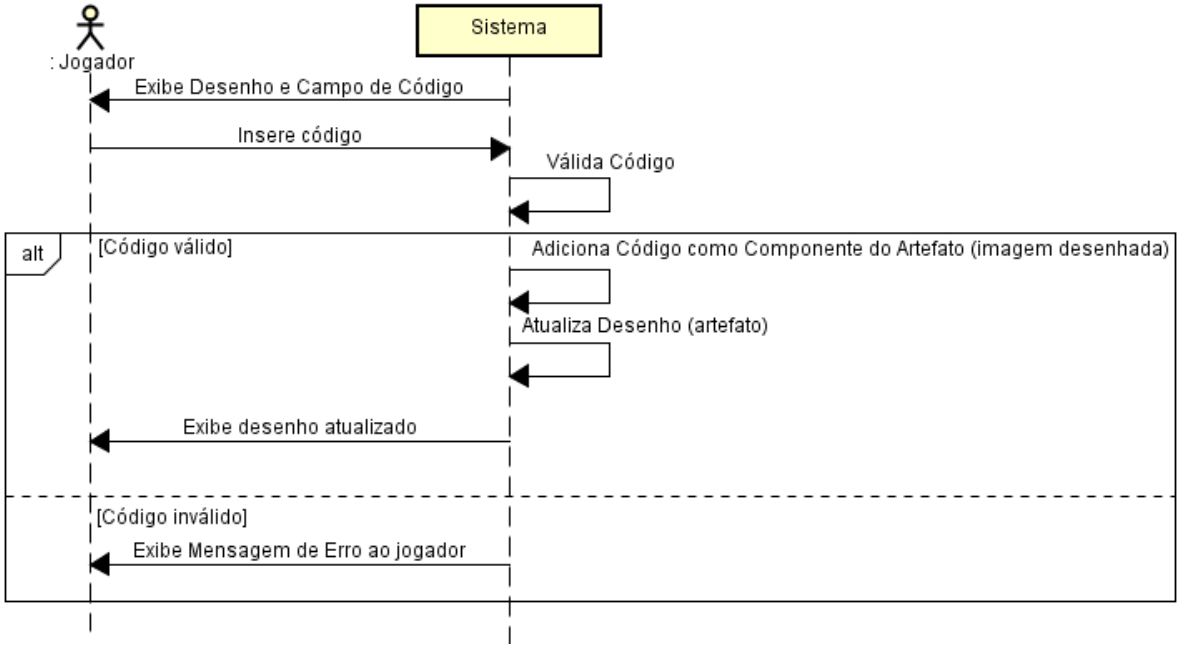
Fonte: Elaborada pelo autor

Nesse teste foi criada uma pequena aplicação apenas para verificar se o *login* recebia as informações corretas respeitando o diagrama de sequência acima.

Para reforçar ainda mais essa etapa de validação, foi utilizada a técnica *Sample Application Strategy* (apresentada na Seção 3.1.4), que visa criar algumas amostras completas para testar se as necessidades foram atingidas de forma satisfatória ou não. As aplicações criadas corresponderam a três jogos utilizando a plataforma em PHP e um jogo utilizando a plataforma Java.

O exemplo de planejamento e execução do teste no *Sample Application Strategy* pode ser visto através do artefato construído no jogo Logo:

Figura 33 - Diagrama de Sequência do Artefato do jogo Logo



Fonte: Elaborada pelo autor

O jogo deve exibir ao jogador um mapa que é construído através da entrada de códigos pelo próprio jogador. O jogador então poderá inserir novos códigos que serão validados pelo sistema do jogo. Caso o código inserido não contenha erros, o sistema irá adicionar os novos códigos como componentes ao artefato que está sendo construído, atualizar o artefato e exibir o novo desenho ao jogador. Contudo, caso o código apresente erros, uma mensagem deverá ser apresentada ao jogador.

Através desse caso de teste é possível testar o artefato e seus componentes, como também a interface do jogo baseada em “Mapa” com interpretador de códigos.

Na próxima seção desse trabalho será apresentado o Estudo de Caso realizado, com a finalidade de validar a LPS com o desenvolvimento de alguns jogos

construcionistas e verificar também o desenvolvimento dos jogos com e sem o uso de uma ferramenta de suporte a técnicas de implementação da variabilidade.

7 ESTUDO DE CASO

Nesta seção será apresentado o Estudo de Caso realizado envolvendo a produção de quatro jogos usando a LPS proposta.

7.1 Metodologia

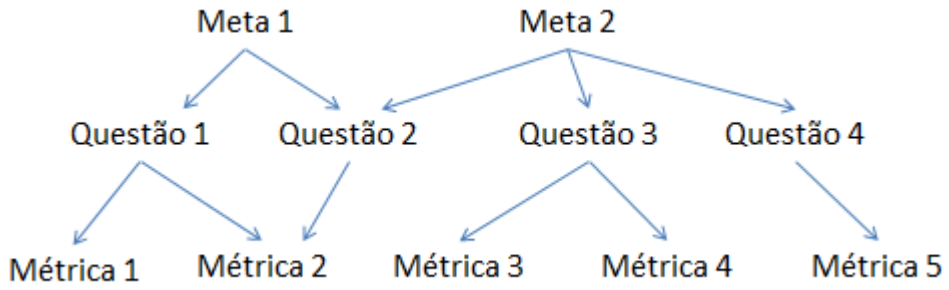
Um Estudo de Caso seguindo o modelo de Runeson e Höst (2009) passa pelas etapas de planejamento, realização e coleta dos resultados. A análise dos dados coletados pode ocorrer de forma quantitativa ou qualitativa. A combinação dos dois termos tende a providênciar um melhor entendimendo do fenômeno estudado.

Nessa pesquisa foi realizado um Estudo de Caso de caso exploratório visando apresentar o construcionismo e LPS ao mesmo tempo em que avalia uma nova proposta.

7.2 Definição

Para a definição dos dados que serão coletados no Estudo de Caso, optou-se pelo uso do paradigma Goal Question Metric (GQM), que no português significaria Meta, Questão e Metrica. No GQM, primeiro se define os objetivos que se pretende alcançar, seguidos das perguntas que surgem dos objetivos, e por fim as métricas que servem para responder as perguntas (RUNESON E HÖST, 2009). A figura 34 ilustra como as métricas estão relacionadas diretamente com os objetivos.

Figura 34 - Representação visual do GQM



Fonte: Elaborada pelo autor.

Partindo desse principio nosso estudo de caso possui como Meta:

1. Verificar se a LPS JIndie é capaz de construir diferentes jogos construcionistas mantendo a qualidade e reduzindo custos, tempo, complexidade e esforço do desenvolvedor;
2. Avaliar a influência do uso da compilação condicional nos jogos construídos com a LPS.

As questões seriam:

1. Qual o tempo gasto na produção dos jogos?
2. Quanto de Linha de Códigos o desenvolvedor do jogo precisou criar?
3. Qual a Complexidade Ciclomática dos códigos criados?
4. Um jogo construído com a LPS apresentou desempenho melhor do que a sua versão original?
5. A construção de um jogo com compilação condicional apresentou facilidade no desenvolvimento comparado ao jogo construído sem a compilação condicional?

As métricas escolhidas foram:

1. Quantidade de tempo gasto na produção dos jogos;
2. Quantidade de Linhas de Código que o desenvolvedor criou;
3. Complexidade Ciclomática dos códigos criados pelo desenvolvedor;
4. Nivel de satisfação do desenvolvedor dos jogos com a LPS;
5. Comparação das métricas anteriores em relação aos jogos com o uso e sem o uso da compilação condicional

Linhas de Código ou Lines of Code (LOC), segundo Barb et. al. (2014) é uma métrica comumente utilizada para medir tamanho de softwares, complexidade, produtividade do desenvolvedor, esforço e custo do desenvolvimento. Sendo através da descrição do tamanho do número de linhas de código possível calcular quanto tempo o projeto necessitará para estar completo, quanto custará, assim como medir a produtividade.

Já a Complexidade Ciclomática (CC) é uma métrica proposta por McCabe para avaliar a complexidade de softwares baseado nas estruturas das linhas de

códigos (WATSON et al., 1996). A CC é baseada na quantidade de caminhos na estrutura da lógica de decisão utilizada em um único modulo do software. Na análise é usado o diagrama de fluxo do código, onde cada linha de código é representada por um nó e cada possível caminho é representado por uma aresta. A representação da CC pode ser realizada através da formula:

$$CC = A - N + 2$$

Onde “A” representa o número de arestas; e “N” o número de Nós. Segundo McCabe (1976) quanto maior a CC, maior será a chance de o sistema apresentar erros durante a manutenção no software. Muitos programas possuem módulos com complexidade por volta de 40-50 pontos, porém estudo realizado por Clark et al (2008) com a CC em programas comerciais, demonstra que CC acima de 20, apresenta um grande risco e necessitam ser reformulados. Deste modo, nesse Estudo de Caso a métrica de CC tomará como hipótese nula qualquer valor acima ou igual a 21 pontos.

Além das métricas adotadas acima, ainda foi realizada uma comparação entre a versão do jogo RoboCode construído através da LPS em PHP e a versão do jogo construída através da LPS em Java. A comparação buscou analisar as facilidades e melhorias que poderiam ser obtidos com o uso da ferramenta CIDE na implementação da variabilidade via compilação condicional.

7.3 Planejamento

Após a definição, foi realizado o planejamento. O Estudo de Caso envolveu a produção de quatro jogos, sendo três com a versão da plataforma comum em PHP e um quarto jogo envolvendo a versão em Java.

Os jogos envolvidos na produção foram: Sim Investigador, Linguagem Logo e RoboCode. A produção do jogo Sim Investigador foi selecionada devido ao fato de se ter informações relacionadas ao tempo de produção, esforço realizado e acesso ao código original, possibilitando testes e comparações entre a versão produzida com a LPS JIndie e o código original. Os jogos da Linguagem Logo e RoboCode também passaram pelo processo de avaliação, entretanto o maior foco nestas produções foi verificar se de fato havia a possibilidade de produzir diferentes jogos construcionistas a partir da LPS. Na tabela 3 é possível verificar as caracteristicas

dos jogos selecionados quanto ao problema apresentado, o artefato construído e o *feedback* apresentado no jogo.

Tabela 3 - Jogos construcionistas selecionados

Jogo	Problema	Artefato	*Feedback*
Sim Investigador (LESSA FILHO et al, 2015)	Representar uma história envolvendo conteúdos de disciplinas do ensino fundamental ou médio	Narrações/Casos/Histórias criadas pelo jogador	*Feedback* do avaliador que irá liberar ou não a história para os demais jogadores
Linguagem Logo (PAPERT; HAREL, 1991)	Criar figuras geométricas	Desenhos e formas geométricas	Desenho formado pelo comando inserido
RoboCode (ROBOCODE, 2015)	Criar um robô para batalhas	Robôs	Comportamento do Rôbo baseado nos comandos inseridos pelo usuário

Fonte: Elaborada pelo autor.

7.4 Produção e Avaliação do Estudo de Caso

Nesta seção será apresentado o desenvolvimento dos jogos utilizando a LPS JIndie e a análise realizada.

7.4.1 JIndie V.1 – PHP

Os jogos desenvolvidos nessa primeira avaliação foram o Sim Investigador, o jogo baseado na Linguagem Logo, e RoboCode. Após apresentar o jogo e a implementação realizada com a LPS JIndie, serão apresentadas as análises de cada jogo.

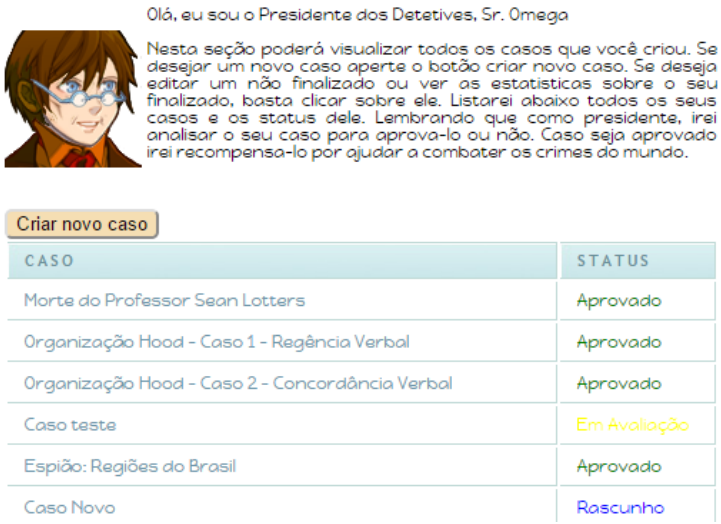
7.4.1.1 Sim Investigador

Sim Investigador é um jogo construcionista desenvolvido com o intuito de possibilitar que os estudantes tivessem uma ferramenta para criar suas próprias histórias envolvendo conteúdos disciplinares em forma de jogo (Lessa Filho et al, 2015).

O desenvolvimento do jogo original em 2012 contou com a participação do desenvolvedor da LPS presente nesse trabalho. Portanto, foi possível ter acesso aos códigos originais e informações relacionadas ao processo de desenvolvimento do jogo.

A LPS JIndie possibilitou a implementação do jogo Sim Investigador com as mesmas funcionalidades do jogo original e visualmente idêntico (Figura 35), devido ao acesso aos arquivos de interface originais disponíveis.

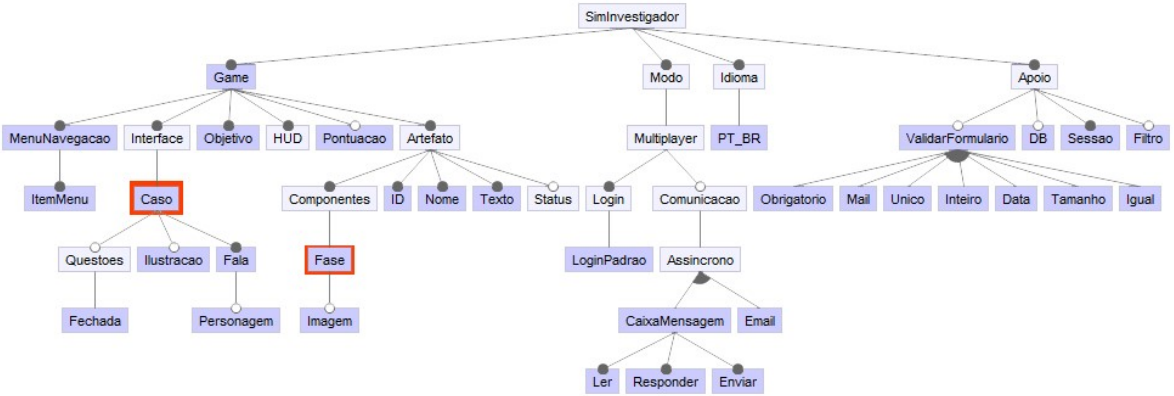
Figura 35 - Interface do Sim Investigador



Fonte: Elaborada pelo autor.

Ao realizar o *instance model* que representa os requisitos necessários para a implementação do jogo (Figura 36), foi possível aproveitar a maior parte das *features* apenas selecionando as variabilidades desejadas. Todavia, dois novos requisitos foram adicionados: “Caso” que representa a interface jogável do Sim Investigador e “Fase”, que representa o componente do artefato do jogo: narração/história criada:

Figura 36 Instance Model do Sim Investigador



Fonte: Elaborada pelo autor

O processo de implementação dos novos requisitos adicionados, não apresentaram dificuldades ou empecilhos como limitação de adicionar novos

recursos gráficos ou composição do artefato, devido à estrutura da interface gráfica e dos componentes dos artefatos, contarem com classes do tipo interface que permitem fácil troca e adição de novos componentes.

Em relação à interface gráfica “Caso” adicionada, a LPS ainda não disponibilizava nenhum componente pronto que unisse falas de personagens, com a possibilidade de adicionar ou não ilustração e questões fechadas. Porém já possuía componentes relacionados a questões e diálogos de personagens, simplificando o processo.

Quanto ao componente “Fase” do artefato do jogo, já era esperada a adição desse tipo de componente, uma vez que os componentes presentes no artefato de um jogo, dificilmente será igual aos componentes do artefato de outro jogo, por isso a LPS contava com uma classe do tipo interface (IComponent) para essa ação.

Durante a implementação do jogo foram necessárias realizar quatro alterações nas classes de apoio: validação de formulários; adição de uma classe responsável por gerar paginação; modificação na classe de banco de dados para listar conteúdos com *rank* (posição); e adição de uma classe de upload de arquivos. Tais melhorias foram adicionadas a versão final da LPS para que novos jogos pudessem contar com essas classes de apoio.

Para realizar a contagem de número de linhas foram ignoradas as linhas envolvendo a interface do jogo tal como html, css e javascript de forma que a avaliação pudesse ser centrada nas funcionalidades entregues pela LPS JIndie.

Tabela 4 - Número de Linhas de Código do Sim Investigador

Original	**Com a LPS**	
Total de Linhas	Total (Usadas)	Implementadas pelo Desenvolvedor
6.937	20.210 (10.936)	1849

Fonte: Elaborada pelo autor.

A versão original do jogo Sim Investigador contém 6.937 linhas de código enquanto a versão utilizando a LPS conta com 20.210. O fato da LPS contar com um valor maior de número de linhas de código está relacionado a 2 fatores: A LPS desenvolvida em PHP não conta com compilação condicional, desta forma vários componentes disponíveis que não são utilizados, também entram na contagem devido a não usar técnicas como a Compilação Condicional, que remove todos os recursos não usados do código final. Outro fato que pode ser considerado está

relacionado ao caso do código original não contar com funcionalidades extras como geração de *logs* e uso de *Uniform Resource Locator* (URL) amigáveis. Porém, mesmo com um valor maior de linhas de códigos total da aplicação, apenas 1.849 foram criadas pelo desenvolvedor representando um ganho de 73,3% menos linhas implementadas. Todas as funcionalidades restantes já haviam sido disponibilizadas pela LPS.

Na avaliação pela Complexidade Ciclomática que visa analisar a facilidade de manutenção do código, pode ser observado o ganho de desempenho na tabela 5:

Tabela 5 - Complexidade Ciclomática do Sim Investigador

Código	Original	Com a LPS
Cadastro de jogador	14	1
Jogar um caso	3	2
Responder uma pergunta do caso	7	5
Criar um caso	11	2
Adicionar uma pergunta ao caso em desenvolvimento	14	14
Investigar jogadores	8	1
Listar Rank com pontuação dos jogadores	9	4
Enviar mensagem	7	3
Ler mensagem	3	4
Aprovar um caso	3	3
Reprovar um caso	5	4

Fonte: Elaborada pelo autor.

Uso da LPS simplificou a estrutura do código de forma que o código responsável por cadastrar o jogo que apresentava Complexidade Ciclomática igual a 14 (Apêndice J), passou a valer apenas 1 ponto usando a LPS JIndie (Apêndice K).

Quanto à análise sobre o desenvolvimento do jogo através da LPS, por se tratar do primeiro jogo desenvolvido através da LPS, foi possível identificar pequenas falhas e melhorias que puderam ser adequadas à versão final da LPS para que os próximos jogos não apresentassem tais problemas. Em relação à vantagem observada em comparação ao código original, está na organização e facilidade de identificar os componentes, devido a usar o padrão *Model-View-Controller* (MVC); Também houve simplificação no desenvolvimento do código devido à existência de classes de apoio; Quanto ao artefato sendo construído no jogo, a manipulação das informações se tornou um processo fácil, uma vez que a

LPS disponibiliza uma fácil troca de informações e acesso ao artefato independe de página, assim como outros recursos do jogo (pontuação do jogador).

O desempenho de produção do jogo pode ser comparado ao tempo gasto no desenvolvimento do primeiro jogo. Sem considerar o tempo para definir como seria as regras de negócio do jogo definidos na versão original, foram utilizados uma média de 120h de produção, enquanto na versão com a LPS JIndie o tempo de produção foi de apenas 65h, apresentando um ganho em 45% do tempo gasto no desenvolvimento.

7.4.1.2 Logo

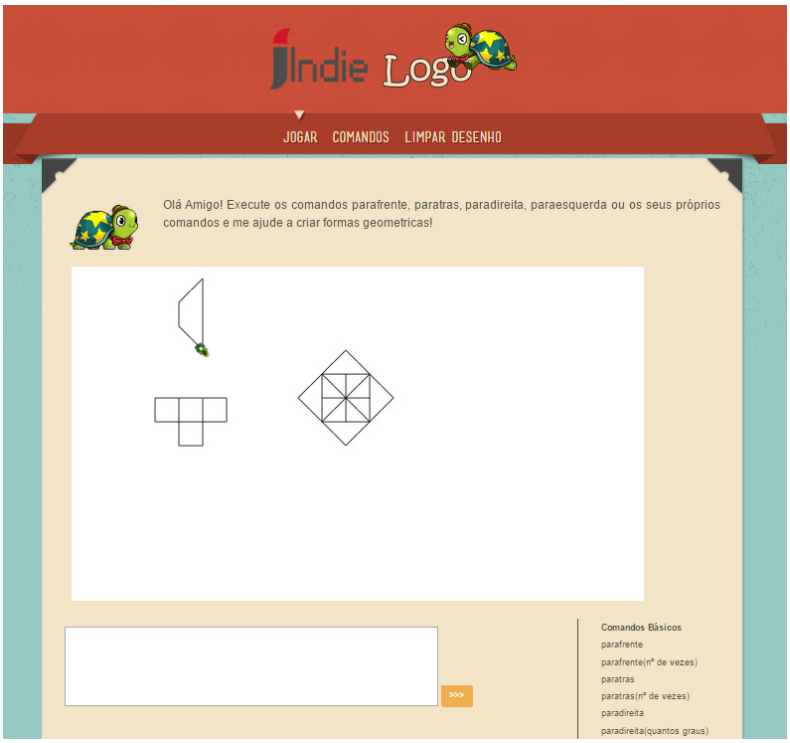
A Linguagem Logo é uma linguagem voltada à produção de figuras geométricas, no qual o usuário poderá controlar uma tartaruga enquanto passa as direções através de linhas de comandos (PAPERT; HAREL, 1991). Durante a partida, o jogador poderá aprender sobre dimensões e estrutura de diferentes figuras geométricas, como também aprender lógica de programação ao trabalhar os comandos disponíveis no jogo. Durante o processo, o jogador estará construindo um desenho como artefato, que é composto por vários códigos.

A versão do jogo Logo desenvolvida através da LPS JIndie (Figura 37) apresenta os seguintes comandos:

- parafrente(passos) – Move a tartaruga para frente X passos;
- paratras(passos) – Move a tartaruga para trás X passos;
- paraesquerda(graus) – Vira a tartaruga em X graus para esquerda;
- paradireita(graus) – Vira a tartaruga em X graus para direita;
- usenada – Desabilita marcação na imagem enquanto move a tartaruga;
- uselapis - Habilita marcação na imagem enquanto move a tartaruga;

Além dos comandos acima, o jogo desenvolvido também permite a criação de comandos personalizados semelhante ao que ocorre com a Linguagem Logo.

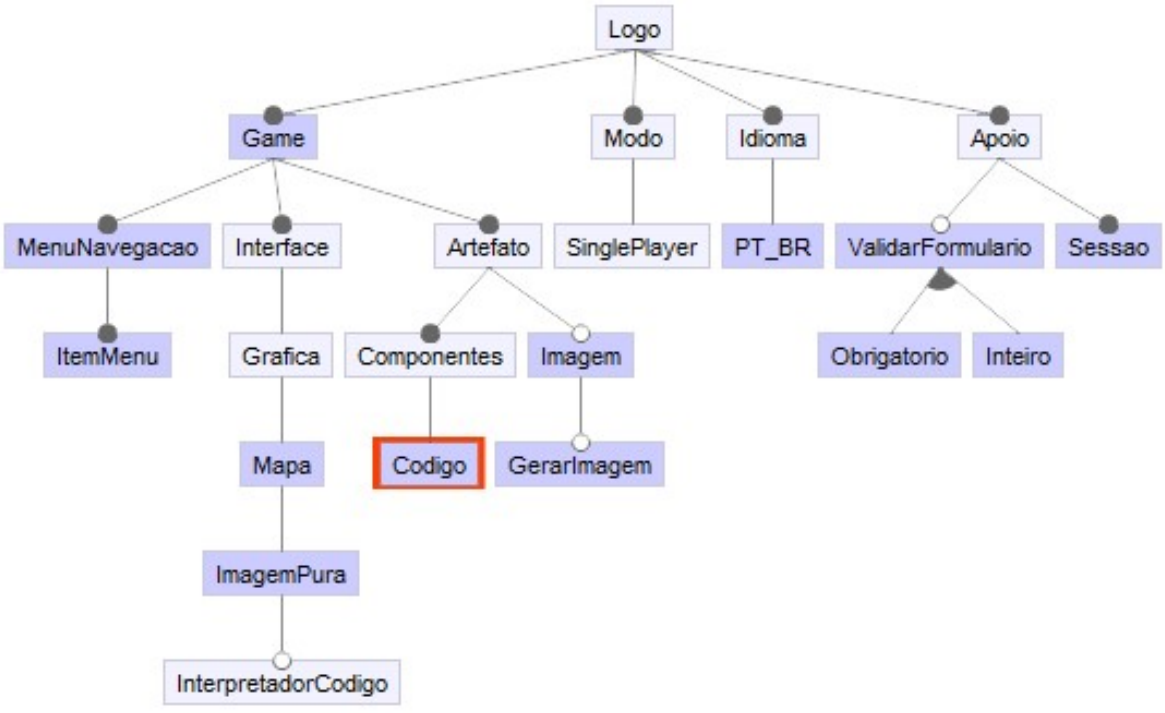
Figura 37 – Interface do jogo Logo



Fonte: Elaborada pelo autor.

Na Engenharia da Aplicação, foi possível realizar o desenvolvimento do *Instance Model* com os requisitos da aplicação como pode ser observado na figura 38.

Figura 38 - *Instance Model* do jogo Logo



Fonte: Elaborada pelo autor

Quase toda estrutura do *instance model* pode ser reaproveitada da Engenharia de Domínio da LPS JIndie. Semelhante ao jogo Sim Investigador, também foi preciso adicionar uma nova *feature* nomeada de código, para representar os componentes que formam o artefato, desenho. A *feature* adicionada representa os códigos inseridos pelo jogador que formarão o desenho desejado. Essa nova *feature* pode ser facilmente adicionada a implementação através do uso de uma interface disponibilizada pela própria LPS.

Porém como pode ser observado na Figura 38, caraterísticas tidas antigamente como mandatórias no artefato como ID e Nome, não são utilizadas neste jogo, uma vez que o desenho gerado não é salvo no banco de dados, ficando apenas disponível durante a sessão da partida jogada.

No desenvolvimento do jogo não foi preciso refatorar nenhum código existente, porém um componente foi preciso ser criado: a classe responsável pelas regras navegação da tartaruga (Comandos disponíveis) no desenho. A adição dessa classe foi integrada com os demais códigos através da classe interface ICode, que faz a relação entre o interpretador de códigos e as regras de comandos disponíveis no código que o usuário pode executar. Devido a classe CodeReader (interpretador de código), disponibilizada pela LPS JIndie, permitir ler e interpretar o código inserido pelo usuário utilizando como base as regras implementadas na interface ICode e a classe responsável por gerar imagens (MapGenerator), foi possível trabalhar o código inserido pelo usuário transformando-os em formas de desenhos sem complexidades, pois o trabalho complexo de interpretar comandos e gerar imagens já é reaizado por esses recursos que a LPS JIndie disponibiliza.

A classe da LPS JIndie responsável por gerar a interface gráfica jogável, SceneMap, que disponibiliza a imagem e o campo onde o usuário digita os comandos que a tartaruga deve executar, foi preciso ser herdado para uma nova classe, para realizar pequenas alterações como adicionar a direção (rotação) que a tartaruga deve seguir e adicionar as linhas geradas na imagem quando a tartaruga caminha.

Foi possível automatizar quase toda produção do jogo, de modo que em relação ao número de linhas do código, apenas cerca de 2% de todas as linhas de código no jogo foram implementadas no desenvolvimento do jogo. Embora a versão da aplicação contasse com 18.886 linhas de código, apenas 4.096 de fato são utilizadas no jogo, sendo as demais funcionalidades de outras *features* não

utilizadas no jogo. Embora haja essa diferença no número de linhas totais e usadas, não há grande impacto no desempenho, uma vez cada ação realizada no jogo, não executará todo o código, apenas os códigos chamados durante o processo.

Tabela 6 - Número de linhas de código do jogo Logo

Linhas de Código da Logo	
Total (Usadas)	Implementados no produto final
18.886 (4096)	409

Fonte: Elaborada pelo autor.

As principais funcionalidades do jogo Logo são a possibilidade de criação de comandos personalizados (Por exemplo: quadrado ou triangulo) além dos já disponíveis: parafrente, paraesquerda, paradireita, paratras, uselapis, usenada; executar os comandos; excluir os comandos criados; mover a tartaruga no desenho; e limpar o desenho criado. Ao analisar a Complexidade Ciclomática dos cinco principais comandos do jogo, é possível observar uma complexidade muito baixa:

Tabela 7 – Complexidade do jogo Logo

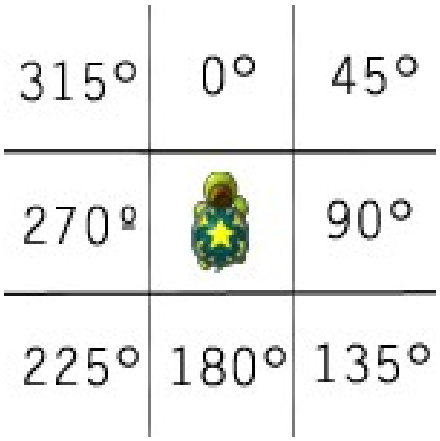
Código	CC
Criar comandos próprios	2
Excluir comando criado	1
Executar comandos	4
Limpar desenho	1
Mover tartaruga	4

Fonte: Elaborada pelo autor.

Os códigos com CC abaixo de 5, são códigos de baixo risco e fácil manutenção (CLARK et al., 2008).

A implementação do jogo realizada através do uso da LPS JIndie tornou-se mais simples, devido as funcionalidades de interpretação de código e geração de imagens já disponibilizados pela LPS, de forma que o desenvolvimento do jogo foi possível ser concretizado em apenas 16h de produção. Por outro lado, o script responsável por gerar o desenho funciona por blocos (*tiles*), semelhante a uma matriz, o que limita a rotação da tartaruga em apenas oito direções:

Figura 39 - Rotação em graus da tartaruga no jogo Logo



Fonte: Elaborada pelo autor.

Para contornar a situação da limitação de rotação da tartaruga em apenas 8 direções e possibilitar desenhos mais arredondados, poderia ser optado como uma solução diminuir o tamanho de pixels padrão da imagem de cada bloco de 32 pixels para 5 pixels, possibilitando desenhos mais curvados semelhante ao que acontece em desenhos baseados em *pixel art*.

7.4.1.3 RoboCode

O jogo RoboCode é um jogo que permite que os jogadores batalhem entre si criando seus próprios robôs, através da plataforma Java ou .NET (ROBOCODE, 2015). Neste jogo o artefato criado pelo jogador será o próprio robô, e seus componentes serão os códigos definidos em cada método.

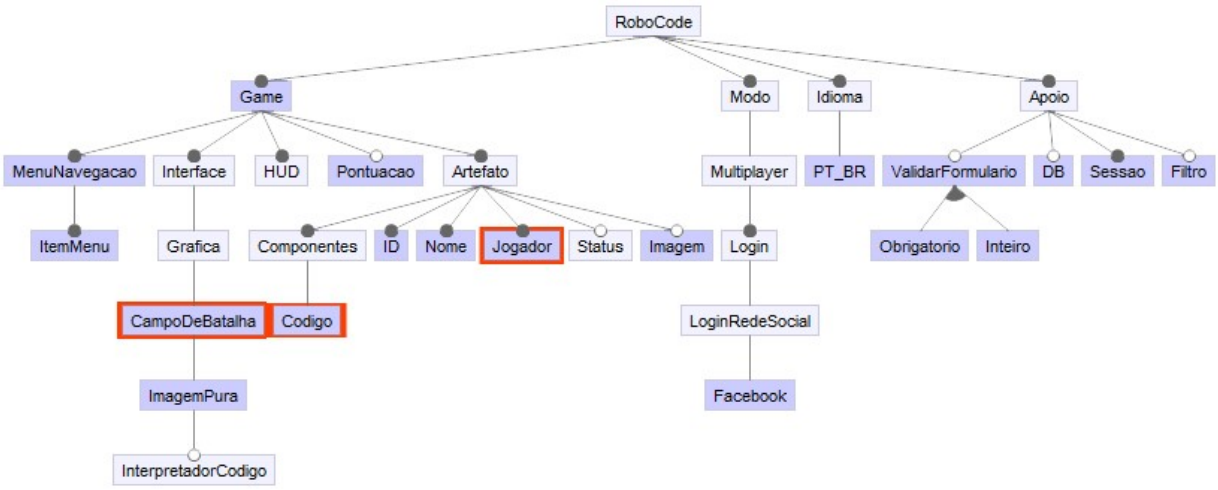
No jogo RoboCode existe basicamente 4 métodos:

- run – Método padrão que é realizado a todo momento enquanto o robô existir;
- onScannedRobot – Método executado quando um robô avista outro;
- onHitByBullet – Método executado quando o robô é acertado por um tiro;
- onHitWall – Método executado quando o robô acerta as bordas limites do cenário;

Além dos métodos, o jogo também conta com comandos estabelecidos em relação ao robô: tiro; movimentação; rotação do robô; e rotação do canhão do rôbo.

Diante dessas informações com o uso da LPS JIndie foi estabelecida a Engenharia de Requisitos da Aplicação através do *Instance Model* (Figura 40).

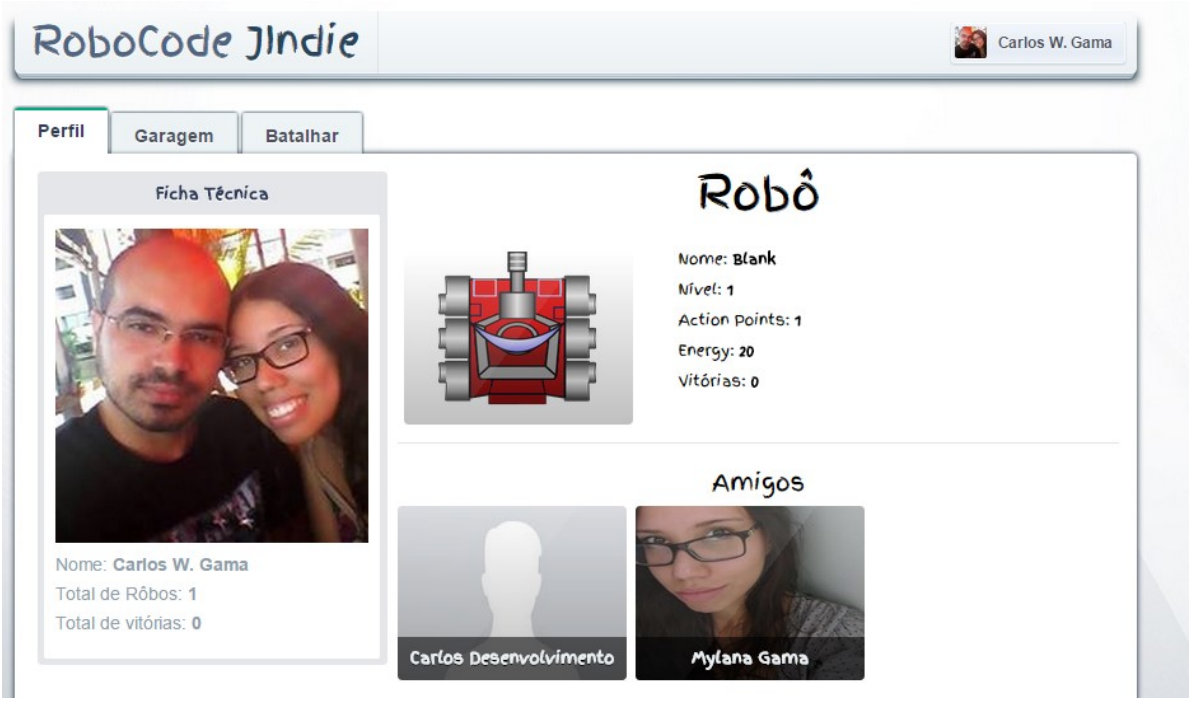
Figura 40 - *Instance Model* do RoboCode



Fonte: Elaborada pelo autor.

No jogo original, embora incentive as batalhas entre jogadores, estas ocorrem localmente inserindo o código de cada robô montado na mesma maquina. Para facilitar a batalha entre jogadores multiplayer, a versão com o JIndie integra o *login* da rede social Facebook, possibilitando que amigos possam batalhar entre si, sem a necessidade de estarem no mesmo ambiente (Figura 41).

Figura 41 - Perfil do jogador com listagem de amigos



Fonte: Elaborada pelo autor

No *Instance model* (Figura 41) do jogo RoboCode podemos observar três diferentes características adicionadas. Na parte de *layout* (Interface) foi adicionado um novo tipo de interface gráfica: CampoDeBatalha (No código implementado, recebeu o nome de *SceneBattle*) que foi implementado através da classe interface IScene para facilitar a integração com o restante do jogo. Essa classe recebe os artefatos (robôs criados) que participarão da batalha e faz todo o gerenciamento da batalha (Figura 42) em si, como pontuação, atualização do HUD (Informações dos robôs como energia e pontos de vida), imagens e interpretação dos códigos dos usuários.

Figura 42 - Batalha no RoboCode criado através do JIndie

Fonte: Elaborada pelo autor.

No artefato do jogo é adicionada a *feature* Códigos, que são os componentes que formam o artefato Robô. Também foi adicionada ao artefato uma *feature* que indica a qual jogador o Robô pertence.

No desenvolvimento da aplicação também foi necessário criar uma classe que gerenciasse as regras de navegação que cada robô pode executar como tiro, movimentação e rotação. A adição dessa classe com as regras foi realizada através

do uso de interface ICode, que a classe de interpretação de código do usuário utiliza para reconhecer os comandos disponíveis a serem executados.

Ao analisar o número de linhas implementadas pelo desenvolvedor tivemos o seguinte resultado:

Tabela 8 - Número de Linhas de Código do jogo RoboCode

Número de Linhas de código com a LPS	
Total (Usadas)	Implementadas pelo Desenvolvedor
19641 (7890)	1194

Fonte: Elaborada pelo autor.

Ao levar em consideração as funcionalidades como interpretação de códigos inseridos pelo jogador, ilustração da imagem do campo de batalha e acesso a rede social, o trabalho que o desenvolvedor teve no desenvolvimento do jogo engloba um número pequeno de linhas, sendo as maiores classes: “Commands” (426 linhas) que define quais são os comandos que os robôs podem realizar e o que cada comando faz; E a classe “SceneBattle” (411 linhas) que gerencia todas as ações realizadas durante a batalha entre os robôs.

Na análise da Complexidade Ciclomática, foi observado em alguns pontos que a LPS ainda pode precisar de melhorias que tornem o código mais simples para o desenvolvedor, como é o caso do método responsável por executar os ataques dos robôs, no qual a Complexidade Ciclomática chegou ao limite de 20 pontos:

Tabela 9 - Complexidade Ciclomática do jogo RoboCode

Código	CC
Realizar Login/Cadastro	2
Criar Robô	1
Editar Ações do Robô	6
Iniciar batalha	3
Executar Ação da Batalha	11
Executar Movimento	8
Executar Ataque	20
Girar Robô	10
Girar Canhão	10
Finalizar batalha	2

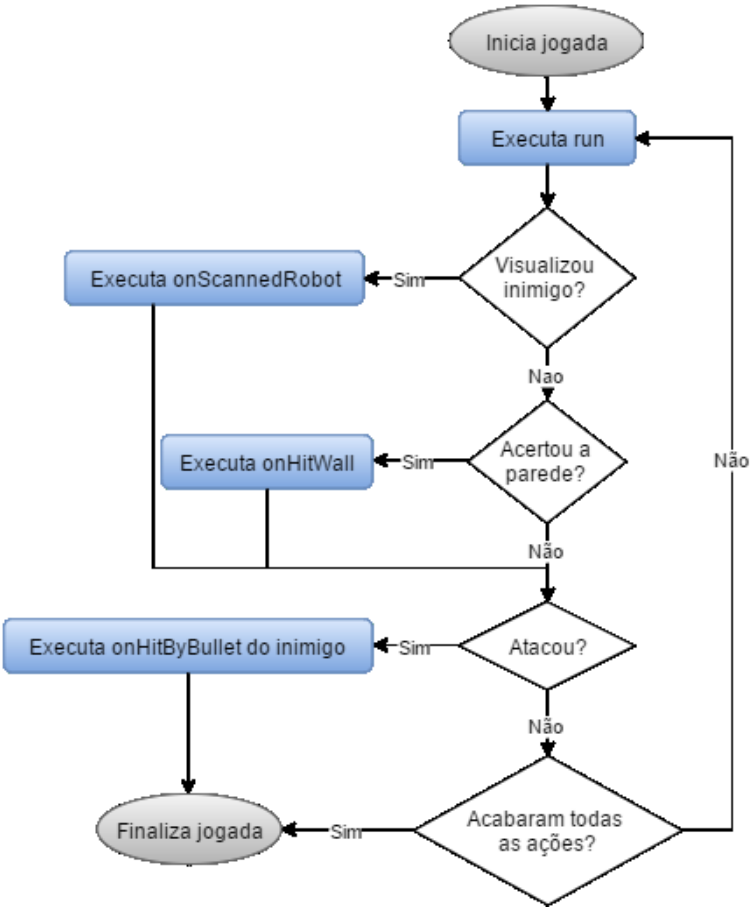
Fonte: Elaborada pelo autor.

O método responsável por executar o ataque é composto por diversas condições como verificação de vida disponível, colisões entre ataques de rôbos, e direção do ataque. Desta forma a LPS em trabalhos futuros pode apresentar uma

classe de apoio para tratamento de decisões. Entretanto, mesmo apresentando um valor alto de CC no método do ataque, ainda está dentro do limite recomendado, e a média da CC no jogo está em 7,4 pontos, considerada um valor agradável.

Quanto ao processo de desenvolvimento foi possível observar três pontos como fatores negativos. O primeiro ponto é semelhante ao jogo Logo, quanto à limitação da rotação dos tanques em apenas oito direções; outro fato encontrado foi no fato de que o JIndie necessita realizar requisições para o servidor para atualizar as informações da batalha, ao contrário do que ocorre no jogo original onde as informações fluem continuamente (tempo real). Desta forma para que o jogo fosse desenvolvido foi necessário alterar o sistema de tempo real para um sistema baseado em turnos, comum em jogos táticos, onde apenas um robô é executado por vez. As ações de cada robô por turno seguem o fluxo padrão do jogo original, conforme a Figura 43 demonstra.

Figura 43 - Fluxo das ações dos robôs



Fonte: Elaborada pelo autor.

O terceiro fator negativo observado está no fato que a LPS embora disponibilize um interpretador de comandos, que o próprio desenvolvedor pode personalizar com facilidade, este recurso não trabalha com compilação de código, portanto se torna difícil ou inviável trabalhar com classes inteiras, sendo mais recomendados trabalhar com pedaços de códigos divididos pelos quatro métodos presentes na construção do robô do jogo.

Em contrapartida, apesar das limitações relacionadas à compilação de código e rotação nas imagens, a LPS possibilitou o desenvolvimento do jogo em apenas 46h, das quais apenas 16 foram utilizadas na implementação de códigos, sendo as demais horas utilizadas na produção de layout e imagens utilizadas no jogo.

Outra vantagem que pode ser observada está presente na facilidade em tornar o jogo *multiplayer* sem a necessidade dos jogadores estarem no mesmo ambiente, podendo realizar as batalhes de suas próprias casas.

7.4.1.4 Conclusão

Nesta subseção será tratada a conclusão dos desenvolvimentos dos três jogos utilizando a LPS JIndie baseada em PHP.

A LPS desenvolvida se mostrou capaz de desenvolver diferentes tipos de jogos construcionistas reduzindo em grande parte o trabalho do desenvolvedor e tempo de desenvolvimento, devido aos componentes reusáveis e definição da arquitetura disponibilizada pela LPS.

As escolhas das técnicas de implementação da variabilidade permitiram o desenvolvimento dos jogos sem problemas, porém pode ser observar através do número de linhas de código que muitas linhas e classes não chegam a ser utilizadas, podendo a compilação condicional ser uma escolha mais adequada para reduzir essas linhas e arquivos que não são utilizados. A vantagem em relação ao número de linhas que não são utilizadas, é que a versão em PHP não é compilada e sim interpretada, de forma que apenas os códigos utilizados durante a requisição da página são de fatos utilizados, ao contrário da versão compilada onde é gerado um único arquivo com todos os dados.

Quanto à complexidade do código é possível observar resultados satisfatórios como pode ser visto na Tabela 10, que demonstra o número de ocorrências de cada CC dividida por grupos:

Tabela 10 - Complexidade Ciclomática da LPS JIndie em PHP

CC	Ocorrência
1-5	19
6-10	4
11-15	2
16-20	1
>=21	0
Média 4,96	

Fonte: Elaborada pelo autor.

Segundo estudo realizado pelo Instituto de Engenharia de Software (BRAY et al, 1997) um programa pode ser classificado como de alto ou baixo risco através da sua Complexidade Ciclomática:

Tabela 11 - Avaliação dos Riscos de um programa

CC	Avaliação do Risco
1-10	Um programa simples, sem muito risco.
11-20	Mais complexo, risco moderado.
21-50	Programa complexo, alto risco.
>50	Programa intestável (Risco muito elevado)

Fonte: Elaborada pelo autor.

O uso da LPS também permitiu que o desenvolvedor se focasse no núcleo do jogo, contando com as classes de apoio para realizar as tarefas secundárias.

Em atividades futuras na LPS poderia dar-se um foco em animações e permitir realizar atividades em tempo real.

7.4.2 JIndie V.2 – Java

A segunda implementação da plataforma utilizada no JIndie, foi realizada em Java utilizando a ferramenta de apoio a construção de LPS, CIDE. O CIDE é uma ferramenta especializada na implementação da variabilidade baseada na Compilação Condicional, utilizando um editor personalizado que permite a seleção dos códigos através de marcadores baseado nas cores dos requisitos do *feature model* (FEIGENSPAN et al., 2010).

A subseção abaixo mostrará os resultados obtidos durante a produção do jogo RoboCode comparando a primeira versão em PHP com a segunda versão em Java.

7.4.2.1 RoboCode

Durante as primeiras etapas da LPS JIndie envolvendo a Engenharia da Aplicação e o Projeto da Aplicação, não houve modificações a serem realizadas, uma vez que apenas o framework disponibilizado foi alterado para linguagem Java e adicionada a implementação da variabilidade por Compilação Condicional com o CIDE. De tal modo que houvessem apenas alterações relacionadas a implementação do código. Nesta versão em Java, por ter sido desenvolvida apenas com o foco em realizar a análise do uso da compilação condicional e do CIDE, alguns recursos secundários que não tinham relação com o domínio, não chegaram a ser implementados por completo, como é o caso de registros de *logs* de acesso pela plataforma.

Ao analisar a quantidade de linhas de código na aplicação utilizando a compilação condicional, pode-se observar que a LPS diminui a quantidade de linhas de código de forma a deixar apenas as linhas que realmente são utilizadas.

Tabela 12 - Número de Linhas de Código do jogo RoboCode com a com compilação condicional

LPS		**RoboCode**	
Total	Após Gerar	Total	Implementados pelo Desenvolvedor
3803	2512	3946	1457

Fonte: Eaborada pelo autor.

Na versão em PHP a plataforma pura sem nenhum jogo possui um total de 18.482 linhas de código, enquanto a versão em Java antes de remover as linhas através da compilação condicional possui apenas um total de 3803 linhas de código. Isso ocorre devido ao fato da versão em Java utilizar bibliotecas externas em pacotes fechados (.jar) que não entraram na contagem de linhas de códigos. Entre os pacotes externos é possível descontar as linhas das SDKs das redes sociais, biblioteca de envio de email, conexão com banco de dados através do Hibernate entre outras bibliotecas, ou algumas funcionalidades secundárias que não há relação com o domínio como o *log*.

Entretanto, mesmo com a diferença do número de linhas de código entre as versões é possível observar que após gerar a versão do código que será utilizada na implementação do jogo, houve uma redução de 33,94% do total de linhas da plataforma utilizada, com o uso da compilação condicional. Após a implementação

do jogo, houve um total de 3.946 linhas em toda a aplicação do qual apenas 1.457 linhas, 36,92% foram implementadas pelo desenvolvedor. Através desses valores é possível observar que a compilação condicional é capaz de reduzir o número de linhas de aplicação, reduzindo consequentemente o tamanho do arquivo compilado e as verificações de códigos realizadas que não há necessidade, todavia não reduz o trabalho do desenvolvedor na etapa de implementação dos componentes exclusivos de seu jogo.

Ao analisar a Complexidade Ciclomática, o uso da compilação condicional não apresentou um impacto muito favorável:

Tabela 13 - Complexidade Ciclomática do RoboCode o na LPS JIndie em Java

Código	CC com Compilação Codicional	CC Sem compilação condicional
Realizar Login/Cadastro	3	2
Criar Robô	1	1
Editar Ações do Robô	6	6
Iniciar batalha	3	3
Executar Ação da Batalha	12	11
Executar Movimento	9	8
Executar Ataque	20	20
Girar Robô	10	10
Girar Canhão	10	10
Finalizar batalha	2	2

Fonte: Elaborada pelo autor.

Os valores foram bastante semelhantes em ambas as versões, ocorrendo maiores variações relacionadas a diferença entre as linguagens de programação. Neste ponto esperava-se observar melhorias na complexidade, ao remover estruturas de decisões que não são utilizadas ao longo do código, contudo a análise realizada é baseada nas atividades relacionadas ao desenvolvimento dos conteúdos no qual o desenvolvedor tem contato, e a diminuição da Complexidade Ciclomática ocorre justamente nos componentes prontos já disponibilizados pela LPS, ao gerar a arquitetura que será utilizada no jogo, antes do desenvolvedor iniciar a produção da aplicação final. Já ao analisar a redução da Complexidade Ciclomática nos componentes disponibilizados e configurados na própria plataforma da LPS, é possível observar uma redução significativa no componente de apoio responsável pela validação de formulário, no qual a complexidade sai de 19 pontos na versão sem compilação condicional para apenas 5 na versão com a compilação condicional.

7.4.2.2 Conclusão

O uso da compilação condicional na plataforma disponibilizada pela LPS JIndie se mostrou eficiente ao reduzir consideravelmente o número de linhas de código e arquivos, tornando a estrutura da aplicação final reduzida para apenas o essencial, todavia o trabalho realizado e a complexidade dos códigos no desenvolvimento dos jogos não apresentaram melhorias significativas, uma vez que a compilação condicional ocorre justamente nos componentes disponibilizados para uso, antes do contato do desenvolvedor. Ainda assim, o seu uso pode resultar em melhorias no processamento, uma vez que os componentes do próprio framework não precisarão fazer verificações extras desnecessárias para o jogo.

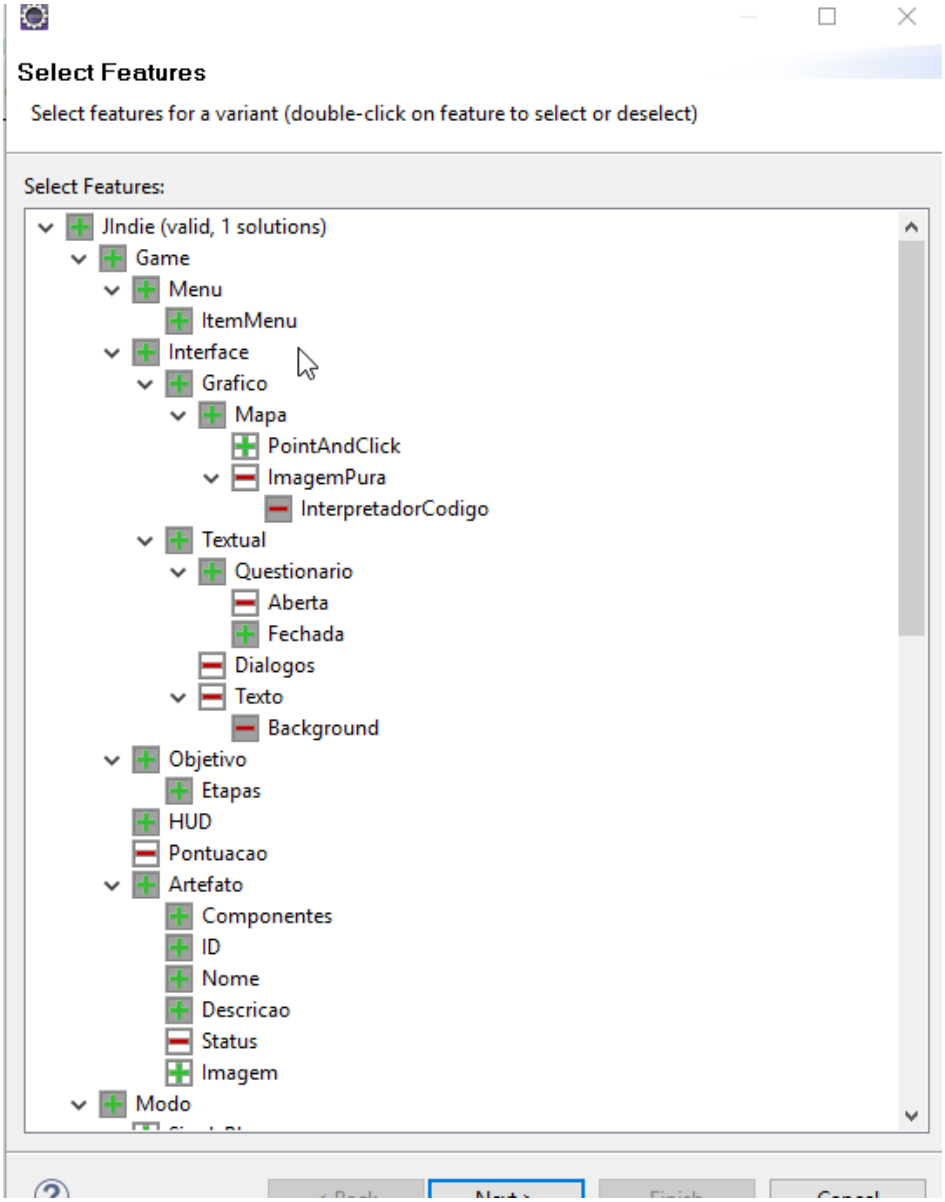
Entretanto a Compilação Condicional para o desenvolvimento da LPS, pode tornar o processamento de manutenção mais trabalhoso e confuso, uma vez que no mesmo script como no exemplo da biblioteca de idiomas, terá diversas linhas de código para alternativas diferentes da variabilidade.

Em relação à ferramenta CIDE, esta se demonstrou de fato ser uma ferramenta com um conceito muito interessante. É comum que ao usar a técnica de compilação condicional, o desenvolvedor necessite aprender sobre as *tags* e anotações que deve utilizar para indicar quais campos devem permanecer ou serem removidos após compilação, além de adicionar mais linhas no código. Porém, o CIDE permitiu que esse processo fosse realizado sem a necessidade de ter o conhecimento das *tags* e regras utilizadas na compilação condicional, apenas selecionando as linhas de código e informando a qual requisito tais linhas pertenciam. Além das linhas no código, o CIDE também permitiu a seleção de arquivos inteiros, de forma a retirar todos os arquivos que não forem necessários na aplicação final.

Caso uma LPS desenvolvida utilizando o CIDE gere sempre softwares padrões que não necessitem da intervenção do desenvolvedor, o CIDE irá possibilitar a construção do produto final realizando todo o trabalho apenas selecionando dos requisitos da aplicação final como na Figura 44. Entretanto para o desenvolvimento de uma LPS com uma quantidade considerável de variabilidade, o desenvolvedor além de ter os problemas citados em relação à Compilação Condicional, poderá ter problemas relacionado a cores com pequenas diferenças de

tonalidades, o que ocorreu quando a LPS ultrapassou mais de 20 pontos de variabilidade.

Figura 44 - Seleção de Requisitos da LPS JIndie na ferramenta CIDE



Fonte: Elaborada pelo autor.

Por outro lado, mesmo que a ferramenta apresente uma proposta interessante e que facilite o processo do uso da compilação condicional, ainda houve alguns problemas relacionados ao seu uso.

Uma das dificuldades encontradas no uso do CIDE ocorreu com arquivos do tipo XML. Após o processo de gerar a aplicação com apenas os requisitos selecionados, todos os XML's presentes na aplicação perderam sua formatação, recebendo o erro de má formatação (Figura 45).

Figura 45 - Erro na formatação dos XML's

Antes

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/x

<persistence-unit name="tabela1">
    <provider>org.hibernate.ejb.Hibernate
    <class>br.com.jindie.app.beans.Tabela

    <properties>
        <property name="javax.persistence
        <property name="javax.persistence
        <property name="javax.persistence
        <property name="javax.persistence

        <property name="hibernate.dialect
        <property name="hibernate.show_sq
        <property name="hibernate.format_
        <property name="hibernate.hbm2ddl

    </properties>
</persistence-unit>
</persistence>
```

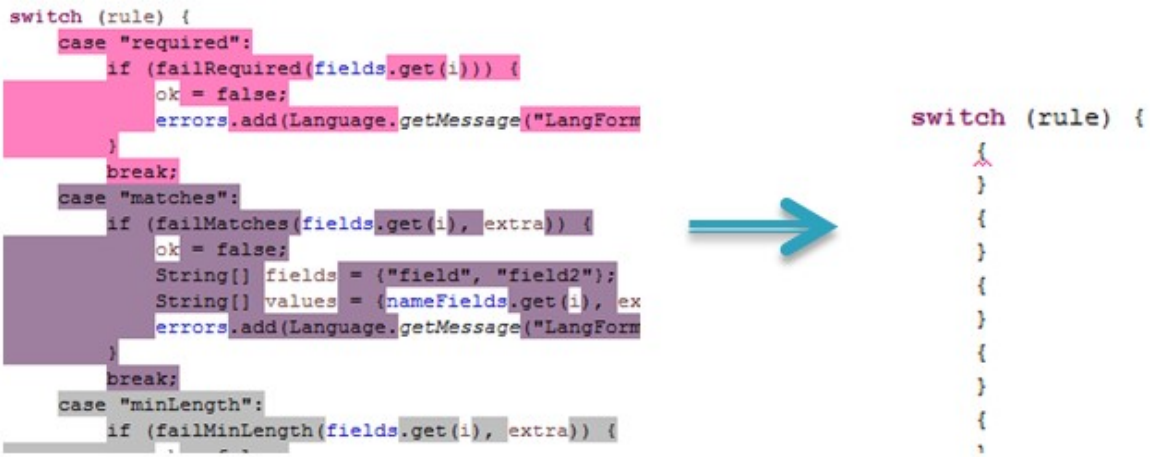
Depois

```
<?xml  version = "1.0"  encoding = "UTF-8" ?>
< persistence  xmlns = "http://java.sun.com/xml/

< persistence-unit  name = "tabela1" >
    < provider > org.hibernate.ejb.HibernatePers
    < class > br.com.jindie.app.beans.Tabela1 </

    < properties >
        < property  name = "javax.persistence.jd
        < property  name = "javax.persistence.jd
        < property  name = "javax.persistence.jd
        < property  name = "javax.persistence.jd

        < property  name = "hibernate.dialect"
        < property  name = "hibernate.show_sql"
        < property  name = "hibernate.format_sql
        < property  name = "hibernate.hbm2ddl.au

    </ properties >
</ persistence-unit >
</ persistence >
```

Fonte: Elaborada pelo autor.

Outro erro encontrado ao utilizar a ferramenta CIDE incidiu na própria funcionalidade de compilação condicional, onde alguns trechos dos códigos não foram removidos de forma corretamente, mantendo ainda as chaves que deveriam ser removidas junto com o resto do código:

Figura 46 - Falha na remoção de trechos de códigos



Fonte: Elaborada pelo autor

Como uma ferramenta voltada a pesquisa e o ensino de LPS, o CIDE se demonstra uma ferramenta promissora, porém ao analisar o uso do CIDE como uma ferramenta para utilizar comercialmente, existem algumas melhorias a serem realizadas, além do risco no alto nível de dependência na ferramenta, uma vez que

por não utilizar as anotações padrões da compilação condicional e sim um recurso exclusivo da ferramenta, não há como realizar a interoperabilidade entre diferentes ferramentas.

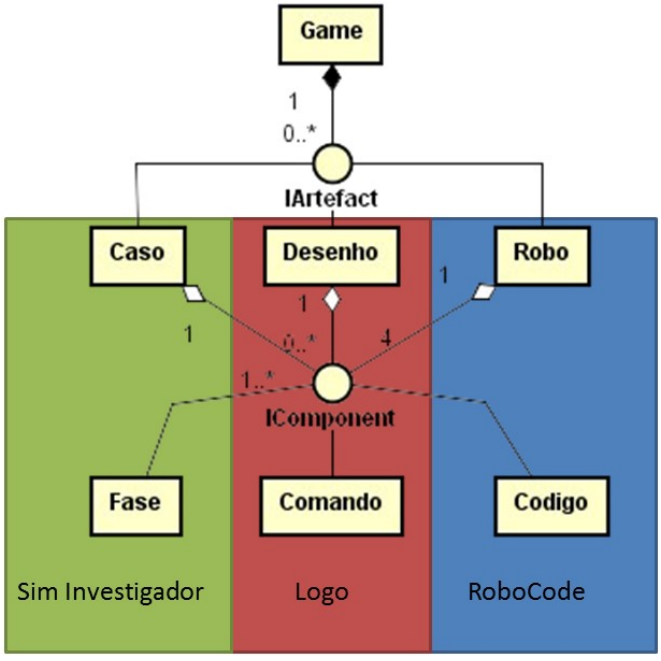
7.4.3 Conclusão da LPS *JIndie*

Nesta seção serão apresentados os resultados observados durante o Estudo de Caso apresentado neste capítulo.

Foi possível observar que a LPS proposta nesse trabalho consegue realizar o desenvolvimento de diferentes tipos de jogos no contexto do construcionismo. Durante o desenvolvimento dos jogos através do uso da LPS, foi possível se focar totalmente nas características exclusivas do jogo e nas principais atividades, sem perder tempo com atividades secundárias, devido às classes de apoio, reduzindo assim consideravelmente o tempo usado no desenvolvimento.

As classes relacionadas ao artefato e aos componentes do artefato puderam ser trabalhadas independentes do contexto do enredo do jogo através do uso de interfaces e componentes que permitiam a reusabilidade de códigos, como pode ser visto na Figura 47:

Figura 47 - Diagrama de Classe dos artefatos e componentes dos jogos desenvolvidos



Fonte: Elaborada pelo autor.

A LPS JIndie também permitiu a automatização de muitas funcionalidades. No caso do jogo Logo, a produção foi quase toda automatizada, restando apenas ao desenvolvedor a definir as regras de navegação da tartaruga e a produção do layout do jogo unidos aos recursos de interface fornecidos pela LPS.

Durante este processo também foram observados melhorias e correções de falhas que foram adicionadas a LPS, para que novos jogos possam usufruir dessas atualizações.

A plataforma comum disponibilizada ainda conta com algumas limitações relacionadas ao uso de animações e conteúdos executados em tempo real que podem ser trabalhadas em projetos futuros. Apesar disto, a finalização dos jogos não foi comprometida e pode-se observar vantagens no desenvolvimento do jogo através de recursos disponibilizados na LPS envolvendo o domínio do jogo.

7.4.3.1 Ameaças a Validade

Além das análises realizadas ainda há fatores de ameaça à validade. Entre as ameaças é possível citar: (i) O fato do código do Sim Investigador ser reescrito pode já contribuir com o aprimoramento da segunda versão do jogo; (ii) A experiência do programador no desenvolvimento da primeira versão do jogo Sim Investigado ter sido aprimorada desde a produção da segunda versão; (iii) Os jogos produzidos serem de tamanhos pequenos e médios; (iv) jogos desenvolvidos para plataformas diferentes; (v) Avaliação do tempo pode não ser precisa; (vi) A redução da Complexidade Ciclomática pode ser coincidência.

Entre o desenvolvimento da primeira versão do jogo Sim Investigador e a segunda versão foram realizados novos estudos e análises sobre o jogo, além de sugestões de melhorias. Logo, ciente de novas técnicas, o fato de apenas reescrever o código utilizando diferentes técnicas e o conhecimento adquirido na produção da primeira versão, já pode influenciar na qualidade da ferramenta e tempo de desenvolvimento do código. Ainda assim, sem o uso da LPS o programador haveria de gastar um tempo maior comparado ao jogo produzido através da LPS, devido ao fato de ter que criar alguns componentes disponibilizados e utilizados pela LPS na produção do jogo, além de definição de requisitos e arquitetura do projeto.

Quanto a experiência do programador na produção do jogo Sim Investigador, houve um intervalo de três anos entre a versão original e a com a LPS JIndie, no

qual o programador continuou estudando e aprimorando seu conhecimento, fazendo com que sua experiência fosse diferente nos momentos de produção do jogo, o que pode influenciar a avaliação da complexidade do código. Todavia ao observar os resultados obtidos nos três jogos relacionados a Complexidade Ciclomática, ainda é possível observar que o uso da LPS mantém um valor baixo e aceitável na sua complexidade, facilitando a manutenção.

A ameaça relacionada ao tamanho dos jogos envolve o fato de que nenhum jogo de porte grande foi desenvolvido, contando com a participação de várias pessoas na produção e em uma grande escala de tempo, como ocorre com jogos de grande porte, demoram meses e anos para serem finalizados. Portanto, não há como saber com precisão de como a produção do jogo irá se comportar em jogos desta dimensão. Esta é uma ameaça que pode ser avaliada em trabalhos futuros com maior disponibilidade de tempo e recursos, porém acredita-se através dos estudos realizados, que o uso da LPS por contar com bastantes componentes reutilizáveis em diferentes artefatos, permita um ganho no desempenho e organização na produção de jogos desse porte.

Alguns dos jogos construcionistas listados estão presentes em plataformas como *desktop* e *mobile*, porém, as versões das plataformas comuns desenvolvidas na LPS JIndie são para jogos em browser. Apesar dessa limitação, na produção dos jogos desenvolvidos, foi possível observar a migração dos jogos Logo e RoboCode da plataforma *desktop* para a *web.* O desenvolvimento de um sistema a partir de linguagem web também pode ser utilizado para produção de jogos em smartphone com uso de técnicas de responsividade, tornando as telas dos jogos responsivas para diferentes tamanhos.

A análise do tempo investido na construção de um software costuma ser uma atividade muito complexa, uma vez que pequenas distrações como receber e-mails ou até mesmo beber água podem influenciar na contagem final.

Quando a avaliação da redução da Complexidade Ciclomática do jogo Sim Investigador, também pode representar uma ameaça uma vez que a redução destes valores pode ser uma coincidência para essa situação, sendo recomendado realizar novos testes com outros jogos para uma confirmação mais real.

No próximo capítulo apresentam-se as considerações finais deste trabalho.

8 CONCLUSÃO

Nesta seção serão discutidas as considerações finais do trabalho desenvolvido e os possíveis trabalhos futuros.

8.1 Considerações Finais

O trabalho apresentado nesta dissertação teve por objetivo a especificação e implementação de uma Linha de Produto de Software para jogos construcionistas, contando com o desenvolvimento de quatro jogos gerados a partir dela.

O uso de jogos na educação está começando a se tornar algo comum, porém os desenvolvedores ainda enfrentam vários desafios no desenvolvimento de um jogo. A LPS JIndie busca facilitar o processo de desenvolvimento, uma vez que o programador apenas precisará se preocupar com o domínio, o diferencial do seu jogo, sem a preocupação com uma estrutura base para o jogo. Também se pretende através dessa estrutura ganhar desempenho na redução de tempo, qualidade do código e diminuir o número de falhas encontradas em jogos.

Através dos estudos realizados no Estudo de Caso foi possível observar resultados satisfatórios relacionados ao uso da LPS, com a diminuição de esforço e complexidade dos códigos, além da LPS permitir o desenvolvimento de diferentes tipos de jogos construcionistas. Porém, como projetos futuros, algumas melhorias ainda podem ser realizadas, como adicionar novos jogos construcionistas na análise da matriz de requisitos para refinar melhor os requisitos, assim como adicionar novas *features* que possam ser relevantes para jogos construcionistas.

Espera-se que através da LPS desenvolvida, os desenvolvedores indies voltados a jogos construcionistas possam encontrar mais facilidades que viabilizem a construção do jogo. Acredita-se que os desenvolvedores possam encontrar facilidades nos problemas como: organização das ideias e metodologia de desenvolvimento, devido ao direcionamento que uma Linha de Produto oferece; e redução da complexidade, do tempo e custo investidos durante a produção do jogo, além de melhorar a qualidade dos jogos desenvolvidos, devido aos recursos presentes já terem sidos usados e validados por outros jogos.

8.1.1 Resultados obtidos

Através deste estudo realizado foi possível obter os seguintes resultados:

- Um estudo sobre os conceitos de jogos construcionistas e o processo de criação de uma LPS;
- Construção de uma Linha de Produto de Software para jogos construcionistas;
- Construção de quatro jogos construcionistas através da LPS JIndie:
    - Sim Investigador
    - Logo
    - RoboCode sem a compilação condicional
    - RoboCode com a compilação condicional
- Um estudo comparativo entre software construído com e sem a compilação condicional;
- Um estudo sobre a influência de uma LPS no processo de desenvolvimento de jogos construcionistas;
- Um registro de software da LPS JIndie e o jogo Sim Investigador construídos através linguagem PHP.

8.1.2 Contribuições da pesquisa desenvolvida

As contribuições científicas presentes nesse trabalho são:

- Em relação a LPS desenvolvida neste mestrado, o JIndie tem como contribuição o modelo dos artefatos desenvolvidos durante as quatro etapas da Linha de Produto;
- A análise do domínio sobre jogos construcionistas;
- A publicação de quatro publicações de artigos sobre jogos na educação, construcionismo e Linhas de Produto de Software (LESSA FILHO et al, 2014a; LESSA FILHO et al, 2014b; LESSA FILHO et al, 2015; LESSA FILHO, DOMÍNGUEZ, 2016).

Como contribuições de engenharia:

- O desenvolvimento de uma LPS para jogos construcionistas;

- O trabalho contribuiu com o desenvolvimento de dois novos jogos, sendo o Logo uma versão da Linguagem Logo para *web* com os comandos e funcionalidades padrões da linguagem; e uma versão do jogo RoboCode para *web* com possibilidade de batalhas *multiplayer* através da rede social Facebook.

8.2 Trabalhos Futuros

Para trabalhos futuros, um estudo envolvendo a implantação de novas técnicas de implementação da variabilidade como a Programação Orientada a Aspecto, pode ser realizada, com a finalidade de buscar melhores desempenhos para a plataforma comum do JIndie.

Trabalhar com o desenvolvimento de jogos originais e aplicá-los em salas de aula, para avaliar a influencia dos jogos construcionistas desenvolvidos através do JIndie no ensino.

O desenvolvimento de novos jogos também pode permitir um ajuste melhor aos requisitos obrigatórios e opcionais presentes na Engenharia do Domínio, como também verificar o desempenho da ferramenta ao trabalhar com jogos de grande escala.

A implementação de novos recursos de animação envolvendo o gerador de mapa disponibilizado pela LPS pode melhorar a usabilidade de alguns jogos, removendo as limitações em relação à locomoção de objetos na imagem.

Também se planeja o desenvolvimento de um ambiente onde possa ser disponibilizada toda documentação da LPS e o *download* da plataforma comum, para que desenvolvedores e pesquisadores tenham acesso.

REFERÊNCIAS

ANASTASOPOULES, M.; GACEK, C. Implementing product line variabilities. In: **ACM SIGSOFT Software Engineering Notes**, v.26, n.3, 2001, pp. 109-117.

ARIFFIN, Mazeyanti Mohd; OXLEY, Alan; SULAIMAN, Suziah. Students' inclinations towards games and perceptions of Game-Based Learning (GBL). In**: International Conference on Computer and Information Sciences (ICCOINS), IEEE**. [s.l : s.n.], p. 1-6, 2014.

ARLEGUI, J. et al. **Discussing about IBSE, Constructivism and Robotics in (and out of the) Schools**. In: Constructionism, Paris. 2010.

BARB, A. S. et al. A statistical study of the relevance of lines of code measures in software projects. **Innovations in Systems and Software Engineering**, v.10, n.4, 2014, p.243-260.

BARROS, A. P. R. M.; AMARAL, R. B.; Um Estudo Sobre Pirâmides Em Um Ambiente De Base Construcionista. In: ENCONTRO NACIONAL DE EDUCAÇÃO MATEMATICA, 11. 2013, Curitiba. **Anais...** Curitiba: PUCPC: Sociedade Brasileira de Educação Matemática, 2013.

BARROS, A.P.R.M.; STIVAM, E. P. O software GeoGebra na Concepção de Micromundo. **Revista do Instituto GeoGebra Internacional de São Paulo**, São Paulo, v.1, n.1, 2012.

BELONIO, L. **Do Atari ao Zeebo: A história dos videogames no Brasil**. 2010. Disponível em: <https://historiadosgames.wordpress.com/2010/09/15/%E2%80%9Cdo-atari-ao-zeebo-a-historia-dos-videogames-no-brasil%E2%80%9D/ >. Acesso em: 14 jan. 2016.

BITTENCOURT, J.R.; OSÓRIO, F.S.. **Motores para Criação de Jogos Digitais:** Gráficos, Áudio, Interação, Rede, Intelligência Artificial e Física. 2006. Disponível em <http://osorio.wait4.org/publications/Bittencourt-Osorio-ERI-MG2006.pdf>. Acesso em: 15 jan. 2016.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language User Guide**. Addison-Wesley, 2005.

BORGES, J. R. **Levantamento da situação escolar em sistemas municipais de ensino no Rio Grande do Sul: Uma determinação política de financiamento do ensino público e/ou ferramenta de gestão?**. 2014. 304 f. Tese (Doutorado em Educação) - Universidade do Vale do Rio dos Sinos, São Leopoldo, RS, 2014.

BRAY, M., et al. **C4 Software Technology Reference Guide:** A Prototype. Carnegie Mellon University, Pittsburgh, Software Engineering Institute. 1997

CADAVID, A. N., IBARRA, D. G.; SALCEDO, S. L. Using 3-D Video Game Technology in Channel Modeling. **IEEE Access**, volume 2, 2014, p. 1652-1659.

CARMEN SANDIEGO. **Where in the world is Carmen Sandiego?**. 2015. Disponível em: <http://www.hmhco.com/parents-and-kids/the-learning-company/carmen-sandiego/history>. Acessado em: 16 jan. 2016.

CATETE, V., PEDDYCORD, B., BARNES, T. Augmenting introductory Computer Science Classes with GameMaker and Mobile Apps. ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 46.. 2015, Columbia,USA, **Anais…** Columbia, USA, 2015, p. 709-709

CHO, K. **Three Reasons Why Indie Devs Don't Finish Their Games**. Bunkyo: Gakuin University, 2009.

CLARK, M. N., et al. **Measuring software complexity to target risky modules in autonomous vehicle systems**. AUVSI North America Conference, San Diego, 2008

CLUA, E. W. G.; BITTENCOURT, J. R. Uma Nova Concepção para a Criação de Jogos Educativos. **Minicurso do Simpósio Brasileiro de Informática na Educação**. 2004.

DALMON, D. L., et al. A domain engineering for interactive learning modules. **Journal of Research and Practice in Information Technology**, 2012.

DOVOGAME. **Business Tycoon Online**. 2010. Disponível em: <http://bto.dovogame.com/btoguide/index.php>. Acesso em: 14 jan. 2016.

DUSTIN, E. **Effective Software Testing:** 50 Ways to Improve Your Software Testing. Boston: Addison-Wesley, 2002.

EKSTRAND, Michael D.; LUDWIG, Michael. Dependency Injection with Static Analysis and Context-Aware Policy. **Journal of Object Technology**, v. 15, n. 1, 2016.

ELLIS, B.; STYLOS, J.; MYERS, B. The factory pattern in API design: A usability evaluation. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING. 29., 2007, Minneapolis, USA. **Proceedings…** Minneapolis, USA: IEEE Computer Society, 2007. p. 302-312.

FEIGENSPAN, J., et al. Visual Support for Understanding Product Lines. In INTERNATIONAL CONFERENCE ON PROGRAM COMPREHENSION, 18., 2010, Los Alamitos, USA. **Proceedings …** Los Alamitos, USA: IEEE Computer Society, 2010, p. 34-35.

FEIN, G., et al. **Computer Space:** A conceptual and Developmental Analysis of LOGO. Constructivism in the Computer Age, Psychology Press. 2013.

FIGUEIREDO, Eduardo et al. Evolving software product lines with aspects. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING. 30., 2008, Leipzig, Germany**. Anais…** Leipzig, Germany**:** IEEE, 2008. p. 261-270.

FIRAXIS GAMES. **Civilization**. 2016. Disponível em: <http://www.civilization.com/en/games/>. Acesso em: 2 fev. 2016.

GOGUEN, J. A. Parameterized programming. **IEEE Transactions on Software Engineering**, n. 5, 1984, p. 528-543.

GREGORY, J.. **Game Engine Architecture**. Taylor and Francis Group. 2009.

GUENDOUZ, A.; BENNOUAR, D. A Software Product Line for Elearning Applications. In: CONFERENCE ON INFORMATION TECHNOLOGY, 14., 2013, Sudan. **Anais…** Sudan: University for Science and Technology, Khartoum, Sudan, 2013.

KAFAI, Y. B.. **Constructionism**. The Cambridge Handbook of The Learning Sciences, Cambridge: University Press, 2006, p.35-46.

KÄKÖLÄ, T., LEITNER, A. Introduction to Software Product Lines: Engineering, Service, and Management Minitrack. In: INTERNATIONAL CONFERENCE ON SYSTEM SCIENCE, 47., 2014, Hawaii. **Anais…** Hawaii: IEEE Computer Society, 2014.

KANG, Kyo C.; LEE, Jaejoon; DONOHOE, Patrick. Feature-oriented product line engineering. **IEEE software**, v. 19, n. 4, p. 58, 2002.

KÄSTNER, C., et al. Variability-aware parsing in the presence of lexical macros and conditional compilation. **ACM SIGPLAN Notices**, v. 46, n. 10, 2011, p.805-824).

KITCHER, P. **The Nature of Mathematical Knowledge**. New York: Oxford University Press. 1983.

KOHWALTER, T. C.; CLUA, E. W. G.; MURTA, L. G. P. Reinforcing Software Engineering Learning Through Provenance. In BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING. 28., 2014, Maceió. **Anais…** Maceió: UFAL, 2014.

LEBOW. Constructivist values for instructional systems design: Five principles toward a new mindset. **Educational Technology Research and Development**, v. 41, n. 3, 1993, p 4-16.

LESSA FILHO, C. A. C., DOMÍNGUEZ, A. H. JIndie: Uma Linha de Produto de Software para Jogos Educativos com foco no Construcionismo. In SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 27., 2016. Uberlândia, **No prelo**.

LESSA FILHO, C. A. C., et al. Sim Investigador: Um jogo construcionista. In SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 25., 2014**,** . Dourados, MS. **Anais...** Dourados, MS: [s.n.], 2014.

LESSA FILHO, C. A. C., et al. Um jogo digital baseado no construcionismo. **Revista Brasileira de Informática na Educação**, v. 23, n. 2., 2015.

LESSA FILHO, C. A. C., et al. Um Jogo Educativo na Web no Contexto do Ensino Fundamental. **Revista Novas Tecnologias na Educação**, v. 12, n. 2. 2014.

LOBO, A. L de C, et al. **A Systematic Approah for Architectural Design of Component-Based Product Lines**. Relatório Técnico, nov. 2007.

LUO, X., WEI, X., ZHANG, J. Guided Game-Based Learning Using Fuzzy Cognitive Maps. **IEEE Transactions On Learning Technologies**, v.3, n. 4, 2010, p.344-357.

MARLOWE, B. A.; CANESTARI, A. S. **Educational Psychology in Context Reading for future Teachers**. Sage Publications, India. 2006.

MATOS JÚNIOR, P. O. A. **Analysis of Tecnhiques for Implementing Software Product Lines Variabilites**. 2008. 121 f. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Pernambuco, Recife, 2008.

MAXIS. **SimCity**. 2014, Disponível em: <http://www.simcity.com/en_US/faq>. Acesso em 12 dez. 2015.

MCCABE, T. J. A complexity measure. **IEEE Transactions on Software Engineering**, n. 4, 1976, p. 308-320.

MCGREGO, J. D., et al Initiating Software Product Lines. **IEEE SOFTWARE,** 2002.

MEDEIROS, F.; LIMA, T.; DALTON, F.; RIBEIRO, M.; GHEYI, R.; FONSECA, B. Colligens. Disponível em: <https://sites.google.com/a/ic.ufal.br/colligens/home>, acesso em: 9 set. 2016.

MICROSOFT RESEARCH. **Code Hunter**. 2016. Disponível em: <https://www.codehunt.com/about.aspx>. Acesso em: 25 Jun. 2015.

MONTANGERO, J., NAVILLE, D. M.. **Piaget Or the Advance of Knowledge:** An Overview and Glossary. New York: Psychology Press. 2013.

MORELATO, L. A.; GUIMA, R. Y.; BORGES, M. A. F. Gene2: jogo via internet de apoio ao aprendizado de genética. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 19., 2008, Fortaleza. **Anais...** Fortaleza, 2008.

MUNIR, Q.; SHAHID, M. **Software Product Line: Survey of Tools.** 2010. 61 f. Tese (Doutorado) - Linköping University. Suecia. 2010.

NASCIMENTO, L. M.; ALMEIDA, E. S.; MEIRA, S. R. de L. A case study in software product lines-the case of the mobile game domain. In**:** EUROMICRO CONFERENCE : SOFTWARE ENGINEERING AND ADVANCED APPLICATIONS, 34., 2008, Washington. **Proceedings…** Washington: IEEE, 2008.

NORTHROP, L. M. SEI's software product line tenets. **IEEE software**, v. 19, n. 4, 2002, p. 32.

NUSEIBEH, B; EASTERBROOK, S. Requirements engineering: a roadmap. In: CONFERENCE ON THE FUTURE OF SOFTWARE ENGINEERING**,** 2000, New Yok. **Proceedings…** New Yok: ACM, 2000. p. 35-46.

Online Universities. **20 Educational Games That Were Ahead of Their Time**. 2012 Disponível em: <http://www.onlineuniversities.com/blog/2012/09/20-educational-games-that-were-ahead-their-time/>. Acesso em: 10 jan. 2016.

ORACLE. **Java**. 2016. Disponível em: <https://www.java.com/pt_BR/about/whatis_java.jsp>, acesso em: 28 jan. 2016.

OVERTON, W. F.. **Developmental Psychology:** Philosophy, Concepts, Methodology. Handbook of Child Psychology. vol. 1. 2006

PAPERT, S.; FREIRE, P. **The Future of School**. 2000. Disponível em: <http://www.papert.org/articles/freire/freirePart1.html>. Acesso em: 20 de mar. 2015.

PAPERT, S; HAREL, I. **Constructionis**. US: Ablex Publishing Corporation, 1991. Disponível em: <http://www.papert.org/articles/SituatingConstructionism.html>. Acesso em: 17 jan. 2015.

POHL, K. 2010. **Requirements Engineering**. Capítulo 4. Editora Springer-Verlag Berlin Heidelberg.

POHL, K.; BÖCKLE, G.; LINDEN, F. J V. D.**Software product line engineering:** foundations, principles and techniques. Berlin: Springer, 2005.

PRESSMAN, R. S. **Software Engineering:** A Practitioner's Approach. 7. ed., California: McGraw-Hill. 2010.

QUAN, N. **Distributed Game Environment:** A Software Product Line for Education and Research.2013. 30 f. Dissertação (Mestrado) - Faculty of Technology, Department of Computer Science. Linnaeus University, Suécia, 2013.

ROBOCODE. 2016. **Robocode**. Disponível em: <http://robocode.sourceforge.net/>. Acesso em: 17 jan. 2016.

ROBSON, C. **Real World Research**. 2 ed., Austrália: Blackwell, 2002.

ROCHA, R. V.; BITTENCOURT, I. I.; ISOTANI, S. Análise, Projeto, Desenvolvimento e Avaliação de Jogos Sérios e Afins: uma revisão de desafios e oportunidades. In SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 26.; WORKSHOP DE INFORMÁTICA NA ESCOLA, 21. 2015, Maceió. **Anais...** Maceió: UFAL, 20015, p.692-701.

ROGERS, S. **Level Up:** The Guide to Grede Video Game Designer. [S.l]: John Wiley & Sons. 2010. (From the game designer ofPac-World and Maximo).

ROUSH, M. Total Cost of Publishing One Indie Game. **Kotaku**. 2014. Disponível em: <http://numbers.kotaku.com/total-cost-of-publishing-an-indie-game-last-week-sixty-1618376285>. Acesso em: 28 jul. 2016.

RPG MAKER. **RPG Maker**. 2016. Disponível em: <http://www.rpgmakerweb.com>. Acesso em: 16 jan. 2016.

RUBEN, B. D. Simulations, Games, and Experience-Based Learning: The Quest for a New Paradigm for Teaching and Learning. **Simulation Gaming December**, v. 30, n. 4, 1999, p-498-505.

RUNESON, P., HÖST, M. Guidelines for conducting and reporting case study research in software engineering. **Empirical Software Engineering**, v. 14, n. 2, 2009, p-131-164.

SANTOS, W. O., et al. I. Avaliação de Jogos Educativos: Uma Abordagem no Ensino de Matemática. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 26.; WORKSHOP DE INFORMÁTICA NA ESCOLA, 21. 2015, Maceió. **Anais...** Maceió: UFAL, 20015, p.489-498.

SAVI R., DRA, V. R. U.. Jogos Digitais Educacionais: Benefícios e Desafios. **Revista Novas Tecnologias na Educação**. v.6, n.2. 2008

SILVA, A.. et al. Linhas de Produto de Software: Uma Tendência da Indústria. In: ENCONTRO REGIONAL DE INFORMÁTICA CEARÁ - PIAUÍ, 5. 2011, Fortaleza. **Anais...** Fortaleza: Sociedade Brasileira de Computação. 2011.

SIMON, Bart. Indie Eh? Some Kind of Game Studies. **Loading...**, v. 7, n. 11, 2012. Disponível em: <http://journals.sfu.ca/loading/index.php/loading/article/view/129/148>. Acesso em: 5 fev. 2016.

SOFTWARE ENGINEERING INSTITUTE **Arcade Game Maker Pedagogical Product Line:** Scope. Carnegie Mellon University. 2003

SOFTWARE ENGINEERING INSTITUTE. **Arcade Game Maker: Pedagogical Product Line**. 2009. Disponível em: <http://www.sei.cmu.edu/productlines/ppl/>. Acesso em: 5 fev. 2016.

THE PHP GROUP. **PHP:** Hypertext Preprocessor. 2016. Disponível em: <https://secure.php.net/manual/pt_BR/preface.php>. Acesso em: 18 jan. 2016.

THÜM, T. et al. FeatureIDE: An Extensible Framework for Feature-Oriented Software Development. **Science of Computer Programming**, 2014.

UNITY TECHNOLOGIES. **Unity.** 2016. Disponível em: <http://unity3d.com/pt/unity>. Acesso em: 18 jan. 2016.

WATSON, A. H.; MCCABE, T. J.; WALLACE, D. R. Structured testing: A testing methodology using the cyclomatic complexity metric. **National Institute of Standards and Technology**, v. 500, n. 235, 1996, p. 1-114.

WOLF, M. J. P. **A History from Pong to Playstation and Beyond**. Londres: Greenwood Press, 2008

WOOD JUNIOR, T. Fordismo, toyotismo e volvismo: os caminhos da indústria em busca do tempo perdido. **Revista de Administração de Empresas**, v. 32, n. 4, 1992, p. 6-18.

YACHT CLUB GAMES. Shovel Knight devs break down costs, sales. **GamesIndustry.** 2014. Disponível em: <http://www.gamesindustry.biz/articles/2014-08-06-shovel-knight-devs-break-down-costs-sales>, Acesso em: 28 jul. 2016.

YOON D. M.; KIM K. J. Challenges and Opportunities in Game Artificial Intelligence Education Using Angry Birds. **IEEE Access**, v. 3, 2015, p.793-804. Disponivel em: <http://cilab.sejong.ac.kr/home/lib/exe/fetch.php?media=public:paper:ieee_access.pdf>. Acesso em: 17 abr. 2016.

YOYO GAMES. **GameMaker**. 2016. Disponível em: <https://www.yoyogames.com/studio>. Acesso em: 18 jan. 2016.

APÊNDICES

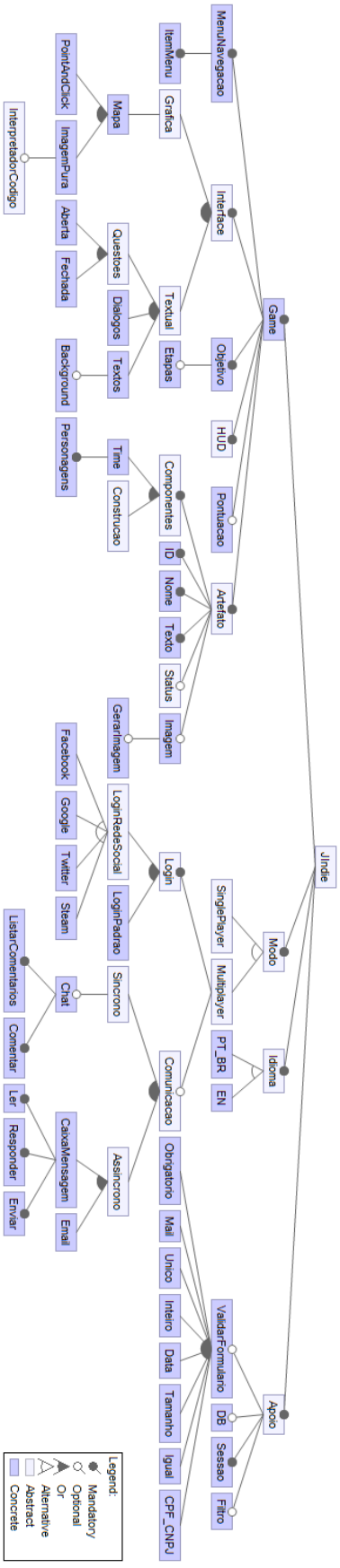
APÊNDICE A – Tabela Matriz de Requisitos para Jogos Construcionistas

Requisitos/Jogos	Sim Investigador	Linguagem Logo	RoboCode	SimCity	Civilization	Businnes Tycoon	Software Development	Gene2	Code Hunter	Geogebra	Total
R.1 Jogo Multiplayer	X		X	X	X	X					5
R.2 Jogo Single Player		X	X	X	X		X	X	X	X	8
R.3 Inserir código no jogo		X	X						X	X	4
R.4 Realizar login	X			X	X	X			X		5
R.5 Realizar Comunicação entre jogadores	X			X	X	X					4
R.6 Inserção de formulários;	X										1
R.7 Classificação por Rank;	X			X	X	X					4
R.8 Uso de pontuação	X		X	X	X	X			X		6
R.9 Uso de Questionários	X							X			2
R.10 Jogabilidade baseada em texto	X							X	X		3
R.11 Jogabilidade baseada em gráficos	X	X	X	X	X	X	X			X	8
R.12 Multi-idioma				X	X						2
R.13 Uso de personagens	X		X	X	X	X	X	X			7
R.14 Menu de navegação	X		X	X	X	X	X				6
R.15 Apresentar Objetivo			X		X	X	X	X	X		6
R.16 Uso de Mapa		X	X	X	X	X	X				6
R.17 HUD	X		X	X	X	X	X	X			7
R.18 Formação de equipe/time/exercito					X	X					2
R.19 Uso de número de tentativas (Contagem de Erros)	X								X		2
R.20 Contagem de atributos				X	X		X	X			4
R.21 Nível				X		X					2

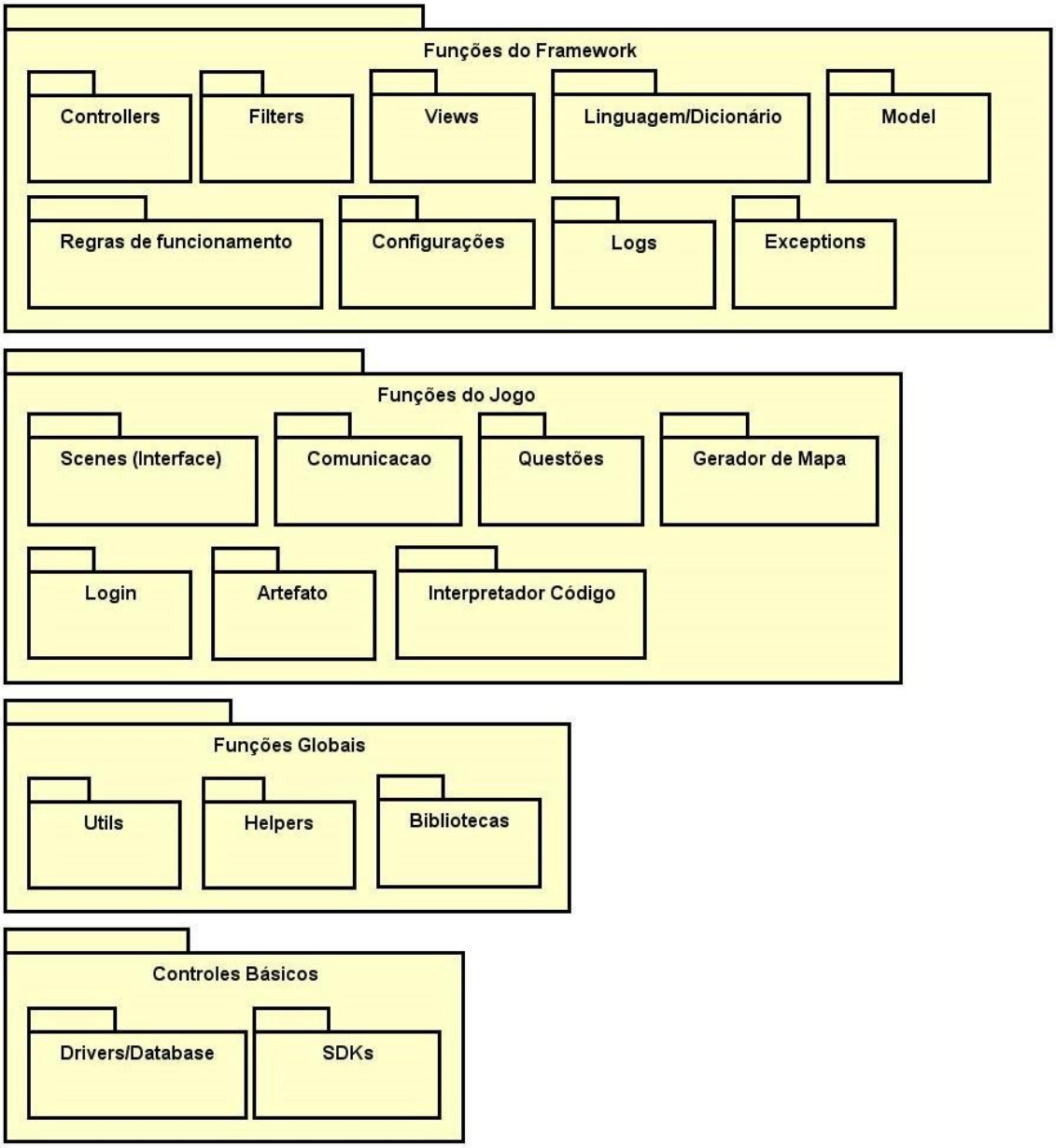
APÊNDICE B – Tabela Matriz de Requisitos para Artefatos dos Jogos Construcionistas

Requisitos/Jogos	Sim Investigador	Linguagem Logo	RoboCode	SimCity	Civilization	Businnes Tycoon Online	Software Development Manager	Gene2	Code Hunter	Geogebra	Total
R.1 Identificador ID	X		X	X	X	X	x	X			7
R.2 Nome	X		X	X		X	X	X			6
R.3 Saúde		X						X			2
R.4 Imagem		X	X					X		X	4
R.5 Dinheiro				X	X	X					3
R.6 População				X	X						2
R.7 Cliente							X				1
R.8 Texto	X	X	X						X	X	5
R.9 Status	X						X	X	X		4
R.10 Peso								X			1
R.11 Humor								X			1
R.12 Período					X						1
R.13 Felicidade								X			1
R.14 Classificação	X				X	X	X	X			5
R.15 Gerador de Imagem		X	X					X		X	4
R.16 Componentes	X			X	X	X	X				5

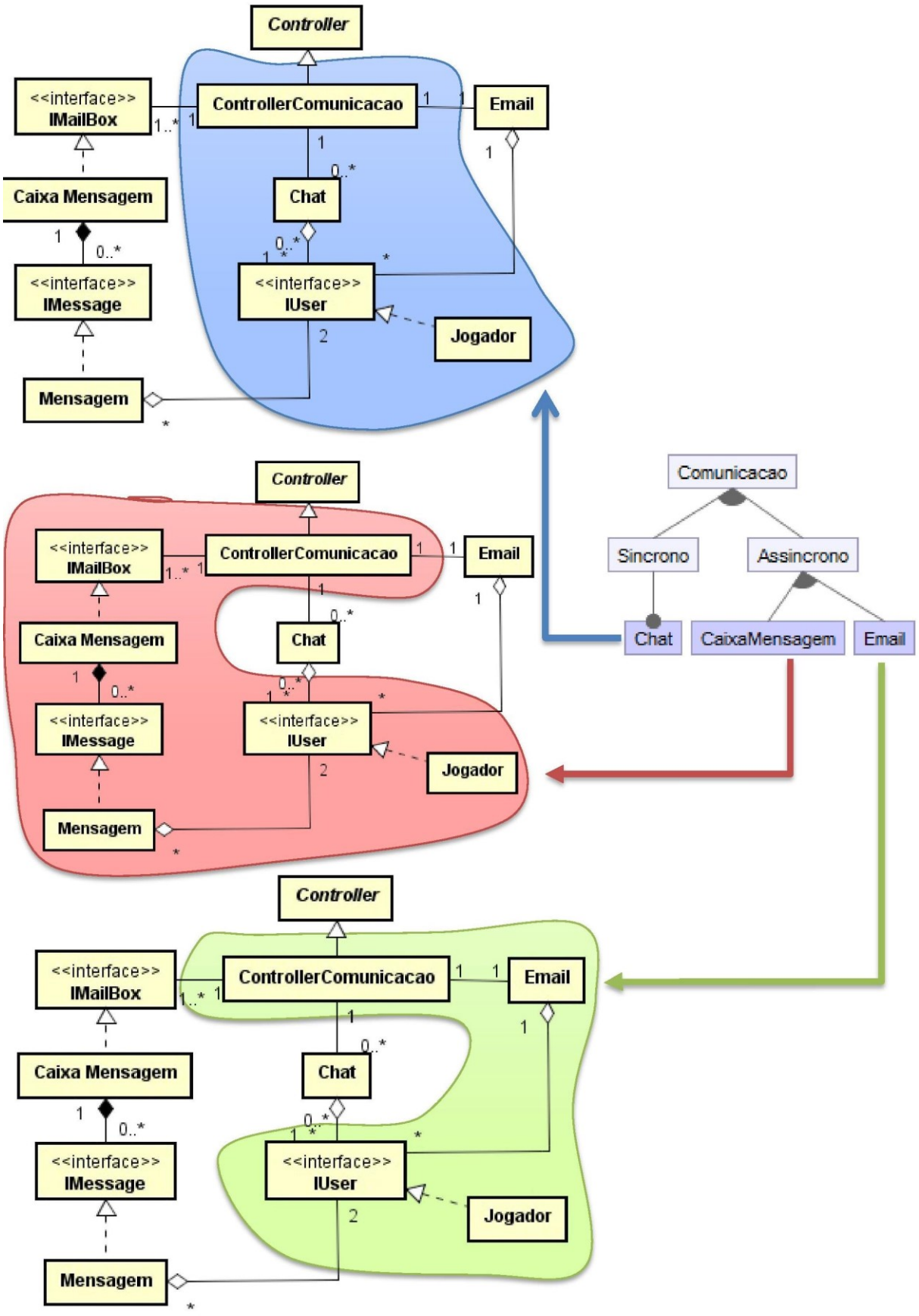
APÊNDICE C – Feature Model de JIndie com funções de apoio ao desenvolvimento



APÊNDICE D – Diagrama de Pacote da LPS JIndie



APÊNDICE E – Documentação do Diagrama de Classe do requisito de comunicação



APÊNDICE F – Código da Classe Game

```php
<?php
/**
*   JIndie
*   @package JIndie
*   @subpackage Game
*   @category Components of Game
*   @author Carlos W. Gama <carloswgama@gmail.com>
*   @copyright Copyright (c) 2015
*   @license   http://opensource.org/licenses/gpl-3.0.html GNU Public License
*   @version   1.0
*/

namespace JIndie\Game;

require_once(GAME_JI_PATH.'Goal.php');
require_once(GAME_JI_PATH.'Score.php');
require_once(GAME_JI_PATH.'Menu.php');
require_once(GAME_JI_PATH.'Artefact.php');
require_once(GAME_JI_PATH.'Component.php');
require_once(SYSTEM_PATH.'libraries/Session.php');

class Game {

    /**
    * @access protected
    * @var IArtefact
    */
    protected $artefact;

    /**
    * @access protected
    * @var Score
    */
    protected $score;

    /**
    * @access protected
    * @var Goal
    */
    protected $goal;

    /**
    * Exibe um conteúdo visto pelo usuário
    * @access protected
    * @var IScene
    */
    protected $scene;

    /**
    * @access protected
    * @var Menu
    */
    protected $menu;

    /**
    * @access protected
    * @var this
    */
    protected static $instance;

    /**
    * @access protected
    */
```

```
64.     protected function __construct() {
65.         $this->score     = new Score;
66.         $this->goal  = new Goal;
67.         $this->menu  = new Menu;
68.         $this->artefact = new Artefact;
69.     }
70.
71.     /**
72.     * @access public
73.     */
74.     public static function getInstance() {
75.         if (self::$instance == null) {
76.             $session = new \Session;
77.             $game = $session->loadGame();
78.
79.             if (!is_null($game)) {
80.                 self::$instance = $game;
81.                 \Log::message(\Language::getMessage('log', 'debug_game_load'), 2);
82.             }
83.             else {
84.                 $class = get_called_class();
85.                 self::$instance = new $class();
86.                 \Log::message(\Language::getMessage('log', 'debug_game_new', array('
    class_game' => $class)), 2);
87.             }
88.         }
89.         return self::$instance;
90.     }
91.
92.     /**
93.     * Exibe uma tela com dados do jogo e personagem para o jogador como vida/erros/.
    ..
94.     * @access public
95.     * @param bool $returnHTML
96.     * @return string   | Apenas se $returnHTML == true
97.     */
98.     public function showHUD($returnHTML = false) {
99.         Log::message(\Language::getMessage('log', 'debug_game_hud', array('return_ht
    ml' => ($returnHTML ? "TRUE" : "FALSE"))), 2);
100.                if ($returnHTML == true) {
101.                    ob_start();
102.                    include(VIEWS_PATH.'game/hud.php');
103.                    $html = ob_get_clean();
104.                    return $html;
105.                } else {
106.                    include(VIEWS_PATH.'game/hud.php');
107.                }
108.            }
109.
110.            /******* Getters and Setters ********/
111.            /**
112.            * @access public
113.            * @param IArtefact $artefact
114.            */
115.            public function setArtefact($artefact) {
116.                if ($artefact == null) {
117.                    \Log::message(\Language::getMessage('log', 'debug_game_artefact_n
    ull'), 2);
118.                    $this->artefact = null;
119.                }
120.                elseif ($artefact instanceof IArtefact) {
121.                    \Log::message(\Language::getMessage('log', 'debug_game_artefact')
    , 2);
122.                    $this->artefact = $artefact;
123.                }
124.                else {
```

```
125.                     $msg = \Language::getMessage('error', 'game_not_artefact');
126.                     \Log::message($msg, 2);
127.                     throw new \Exception($msg, 16);
128.                 }
129.             }
130.
131.             /**
132.             * @access public
133.             * @return IArtefact
134.             */
135.             public function getArtefact() {
136.                 return $this->artefact;
137.             }
138.
139.             /**
140.             * @access public
141.             * @param Score $score
142.             */
143.             public function setScore($score) {
144.                 if (is_subclass_of($score, 'JIndie\Game\Score') || is_a($score, 'JInd
    ie\Game\Score')) {
145.                     \Log::message(\Language::getMessage('log', 'debug_game_score'), 2
    );
146.                     $this->score = $score;
147.                 }
148.                 else {
149.                     $msg = \Language::getMessage('error', 'game_not_score');
150.                     \Log::message($msg, 2);
151.                     throw new Exception($msg, 17);
152.                 }
153.             }
154.
155.             /**
156.             * @access public
157.             * @return Score
158.             */
159.             public function getScore() {
160.                 return $this->score;
161.             }
162.
163.             /**
164.             * @access public
165.             * @param Goal $goal
166.             */
167.             public function setGoal($goal) {
168.                 if (is_subclass_of($goal, 'JIndie\Game\Goal') || is_a($goal, 'JIndie\
    Game\Goal')) {
169.                     \Log::message(Language::getMessage('log', 'debug_game_goal'), 2);

170.                     $this->goal = $goal;
171.                 }
172.                 else {
173.                     $msg = \Language::getMessage('error', 'game_not_goal');
174.                     \Log::message($msg, 2);
175.                     throw new Exception($msg, 18);
176.                 }
177.             }
178.
179.             /**
180.             * @access public
181.             * @return Goal
182.             */
183.             public function getGoal() {
184.                 return $this->goal;
185.             }
186.
```

```
187.            /**
188.            * @access public
189.            * @param IScene $scene
190.            */
191.            public function setScene($scene) {
192.                if ($scene instanceof IScene) {
193.                    $this->scene = $scene;
194.                    \Log::message(\Language::getMessage('log', 'debug_game_scene'), 2
    );
195.                } else {
196.                    $msg = \Language::getMessage('error', 'game_not_scene');
197.                    \Log::message($msg, 2);
198.                    throw new Exception($msg, 19);
199.                }
200.            }
201.
202.            /**
203.            * @access public
204.            * @return IScene
205.            */
206.            public function getScene() {
207.                return $this->scene;
208.            }
209.
210.            /**
211.            * Exibe os conteúdos da Scene
212.            * @param bool $returnHTML
213.            * @return string
214.            */
215.            public function showScene($returnHTML = false) {
216.                if (is_null($this->scene)) {
217.                    $msg = \Language::getMessage('error', 'game_scene_null');
218.                    \Log::message($msg, 2);
219.                    throw new \Exception($msg, 35);
220.                }
221.
222.                try {
223.                    \Log::message(\Language::getMessage('log', 'debug_game_scene_chec
    k'), 2);
224.                    $this->scene->check();
225.
226.                    if ($returnHTML) {
227.                        ob_start();
228.                        $this->scene->showScene();
229.                        return ob_get_clean();
230.                    }
231.                    else
232.                        $this->scene->showScene();
233.                } catch (Exception $ex) {
234.                        \Log::message("GAME/Scene: " . $ex->getMessage(), 2);
235.                        throw new Exception($ex->getMessage(), $ex->getCode());
236.                }
237.            }
238.
239.            /**
240.            * @access public
241.            * @param Menu $menu
242.            */
243.            public function setMenu($menu) {
244.                if (is_subclass_of($menu, 'JIndie\Game\Menu') || is_a($menu, 'JIndie\
    Game\Menu')) {
245.                    \Log::message(\Language::getMessage('log', 'debug_game_menu'), 2)
    ;
246.                    $this->menu = $menu;
247.                }
248.                else {
```

```
249.                 $msg = \Language::getMessage('error', 'game_not_menu');
250.                 \Log::message($msg, 2);
251.                 throw new Exception($msg, 20);
252.             }
253.         }
254.
255.         /**
256.         * @access public
257.         * @return Menu
258.         */
259.         public function getMenu() {
260.             return $this->menu;
261.         }
262.     }
```

APÊNDICE G – Código da Classe Gerador de Mapas

```php
<?php
/**
*   JIndie
*   @package JIndie
*   @category Library
*   @author Carlos W. Gama <carloswgama@gmail.com>
*   @copyright Copyright (c) 2015
*   @license   http://opensource.org/licenses/gpl-3.0.html GNU Public License
*   @version   1.0
*/

class MapGenerator {

    /**
    * Tamanho das imagens no mapa
    * @access protected
    * @var int
    */
    protected $imageSize = 32;

    /**
    * Tamanho do mapa em X,Y
    * @access protected
    * @var array|Vector
    */
    protected $mapSize = array('x' => 5, 'y' => 5);

    /**
    * A posição atual do ponteiro na adição dos tiles
    * @access protected
    * @var array|Vector
    */
    protected $currentPosition = array('x' => 1, 'y' => 1);

    /**
    * Lista de tiles
    * @access protected
    * @var array|Tiles
    */
    protected $tiles = array();

    /**
    * Tile padrão a ser inserido quando não houve nenhum tile na posição
    * @access protected
    * @var array|Tile
    */
    protected $defaultTile = array('field' => '
NSUhEUgAAAAoAAAAKCAYAAACNMs+9AAAAFElEQVR42mNI687+TwxmGFVIX4UAWFTrjQunmzcAAAAASUVORK5
CYII=');

    /**
    * Usuário pode clicar caso haja URL configurada
    * @access protected
    * @var bool
    */
    protected $clickable = true;
    /**
    * @param int $size
    */
    public function setImageSize($size) {
        if (filter_var($size, FILTER_VALIDATE_INT) && $size > 0) {
            $this->imageSize = $size;

```

```
62.             $msg = Language::getMessage('log', 'debug_map_new_image_size', array('si
    ze' => $size));
63.             Log::message($msg, 2);
64.         }
65.         else {
66.             $msg = Language::getMessage('map', 'error_int_val');
67.             Log::message($msg, 2);
68.             throw new Exception($msg, 23);
69.         }
70.     }
71.
72.     /**
73.     * Define os tile padrão
74.     * @uses $this->mapGenerator->setDefaultTile('link-field', 'link-object', 'link-
    field');
75.     * @uses $this->mapGenerator->setDefaultTile(array('field' => 'link-
    field', 'object' => 'link-object', 'url' => 'link-field'));
76.     * @param string|array $field
77.     * @param string $object
78.     * @param string $url
79.     */
80.     public function setDefaultTile($field, $object = null, $url = null) {
81.         $this->defaultTile = array('field' => "", 'object' => "", 'url' => "");
82.
83.         if (is_array($field)) {
84.             extract($field);
85.         }
86.
87.         if (!is_null($field)) $this->defaultTile['field'] = $field;
88.         if (!is_null($object)) $this->defaultTile['object'] = $object;
89.         if (!is_null($url)) $this->defaultTile['url'] = $url;
90.
91.         $msg = Language::getMessage('log', 'debug_map_new_default_tile', array('tile
    ' => json_encode($this->defaultTile)));
92.         Log::message($msg, 2);
93.     }
94.
95.     /**
96.     * Define o tamanho do mapa
97.     * @uses $this->mapGenerator->setMapSize(array(10, 10));
98.     * @uses $this->mapGenerator->setMapSize(10, 10);
99.     * @param array|int $vector
100.          * @param int $y
101.          */
102.         public function setMapSize($vector, $y = null) {
103.             if (!is_null($y))
104.                 $vector = array($vector, $y);
105.
106.
107.             $this->validVector($vector);
108.             $vector = $this->formatVector($vector);
109.
110.             $this->resetMap();
111.             $this->mapSize = $vector;
112.
113.             $msg = Language::getMessage('log', 'debug_map_new_map_size', array('m
    ap' => '('.$this->mapSize['x'].','.$this->mapSize['y'].')'));
114.             Log::message($msg, 2);
115.         }
116.
117.         /**
118.         * Retorna o tamanho do Mapa Definido
119.         * @return array|Vector(X,Y)
120.         */
121.         public function getMapSize() {
122.             return $this->mapSize;
```

```
123.          }
124.
125.          /**
126.          * Zera todas as informações relacionada ao mapa atual
127.          */
128.          public function resetMap() {
129.              $this->tiles = array();
130.              $this->mapSize = null;
131.              $this->currentPosition = array('x' => 1, 'y' => 1);
132.
133.              $msg = Language::getMessage('log', 'debug_map_reset_map');
134.              Log::message($msg, 2);
135.          }
136.
137.          /**
138.          * Adiciona um novo tile na posição atual do ponteiro e avança o ponteiro
    para a próxima posição válida
139.          * @uses $this->mapGenerator->addTile('link-field', 'link-object', 'link-
    field');
140.          * @uses $this->mapGenerator->addTile(array('field' => 'link-
    field', 'object' => 'link-object', 'url' => 'link-field'));
141.          * @param string|array $field
142.          * @param string $object
143.          * @param string $url
144.          */
145.          public function addTile($field, $object = null, $url = null) {
146.              //Add by Array
147.              if (is_array($field)) {
148.
149.                  $f = null;
150.                  $o = null;
151.                  $u = null;
152.                  //By Name
153.                  if (isset($field['field']) || isset($field['object']) || isset($f
    ield['url'])) {
154.                      if (isset($field['field']))     $f = $field['field'];
155.                      if (isset($field['object']))    $o = $field['object'];
156.                      if (isset($field['url']))       $u = $field['url'];
157.                  }
158.                  //By Position
159.                  elseif (isset($field[0]) || isset($field[1]) || isset($field[2]))
     {
160.                      if (isset($field[0]))   $f = $field[0];
161.                      if (isset($field[1]))   $o = $field[1];
162.                      if (isset($field[2]))   $u = $field[2];
163.                  }
164.
165.                  $this->addTile($f, $o, $u);
166.              } else {
167.                  //Add on currentPosition
168.                  list($x, $y) = array_values($this->currentPosition);
169.
170.                  $this->tiles[$x][$y] = array(
171.                      'field'         => $field,
172.                      'object'        => $object,
173.                      'url'           => $url
174.                  );
175.
176.                  $msg = Language::getMessage('log', 'debug_map_new_tile', array('p
    osition' => '(' . $x . ',' . $y . ')'));
177.                  Log::message($msg, 2);
178.
179.                  //Change vector to new position
180.                  if (array_diff_assoc($this->currentPosition, $this->mapSize)) {
181.                      $x++;
182.
```

```
183.                         if ($x > $this->mapSize['x']) {
184.                             $x = 1;
185.                             $y++;
186.                         }
187.
188.                         $this->currentPosition = array('x' => $x, 'y' => $y);
189.
190.                         $msg = Language::getMessage('log', 'debug_map_current_positio
    n', array('position' => '(' . $this->currentPosition['x'] . ',' . $this-
    >currentPosition['y'] . ')'));
191.                         Log::message($msg, 2);
192.                     }
193.                 }
194.             }
195.
196.             /**
197.             * Adiciona um novo tile numa posição determinada
198.             * @uses $this->mapGenerator->addTileAtPosition(array(10, 10), 'link-
    field', 'link-object', 'link-field');
199.             * @uses $this->mapGenerator-
    >addTileAtPosition(array(10, 10), array('field' => 'link-field', 'object' => 'link-
    object', 'url' => 'link-field'));
200.             * @param array $position
201.             * @param string|array $field
202.             * @param string $object
203.             * @param string $url
204.             */
205.             public function addTileAtPosition($position, $field, $object = null, $url
    ToAction = null) {
206.                 //Check
207.                 $this->validVector($position);
208.                 $vector = $this->formatVector($position);
209.
210.                 //Save old positions
211.                 list($x, $y) = array_values($this->currentPosition);
212.
213.                 //Change Position
214.                 $this->currentPosition =  $vector;
215.
216.                 //Save Tile
217.                 $this->addTile($field, $object, $urlToAction);
218.
219.                 //Return to old position
220.                 $this->currentPosition = array($x, $y);
221.             }
222.
223.             /**
224.             * Altera o valor de um tile
225.             * @uses $this->mapGenerator->alterTile(array(10, 10), 'object', 'new-
    link');
226.             * @uses $this->mapGenerator-
    >alterTile(array(10, 10), array('object' => 'new-link', 'url' => 'new-url'));
227.             * @param array $position
228.             * @param array|string $param
229.             * @param string $value
230.             */
231.             public function alterTile($position, $param, $value = null) {
232.                 $this->validVector($position, true, true);
233.                 $vector = $this->formatVector($position);
234.
235.                 //Array
236.                 if (is_array($param)) {
237.                     foreach ($param as $key => $value)
238.                         $this->alterTile($vector, $key, $value);
239.                 } else {
240.
```

```
241.                 if (isset($this->tiles[$vector['x']][$vector['y']])) {
242.                     if (in_array($param, array('object', 'field', 'url'))) {
243.                         $this-
    >tiles[$vector['x']][$vector['y']][$param] = $value;
244.
245.                         $msg = Language::getMessage('log', 'debug_map_alter_value
    ', array('position' => '('.$vector['x'].','.$vector['y'].')', 'param' => $param, 'va
    lue' => $value));
246.                         Log::message($msg, 2);
247.                     }
248.                     else {
249.                         $msg = Language::getMessage('map', 'error_param_not_exist
    s', array('param' => $param));
250.                         Log::message($msg, 2);
251.                         throw new Exception($msg, 24);
252.                     }
253.                 }
254.                 else {
255.                     $msg = Language::getMessage('map', 'tile_not_exists', array('
    position' => $vector['x'].','.$vector['y']));
256.                     Log::message($msg, 2);
257.                     throw new Exception($msg, 25);
258.                 }
259.             }
260.         }
261.
262.         /**
263.         * Retorna um tile pela sua posição
264.         * @uses $this->mapGenerator->getTile(array(10, 10));
265.         * @uses $this->mapGenerator->getTile(10, 10);
266.         * @param int|array $position
267.         * @param int $int
268.         * @return array|Tile
269.         */
270.         public function getTile($position, $y = null) {
271.             if (!is_null($y))
272.                 $position = array($position, $y);
273.
274.             $this->validVector($position, true, true);
275.             $vector = $this->formatVector($position);
276.             return $this->tiles[$vector['x']][$vector['y']];
277.         }
278.
279.         /**
280.         * Insere uma lista de $tiles organizados nas posição X,Y
281.         * @param array $tiles
282.         */
283.         public function setMap($tiles) {
284.             if (!is_array($tiles)) {
285.                 $msg = Language::getMessage('map', 'not_array');
286.                 Log::message($msg, 2);
287.                 throw new Exception($msg, 26);
288.             }
289.
290.             $newTiles = array();
291.             $maxValueX = 0;
292.             $maxValueY = 0;
293.             foreach ($tiles as $x => $row) {
294.
295.                 if (!filter_var($x, FILTER_VALIDATE_INT) && $x > 0) {
296.                     $msg = Language::getMessage('map', 'error_position');
297.                     Log::message($msg, 2);
298.                     throw new Exception($msg, 27);
299.                 }
300.
301.                 if (!is_array($row)) {
```

```
302.                         $msg = Language::getMessage('map', 'error_map_structure');
303.                         Log::message($msg, 2);
304.                         throw new Exception($msg, 28);
305.                     }
306.
307.                     if ($maxValueX < $x)
308.                         $maxValueX = $x;
309.
310.                     foreach ($row as $y => $tile) {
311.
312.                         if (!filter_var($y, FILTER_VALIDATE_INT) && $y > 0) {
313.                             $msg = Language::getMessage('map', 'error_position');
314.                             Log::message($msg, 2);
315.                             throw new Exception($msg, 27);
316.                         }
317.
318.                         if (!is_array($tile)) {
319.                             $msg = Language::getMessage('map', 'error_tile_structure'
    , array('position' => $x.','.$y));
320.                             Log::message($msg, 2);
321.                             throw new Exception($msg, 29);
322.                         }
323.
324.
325.                         if ($maxValueY < $y)
326.                             $maxValueY = $y;
327.
328.                         $newTiles[$x][$y] = $tile;
329.                     }
330.                 }
331.
332.                 $this->setMapSize($maxValueX, $maxValueY);
333.                 $this->tiles = $newTiles;
334.
335.                 $msg = Language::getMessage('log', 'debug_map_set_map');
336.                 Log::message($msg, 2);
337.             }
338.
339.             public function getMap($full = false) {
340.                 if (!$full)
341.                     return $this->tiles;
342.
343.                 //Com o Default
344.                 $tiles = array();
345.
346.                 for ($x = 1; $x <= $this->mapSize['x']; $x++) {
347.                     for ($y = 1; $y <= $this->mapSize['y']; $y++) {
348.                         if (isset($this->tiles[$x][$y]))
349.                             $tiles[$x][$y] = $this->tiles[$x][$y];
350.                         else
351.                             $tiles[$x][$y] = $this->defaultTile;
352.                     }
353.                 }
354.                 return $tiles;
355.             }
356.
357.             /**
358.             * Gera o mapa
359.             * @param bool $returnHTML
360.             * @return string |Caso returnHTML = true
361.             */
362.             public function generate($returnHTML = true) {
363.
364.                 $msg = Language::getMessage('log', 'debug_map_generate');
365.                 Log::message($msg, 2);
366.
```

```
367.                //
368.                $imageSize = $this->imageSize;
369.                $map['tiles'] = $this->getMap(true);
370.                $map['clickable'] = $this->isClickable();
371.                $map = json_encode($map);
372.
373.
374.                if ($returnHTML) {
375.                    ob_start();
376.                    include(APP_PATH.'views/library/map.php');
377.                    $map = ob_get_clean();
378.                    return $map;
379.                } else
380.                    include(APP_PATH.'views/library/map.php');
381.            }
382.
383.            /**
384.            * Gera imagens de cores solidas
385.            * @param string $color
386.            * @return string
387.            */
388.            public function getColor($color) {
389.                switch ($color) {
390.                    case "green":
391.                        return "
    KCAYAAACNMs+9AAAAFElEQVR42mNI687+TwxmGFVIX4UAWFTrjQunmzcAAAAASUVORK5CYII=";
392.                        break;
393.                    case "red":
394.                        return "
    KCAYAAACNMs+9AAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADsQAAA7EAZUrDhsAAAB
    PSURBVChTY/zp7Pyf+d49BgZWFgas4Pcfhr9KSgwMfxQV/v9nYMCLQWqY/rOyYjcJSRSkhomgKqiCIaGQ8fd
    vgv4BqWGEBfh/HAHOCA1wAJO6KnWcSPzpAAAAAElFTkSuQmCC";
395.                        break;
396.                    case "blue":
397.                        return "
    KCAIAAAF1V2h8AAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAAAAD9JREFUeNpilA1axMD
    AwMQABgABxADkgZgAAcQIEQUCgABiRJGHUgABhJBGEYYDgAAiJE8qHyDA0M0joJye0gDR7geYDW9lNQAAAAB
    JRU5ErkJggg==";
398.                        break;
399.                    case "white":
400.                        return "
    KCAYAAACNMs+9AAAABHNCSVQICAgIfAhkiAAAAAlwSFlzAAALEgAACxIB0t1+/AAAAB90RVh0U29mdHdhcmU
    ATWFjcm9tZWRpYSBGaXJld29ya3MgOLVo0ngAAAAWdEVYdENyZWF0aW9uIFRpbWUAMTAvMDMvMTNA0CjrAAA
    AGElEQVQYlWP8//+/PAMRgIkYRaMKqacQAAZLAzDnLBG8AAAAAElFTkSuQmCC";
401.                        break;
402.                    case "gray":
403.                    case "grey":
404.                        return "
    KCAQAAAFQPDegAAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAAAABlJREFUGNNjYFADQhg
    BYyJEkAk16jNpKggAKCIO2TdVg/oAAAAASUVORK5CYII=";
405.                        break;
406.                    case "yellow":
407.                        return "
    KCAYAAACNMs+9AAAABHNCSVQICAgIfAhkiAAAAAlwSFlzAAALEgAACxIB0t1+/AAAABx0RVh0U29mdHdhcmU
    AQWRvYmUgRmlyZXdvcmtzIENTNui8sowAAAAWdEVYdENyZWF0aW9uIFRpbWUAMDMvMjgvMTSON3MqAAAAGEl
    EQVQYlWP8/5/hPwMRgIkYRaMKqacQANQTAxHwovCxAAAAAElFTkSuQmCC";
408.                        break;
409.                    case "pink":
410.                        return "
    KAQMAAAC3/F3+AAAABGdBTUEAALGPC/xhBQAAAAZQTFRF+AV7AAAAKHiKVgAAAAtJREFUGNNjYMAHAAAeAAH
    JmCODAAAAAElFTkSuQmCC";
411.                        break;
412.                    case "black":
413.                    default:
414.                        return "
    KEAIAAABSwISpAAAACXBIWXMAAABIAAAASABGyWs+AAAACXZwQWcAAAAKAAAACgBOpnblAAAAD0lEQVQoz2N
```

```
    gGAWjgJYAAAJiAAEQ3MCgAAAAJXRFWHRjcmVhdGUtZGF0ZQAyMDA5LTA3LTA4VDE5OjE1OjMyKzAyOjAwm1P
    ZQQAAACV0RVh0bW9kaWZ5LWRhdGUAMjAwOS0wNy0wOFQxOToxNTozMiswMjowMMTir3UAAAAASUVORK5CYII
    =";
                        break;
                }
            }

            /**
            * @param bool clickable
            */
            public function setClickable($clickable) {
                $this->clickable = (bool) $clickable;
            }

            /**
            * @return bool
            */
            public function isClickable() {
                return $this->clickable;
            }

            /**
            * Realiza validações no Vector(x, y)
            * @access protected
            * @param array $vector
            * @param bool $zero |Caso true, o vector não pode ter posições X < 1 e Y
    < 1
            * @param bool $isset |Caso true, é preciso ter um tile na posição informa
    da do vector
            */
            protected function validVector($vector, $zero = false, $isset = false) {

                if (!is_array($vector)) {
                    $msg = Language::getMessage('map', 'not_array');
                    Log::message($msg, 2);
                    throw new Exception($msg, 26);
                }

                if (count($vector) != 2) {
                    $msg = Language::getMessage('map', 'error_count_param_vector');
                    Log::message($msg, 2);
                    throw new Exception($msg, 29);
                }

                //formata o vetor
                $vector = $this->formatVector($vector);

                if (!filter_var($vector['x'], FILTER_VALIDATE_INT) || !filter_var($ve
    ctor['y'], FILTER_VALIDATE_INT)) {
                    $msg = Language::getMessage('map', 'error_position');
                    Log::message($msg, 2);
                    throw new Exception($msg, 27);
                }

                if ($zero) {
                    if (($vector['x'] < 1) || ($vector['y'] < 1)) {
                        $msg = Language::getMessage('map', 'error_position_zero');
                        Log::message($msg, 2);
                        throw new Exception($msg, 31);
                    }
                }

                if ($isset) {
                    if (!$this->hasTile($vector)) {
                        $msg = Language::getMessage('map', 'tile_not_exists', array('
    position' => $vector['x'].','.$vector['y']));
```

```
473.                     Log::message($msg, 2);
474.                     throw new Exception($msg, 25);
475.                 }
476.             }
477.         }
478.
479.         /**
480.         * @access protected
481.         * @param array $vector
482.         * @return array
483.         */
484.         protected function formatVector($vector) {
485.             if (!isset($vector['x']) || isset($vector['y'])) {
486.                 $vector = array('x' => current($vector), 'y' => end($vector));
487.             }
488.
489.             return $vector;
490.         }
491.
492.         /**
493.         * @return array
494.         */
495.         public function getDefaultTile() {
496.             return $this->defaultTile;
497.         }
498.
499.         public function hasTile($vector, $y = null) {
500.             if (!is_null($y))
501.                 $vector = array($vector, $y);
502.
503.             try {
504.                 $vector = $this->formatVector($vector, true);
505.                 return isset($this->tiles[$vector['x']][$vector['y']]);
506.             } catch (Exception $ex) {
507.                 return false;
508.             }
509.
510.
511.         }
512.     }
```

APÊNDICE H – Código da classe de Interpretador de código

```php
<?php
/**
*   JIndie
*   @package JIndie
*   @category Library
*   @author Carlos W. Gama <carloswgama@gmail.com>
*   @copyright Copyright (c) 2015
*   @license   http://opensource.org/licenses/gpl-3.0.html GNU Public License
*   @version   1.0
*/

use JIndie\Code\ICode;

class CodeReader {

    /**
    * Tipo de código escolhido | Código puro, navegação...
    * @access protected
    * @var string
    */
    protected $codeName;

    /**
    * Total de linha do código do usuário
    * @access protected
    * @var int
    */
    protected $totalLine;

    /**
    * Linha atual sendo executada
    * @access protected
    * @var int
    */
    protected $currentLine;

    /**
    * Linhas de código
    * @access protected
    * @var array
    */
    protected $lines;

    /**
    * Código recebido do usuário a ser executado
    * @access protected
    * @var string
    */
    protected $script;

    /**
    * Class onde ficam as regras do código que será executado
    * @access protected
    * @var ICode
    */
    protected $code;

    /**
    * Armazena a mensagen de erro
    * @access protected
    * @var array
    */
    protected $error = array();
```

```
64.
65.     /**
66.     * Variável que serve para armazenar informações na montagem das linhas
67.     * @access private
68.     * @var array
69.     */
70.     private $auxLexer = array();
71.
72.     /**
73.     * Variavel que armazena o momento que o script começou a ser executado, para con
    trolar o tempo máximo de execução
74.     * @var int
75.     */
76.     private $timeExecution = 0;
77.
78.     public function __construct() {
79.         $this->setRules('Navegation');
80.     }
81.     /****************** PREPARAÇÃO ********************/
82.     /**
83.     * Classifica o script por tipo e dividi por linhas
84.     * @access private
85.     * @return array (Linhas)
86.     */
87.     private function explodeLines() {
88.
89.         //Quebra Linhas
90.         $this->auxLexer['lines']     = explode($this->code->getBreakLine(), $this-
    >script);
91.         $this->auxLexer['position'] = 0;
92.         $this->totalLine             = 0;
93.
94.         //Remove linhs em branco no final
95.         for ($i = (count($this->auxLexer['lines']) - 1); $i >= 0; $i--) {
96.             if (emptyempty(trim($this->auxLexer['lines'][$i])) || $this-
    >auxLexer['lines'][$i] == "\n")
97.                 unset($this->auxLexer['lines'][$i]);
98.             else
99.                 break;
100.            }
101.
102.            return $this->lexerLines();
103.        }
104.
105.        /**
106.        * Classifica as linhas por tipo (COMMAND, IF, ELSE, END-IF, WHILE, END-
    WHILE)
107.        * @access private
108.        * @return array (Linhas)
109.        */
110.        private function lexerLines($returnCondition = null) {
111.            //Recupera as estruturas do IF e WHILE
112.            $ifStructure        = '/^' . str_replace('[CONDITION]', '(.*)', $this
    ->code->getIfStructure()) . '/' . ($this->code->isCaseSensitive()? "" :"i");
113.            $elseStructure      = '/^' . $this->code-
    >getElseStructure() . '/'  . ($this->code->isCaseSensitive()? "" :"i");
114.            $endIfStructure     = '/^' . $this->code-
    >getEndIfStructure() . '/'  . ($this->code->isCaseSensitive()? "" :"i");
115.            $whileStructure     = '/^' . str_replace('[CONDITION]', '(.*)', $this
    ->code->getWhileStructure()) . '/'  . ($this->code->isCaseSensitive()? "" :"i");
116.            $endWhileStructure  = '/^' . $this->code-
    >getEndWhileStructure() . '/'  . ($this->code->isCaseSensitive()? "" :"i");
117.            $lines = array();
118.
119.            //Atribuid o que a linha representa e seta a linha em sua Key
```

```
120.                for (; $this->auxLexer['position'] < count($this-
    >auxLexer['lines']);) {
121.                    $line = trim($this->auxLexer['lines'][$this-
    >auxLexer['position']]);
122.                    $this->totalLine++;
123.                    $this->auxLexer['position']++;
124.
125.                    //IF
126.                    if ($ifStructure != '/^/' && preg_match($ifStructure, $line, $mat
    ch)) {
127.                        $nextCommand = trim(substr($line, strlen($match[0])));
128.
129.                        if (!emptyempty($nextCommand))
130.                            array_splice($this->auxLexer['lines'], $this-
    >auxLexer['position'], 0, $nextCommand);
131.
132.
133.                        $lines[$this->totalLine] = array(
134.                            'line'          => $match[0],
135.                            'type'          => "IF",
136.                            'condition'     => $match[1],
137.                            'statements'    => $this->lexerLines("END-IF")
138.                        );
139.                        continue;
140.                    }
141.
142.                    //ENDIF
143.                    if ($endIfStructure != '/^/' && preg_match($endIfStructure, $line
    , $match)) {
144.                        $nextCommand = trim(substr($line, strlen($match[0])));
145.                        if (!emptyempty($nextCommand))
146.                            array_splice($this->auxLexer['lines'], $this-
    >auxLexer['position'], 0, $nextCommand);
147.
148.                        $lines[$this->totalLine] = array(
149.                            'line'          => $match[0],
150.                            'type'          => "END-IF",
151.                        );
152.
153.                        if (!is_null($returnCondition) && ($returnCondition == "END-
    IF"))
154.                            return $lines;
155.                    }
156.
157.                    //ELSE
158.                    if ($elseStructure != '/^/' && preg_match($elseStructure, $line,
    $match)) {
159.                        $nextCommand = trim(substr($line, strlen($match[0])));
160.
161.                        if (!emptyempty($nextCommand))
162.                            array_splice($this->auxLexer['lines'], $this-
    >auxLexer['position'], 0, $nextCommand);
163.
164.                        $lines[$this->totalLine] = array(
165.                            'line'          => $match[0],
166.                            'type'          => "ELSE",
167.                            'statements'    => $this->lexerLines("END-IF")
168.                        );
169.                        continue;
170.                    }
171.
172.                    //WHILE
173.                    if ($whileStructure != '/^/' && preg_match($whileStructure, $line
    , $match)) {
174.                        $nextCommand = trim(substr($line, strlen($match[0])));
175.
```

```
176.                         if (!emptyempty($nextCommand))
177.                             array_splice($this->auxLexer['lines'], $this-
    >auxLexer['position'], 0, $nextCommand);
178.
179.                         $lines[$this->totalLine] = array(
180.                             'line'          => $match[0],
181.                             'type'          => "WHILE",
182.                             'condition'     => $match[1],
183.                             'statements'    => $this->lexerLines("END-WHILE")
184.                         );
185.                         continue;
186.                     }
187.
188.                     //ENDWHILE
189.                     if ($endWhileStructure != '/^/' && preg_match($endWhileStructure,
     $line, $match)) {
190.                         $nextCommand = trim(substr($line, strlen($match[0])));
191.
192.                         if (!emptyempty($nextCommand))
193.                             array_splice($this->auxLexer['lines'], $this-
    >auxLexer['position'], 0, $nextCommand);
194.
195.                         $lines[$this->totalLine] = array(
196.                             'line'          => $match[0],
197.                             'type'          => "END-WHILE",
198.                         );
199.
200.                         if (!is_null($returnCondition) && ($returnCondition == "END-
    WHILE"))
201.                             return $lines;
202.                     }
203.
204.                     //COMMAND
205.                     if (emptyempty($lines[$this->totalLine])) {
206.                         $lines[$this->totalLine] = array(
207.                             'line'      => $line,
208.                             'type'      => "COMMAND"
209.                         );
210.                     }
211.                 }
212.                 return $lines;
213.             }
214.
215.             /***************************** EXECUÇÃO ******************************
    /
216.             /**
217.             * Método que apenas verifica se o script está correto, mas não o executa
218.             * @param string $script
219.             * @return bool
220.             */
221.             public function validScript($script) {
222.                 if (is_null($this->code)) {
223.                     $msg = Language::getMessage('code_reader', 'error_no_code');
224.                     Log::message($msg, 2);
225.                     throw new Exception($msg, 34);
226.                 }
227.
228.                 $this->script = $script;
229.
230.                 //Gera as linhas de comando
231.                 Log::message(Language::getMessage('code_reader', 'parse_script'), 2);

232.                 $this->currentLine = 0;
233.                 $this->lines = $this->explodeLines();
234.                 Log::message(Language::getMessage('code_reader', 'end_parse_script',
    array('total_lines' => $this->totalLine)), 2);
```

```
235.
236.                 if ($this->totalLine > 0)  {
237.                     $this->timeExecution = microtime(true);
238.                     try {
239.                         Log::message(Language::getMessage('code_reader', 'start_check
    _script'), 2);
240.                         return $this->checkLines($this->lines);
241.                     } catch (CodeReaderException $ex) {
242.                         $this->error['error'] = array($ex->getMessage());
243.                         return false;
244.                     }
245.                 }
246.                 return true;
247.             }
248.             /**
249.             * Checa todas as linhas são válidas
250.             */
251.             private function checkLines($lines) {
252.                 //Tratamento de Condição
253.                 $search = '/\s+(' . $this->code->getAND() . '|' . $this->code-
    >getOR() . ')\s+/'  . ($this->code->isCaseSensitive()? "" :"i");
254.
255.                 foreach ($lines as $currentLine => $line) {
256.                     $this->currentLine = $currentLine;
257.
258.                     $this->checkLimitTimeExecution();
259.
260.                     //IF
261.                     try {
262.                         if ($line['type'] == "IF") {
263.                             //Exec IF
264.                             //Checa condição
265.                             $commands = (preg_split($search, $line['condition'], -
    1, PREG_SPLIT_OFFSET_CAPTURE)); //Lista de comandos na condição
266.                             foreach ($commands as $subcondition)
267.                                 $command = $this-
    >getCommand(trim($subcondition[0]));
268.
269.                             //Checa conteúdo interno do IF
270.                             if (!$this->checkLines($line['statements']))
271.                                 return false;  //ERROR
272.
273.                             $this->currentLine = $currentLine;
274.                         }
275.
276.                         //else
277.                         if ($line['type'] == "ELSE") {
278.                             if (!$this->checkLines($line['statements']))
279.                                 return false;  //ERROR
280.
281.                             $this->currentLine = $currentLine;
282.                         }
283.
284.                         //WHILE
285.                         if ($line['type'] == "WHILE") {
286.                             //Checa condição
287.                             $commands = (preg_split($search, $line['condition'], -
    1, PREG_SPLIT_OFFSET_CAPTURE)); //Lista de comandos na condição
288.                             foreach ($commands as $subcondition)
289.                                 $command = $this-
    >getCommand(trim($subcondition[0]));
290.
291.                             //Checa conteúdo interno do WHILE
292.                             if (!$this->checkLines($line['statements']))
293.                                 return false;  //ERROR
294.
```

```
295.                             $this->currentLine = $currentLine;
296.                             continue;
297.                         }
298.
299.                         //COMMAND
300.                         if ($line['type'] == "COMMAND")
301.                             $command = $this->getCommand($line['line']);
302.
303.
304.                     } catch (CodeReaderException $ex) {
305.                         if ($ex->getCode() == 2) {
306.                             $this->error = array(
307.                                 'error'     => $ex->getMessage()
308.                             );
309.                         } else {
310.                             $this->error = array(
311.                                 'line'      => $currentLine,
312.                                 'code'      => $line['line'],
313.                                 'error'     => $ex->getMessage()
314.                             );
315.                         }
316.
317.                         return false;
318.                     }
319.                 }
320.                 return true;
321.             }
322.
323.
324.             /**
325.             * Executa o código do usuário
326.             * @param string $script
327.             * @return bool
328.             */
329.             public function runScript($script) {
330.                 if (is_null($this->code)) {
331.                     $msg = Language::getMessage('code_reader', 'error_no_code');
332.                     Log::message($msg, 2);
333.                     throw new Exception($msg, 34);
334.                 }
335.
336.
337.                 $this->script = $script;
338.
339.                 //Gera as linhas de comando
340.                 Log::message(Language::getMessage('code_reader', 'parse_script'), 2);

341.                 $this->currentLine = 0;
342.                 $this->lines = $this->explodeLines();
343.                 Log::message(Language::getMessage('code_reader', 'end_parse_script',
    array('total_lines' => $this->totalLine)), 2);
344.
345.                 //Seleciona a primeira linha
346.
347.                 if ($this->totalLine > 0)  {
348.                     $this->timeExecution = microtime(true);
349.                     try {
350.                         Log::message(Language::getMessage('code_reader', 'start_run_s
    cript'), 2);
351.                         return $this->runLines($this->lines);
352.                     } catch (CodeReaderException $ex) {
353.                         $this->error['error'] = array($ex->getMessage());
354.                         return false;
355.                     }
356.
357.                 }
```

```
358.
359.                 return true;
360.             }
361.
362.             /**
363.             * Executa as linhas do Script
364.             * @access private
365.             * @param array $lines
366.             * @return bool
367.             */
368.             private function runLines($lines) {
369.                 foreach ($lines as $currentLine => $line) {
370.
371.                     $this->checkLimitTimeExecution();
372.
373.                     $this->currentLine = $currentLine;
374.
375.                     //IF
376.                     try {
377.                         if ($line['type'] == "IF") {
378.                             Log::message(Language::getMessage('code_reader', 'run_lin
    e', array('code' => $line['line'], 'line' => $this->getCurrentLine())), 2);
379.                             //Exec IF
380.                             if ($this->checkCondition($line['condition'])) {
381.
382.                                 if (!$this->runLines($line['statements']))
383.                                     return false;  //ERROR
384.
385.                                 $this->currentLine = $currentLine;
386.                             } else {
387.
388.                                 //CHECK ELSE
389.                                 $nextKey = (array_search($currentLine, array_keys($li
    nes))+1);
390.                                 $nextLine = array_keys($lines);
391.                                 if (!isset($nextLine[$nextKey]))
392.                                     continue;
393.                                 $this->currentLine = $nextLine[$nextKey];
394.                                 $line = $lines[$this->currentLine];
395.
396.                                 Log::message(Language::getMessage('code_reader', 'run
    _line', array('code' => $line['line'], 'line' => $this->getCurrentLine())), 2);
397.                                 //Exec ELSE
398.                                 if (!emptyempty($line) && $line['type'] == "ELSE") {

399.                                     $this->currentLine = $currentLine;
400.
401.                                     if (!$this->runLines($line['statements']))
402.                                         return false; //ERROR
403.                                     $this->currentLine = $currentLine;
404.                                 }
405.
406.                                 $this->currentLine = $currentLine;
407.                             }
408.                             continue;
409.                         }
410.
411.                         //WHILE
412.                         if ($line['type'] == "WHILE") {
413.                             Log::message(Language::getMessage('code_reader', 'run_lin
    e', array('code' => $line['line'], 'line' => $this->getCurrentLine())), 2);
414.                             while ($this->checkCondition($line['condition'])) {
415.                                 if (!$this->runLines($line['statements']))
416.                                     return false; //ERROR
417.
418.                                 $this->currentLine = $currentLine;
```

```
419.                                 Log::message(Language::getMessage('code_reader', 'run
    _line', array('code' => $line['line'], 'line' => $this->getCurrentLine())), 2);
420.                             }
421.                             continue;
422.                         }
423.
424.                         //END IF/WHILE
425.                         if (in_array($line['type'], array("END-IF", "END-
    WHILE"))) {
426.                             Log::message(Language::getMessage('code_reader', 'run_lin
    e', array('code' => $line['line'], 'line' => $this->getCurrentLine())), 2);
427.                         }
428.
429.                         //COMMAND
430.                         if ($line['type'] == "COMMAND") {
431.                             Log::message(Language::getMessage('code_reader', 'run_lin
    e', array('code' => $line['line'], 'line' => $this->getCurrentLine())), 2);
432.
433.                             //Recupera comando
434.                             $command = $this->getCommand($line['line']);
435.                             $this->execCommand($command);
436.                         }
437.
438.                     } catch (CodeReaderException $ex) {
439.                         if ($ex->getCode() == 2) {
440.                             $this->error = array(
441.                                 'error'     => $ex->getMessage()
442.                             );
443.                         } else {
444.                             $this->error = array(
445.                                 'line'      => $this->currentLine,
446.                                 'code'      => $line['line'],
447.                                 'error'     => $ex->getMessage()
448.                             );
449.                         }
450.
451.                         return false;
452.                     }
453.                 }
454.
455.                 return true;
456.             }
457.
458.             /**
459.             * Verifica se uma condição retorna true ou false
460.             * @access private
461.             * @param string $condition
462.             * @return bool;
463.             */
464.             private function checkCondition($condition) {
465.                 $search = '/\s+(' . $this->code->getAND() . '|' . $this->code-
    >getOR() . ')\s+/'  . ($this->code->isCaseSensitive()? "" :"i");
466.
467.                 $return = null;     //Condição que será retornada
468.                 $position = 0;      //Posição atual da condição na String
469.                 $commands = (preg_split($search, $condition, -
    1, PREG_SPLIT_OFFSET_CAPTURE)); //Lista de comandos na condição
470.
471.                 foreach ($commands as $subcondition) {
472.
473.                     //Executa o comando
474.                     $command = $this->getCommand(trim($subcondition[0]));
475.                     $result = (bool)$this->execCommand($command);
476.
477.                     //Atribui o valor que será retornado
478.                     if (is_null($return))
```

```
479.                         $return = $result;
480.                     else {
481.                         //Caso exista mais de uma condição, verifica se a ação é de A
    ND ou OR
482.                         $operator = substr($condition, $position, $subcondition[1] -
     $position);
483.                         $position = strlen($subcondition[0]);
484.
485.                         //AND
486.                         if (preg_match('/('.$this->code->getAND().')/'  . ($this-
    >code->isCaseSensitive()? "" :"i"), $operator))     //AND
487.                         $return = ($return && $result);
488.
489.                         //OR
490.                         elseif (preg_match('/('.$this->code->getOR().')/'  . ($this-
    >code->isCaseSensitive()? "" :"i"), $operator))  //OR
491.                         $return = ($return || $result);
492.                     }
493.                 }
494.                 Log::message(Language::getMessage('code_reader', 'check_condition', a
    rray('line' => $this-
    >getCurrentLine(), 'condition' => $condition, 'return' => ($return? "true" : "false"
    ))), 2);
495.                 return (bool)$return;
496.             }
497.
498.             /**
499.             * Recupera o método e os parâmetros de uma linha de comando
500.             * @access private
501.             * @param string $line
502.             * @return array (command)
503.             */
504.             private function getCommand($line) {
505.
506.                 $commandList = $this->code->getCommands();
507.                 $commandLine = false;
508.                 foreach ($commandList as $command => $method) {
509.                     $command = WordsUtil::convertToRegex($command, $this->code-
    >isCaseSensitive());
510.
511.                     if (preg_match($command, $line, $match)) {
512.
513.                         $param = array();
514.                         if (count($match) > 1)
515.                             $param = array_slice($match, 1);
516.
517.                         $commandLine = array(
518.                             'method'    => $method,
519.                             'param'     => $param
520.                         );
521.                         break;
522.                     }
523.                 }
524.
525.                 if ($commandLine == false)
526.                     throw new CodeReaderException(Language::getMessage('code_reader',
     'code_not_found', array('line' => $line, 'current_line' => $this-
    >getCurrentLine())));
527.
528.                 return $commandLine;
529.             }
530.
531.             /**
532.             * Executa o comando que o usuário digitou
533.             * @access private
534.             * @param array $command
```

```
535.          * @return bool;
536.          */
537.          private function execCommand($command) {
538.              if (method_exists($this->code, $command['method'])) {
539.                  Log::message(Language::getMessage('code_reader', 'run_command', a
    rray('method' => $command['method'], 'param' => json_encode($command['param']))), 2)
    ;
540.
541.                  if (!emptyempty($command['param']))
542.                      return call_user_func_array(array($this-
    >code, $command['method']), $command['param']);
543.                  else
544.                      return call_user_func(array($this-
    >code, $command['method']));
545.              }
546.          }
547.
548.          /**
549.          * faz o controle do tempo máximo de execução
550.          * @access private
551.          */
552.          private function checkLimitTimeExecution() {
553.              $time = microtime(true);
554.              $time = round(($time - $this->timeExecution), 4);
555.              if ($time > $this->code->maxTimeExecution())
556.                  throw new CodeReaderException(Language::getMessage('code_reader',
     'max_time_execution'), 2);
557.
558.          }
559.          /****************** GETTERS E SETTERS ***********************/
560.          /**
561.          * Informa o código que será verificado
562.          * @param string $codeName
563.          */
564.          public function setRules($codeName) {
565.              $codeName = ucfirst($codeName);
566.
567.              if (file_exists(LIBRARIES_PATH.'code/'.$codeName.'.php'))
568.                  require_once(LIBRARIES_PATH.'code/'.$codeName.'.php');
569.              elseif(file_exists(LIBRARIES_JI_PATH.'code/'.$codeName.'.php'))
570.                  require_once(LIBRARIES_JI_PATH.'code/'.$codeName.'.php');
571.              else {
572.                  $msg = Language::getMessage('code_reader', 'error_rules', array('
    code_name' => $codeName));
573.                  Log::message($msg, 2);
574.                  throw new Exception($msg, 32);
575.              }
576.
577.              $code = new $codeName;
578.
579.              $this->setCode($code);
580.              $this->codeName = $codeName;
581.          }
582.
583.          /**
584.          * Seta o código que será interpretado na execução do script
585.          * @param ICode $code
586.          */
587.          public function setCode($code) {
588.              if ($code instanceof JIndie\Code\ICode)
589.                  $this->code = $code;
590.              else {
591.                  $msg = Language::getMessage('code_reader', 'error_icode');
592.                  Log::message($msg, 2);
593.                  throw new Exception($msg, 33);
594.              }
```

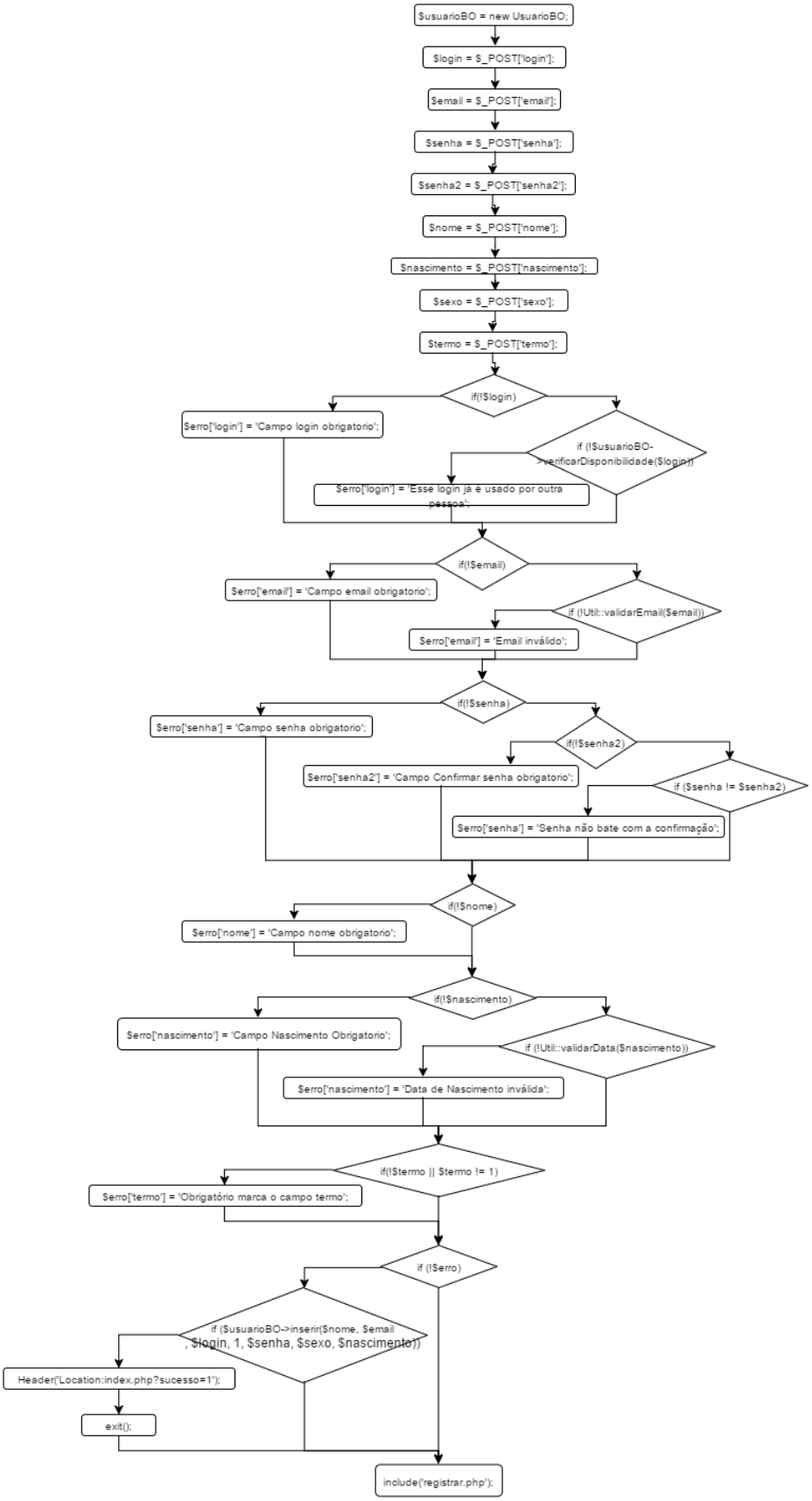
```
595.            }
596.            /**
597.            * Retorna o ICode usado
598.            * @return ICode
599.            */
600.            public function getCode() {
601.                return $this->code;
602.            }
603.
604.            /**
605.            * Retorna o erro no script do usuário
606.            * @return array;
607.            */
608.            public function getError() {
609.                return $this->error;
610.            }
611.
612.            /**
613.            * Retorna a ultima linha executada
614.            * @return int
615.            */
616.            public function getCurrentLine() {
617.                return $this->currentLine;
618.            }
619.
620.            /**
621.            * Retorna o total de linhas geradas no script
622.            * @return int
623.            */
624.            public function getTotalLine() {
625.                return $this->totalLine;
626.            }
627.
628.            /**
629.            * Retorna as linhas geradas no script
630.            * @return array
631.            */
632.            public function getLines() {
633.                return $this->lines;
634.            }
635.        }
```

APÊNDICE I – Código da Interface dos Artefatos

```php
<?php
/**
*   JIndie
*   @package JIndie
*   @subpackage Game
*   @category Components of Game
*   @author Carlos W. Gama <carloswgama@gmail.com>
*   @copyright Copyright (c) 2015
*   @license   http://opensource.org/licenses/gpl-3.0.html GNU Public License
*   @version   1.0
*/

namespace JIndie\Game;

interface IArtefact  {

    /**
    * Retorna o identificador do Artefato
    * @return int
    */
    public function getId();
    /**
    * Seta o identificador do Artefato
    * @param int $id
    */
    public function setId($id);


    /**
    * Retorna o nome do artefato
    * @return string
    */
    public function getName();
    /**
    * Seta o nome do artefato
    * @param string $name
    */
    public function setName($name);


    /**
    * Retorna a descrição do artefato
    * @return string
    */
    public function getDescription();
    /**
    * Seta a descrilção do artefato
    * @param string $description
    */
    public function setDescription($description);


    /**
    * Seta o Status do Artefato como iniciado
    */
    public function start();
    /**
    * Retorna o Status do Artefato
    * @return mix
    */
    public function getStatus();
    /**
    * Seta o Status do Artefato
```

```
64.     * @param mix $status
65.     */
66.     public function setStatus($status);
67.     /**
68.     * Seta o Status do Artefato como concluído
69.     */
70.     public function complete();
71.
72.
73.     /**
74.     * Recupera a lista de componentes do Artefato
75.     * @return array of IComponents
76.     */
77.     public function getComponents();
78.     /**
79.     * Adicionar um componente ao Artefato
80.     * @param IComponents $component
81.     */
82.     public function addComponent($component);
83.     /**
84.     * Deleta um componente do Artefato
85.     * @param IComponents $component
86.     */
87.     public function deleteComponent($component);
88.     /**
89.     * Deleta todos componentes do artefato
90.     */
91.     public function clearComponents();
92.
93. }
```

APÊNDICE J – Representação visual do código de cadastro de usuários do jogo Sim Investigador sem LPS



APÊNDICE K – Representação visual do código de cadastro de usuários do jogo Sim Investigador após uso da LPS JIndie