6
Construindo o SimulES-W

6.1.Background

Neste capítulo detalharemos o método de desenvolvimento proposto para o SimulES-W uma implementação em plataforma Web do jogo antes só disponível na versão do tabuleiro. Como temos apresentado ao longo desta dissertação varias etapas tem acontecido até chegar neste ponto. Nas primeiras etapas para alcançar o entendimento do contexto do SimulES foram analisadas as diferentes fontes de informação, algumas já mencionadas nesta dissertação, a monografia [4] e o trabalho de dissertação [44], além das diferentes versões de léxicos e cenários disponíveis em [22] produto das evoluções do jogo.

Lembremos que para obter um entendimento razoável da dinâmica do jogo, foram analisados os elementos físicos criados e apresentados no trabalho [4], além disso, foram planejadas reuniões no grupo de Engenharia de Requisitos da PUC-Rio para revisar o conteúdo das cartas conceito e problema. Depois, varias sessões para jogar SimulES foram programadas para receber treinamento pratico do jogo e fortalecer os conceitos de *rodadas* e *jogadas possíveis* no jogo.

Visando uma evolução do SimulES incluindo a implementação do mesmo, partimos na procura de literatura sobre jogos educacionais na engenharia de software (Capítulo 2), com o foco de entender o objetivo de cada um e identificar os métodos usados para suas modelagens e posteriores implementações. Identificamos nesta parte do trabalho que as descrições sobre as modelagens eram vagas ou inexistentes. Identificamos que a iteração entre usuários não era levada em consideração na literatura pesquisada, mas identificamos que a modelagem intencional usada para modelagem de processos organizacionais levava em consideração iteração entre usuários. Porem, nós propusemos trabalhar no entendimento da modelagem intencional, além disso, vimos que nenhum dos jogos pesquisados usava a modelagem intencional como insumo para implementação dos jogos.

Acreditávamos na potencialidade da modelagem intencional para representação do jogo e como insumo útil para afrontar uma implementação, é assim, que para facilitar a criação dos modelos iniciais foi usado o método *ERi*c*, modelos que foram apresentados e validados segundo a proposta de validação de modelos *i** proposta em [44]. Estes modelos foram indispensáveis na implementação de SimulES-W.

Como precisávamos saber se a modelagem que estávamos usando era adequada e suficiente para abordar uma implementação, buscamos literatura sobre experiências de modelagem intencional que derivaram na criação de algum protótipo. Encontramos algumas propostas, as quais são descritas no Capítulo 5 e que aportaram idéias importantes para esta dissertação. Além disso, procuramos literatura relacionada com a transparência de software que nos auxiliará na procura de conceitos sobre como combinar a nossa modelagem com atributos de transparência de software e que estes elementos fossem refletidos nossa implementação.

O protótipo começou a ser desenvolvido e suas primeiras funcionalidades foram apresentadas como Projeto Final de Programação, na medida em que o protótipo foi desenvolvido, os modelos foram sendo refinados. Tivemos a oportunidade de validar e incorporar novo conhecimento a nosso trabalho a partir de uma experiência real com o jogo de mesa e estudantes na UERJ, estudantes que pertencia à área de geomática. Destes estudantes alguns não tinham conhecimento sobre um processo de desenvolvimento de software embora trabalhassem em áreas afins à informática (Capítulo 3). Precisávamos de um grupo desta natureza que fosse imparcial. Pois, as sessões ou reuniões anteriores para jogar o SimulES foram realizadas com grupos da área da comparação.

O resultado destas atividades derivaram em uma nova versão do SimulES que incorpora uma versão digital chamada SimulES-W a qual será detalhada a seguir.

6.2.Arquitetura do SimulES-W

SimulES-W como uma aplicação Web:

O protótipo de SimulES é uma aplicação Web que utiliza software livre. O foco deste tipo de aplicações é proporcionar valor real e oferecer uma experiência positiva para o usuário ou jogador.

As vantagens das aplicações Web é que estas não exigem compra física, não precisam downloads, instalações e configurações, funcionam diretamente em qualquer computador sendo mais confiáveis que outras aplicações disponíveis para download. Além disso, as páginas Web são usadas como as interfaces de usuário. Estas vantagens foram as motivações pelas quais foi escolhida este tipo de plataforma para nossa implementação.

O SimulES-W tem um desenho amigável, simples e fácil de usar, cumprindo com as características de uma boa aplicação Web. Além disso, procura implementar requisitos não funcionais de transparência. Estes atributos serão analisados desde a perspectiva **Transparência de Software** mais adiante neste documento.

Ferramentas usadas:

Como linguagem de programação foi escolhida a linguagem Java [60]. A primeira vantagem do Java é a independência de plataforma e facilidade de acesso para os usuários por ser de fonte aberta. Java é orientada a objetos e foi projetada para servir como uma nova forma de gerir a complexidade de software.

Além disso, precisávamos de uma base de dados para gerenciar as informações utilizadas numa sessão do jogo. Nossa escolha foi por MySQL, conforme [61] MySQL tornou-se a base de dados de código aberto mais popular do mundo principalmente em desenvolvimentos Web.

Padrões e Estilos:

O padrão de desenho escolhido foi o MVC (*Model-View-Controller*) Este padrão é usado para separar a lógica de negocio, a interface e o controle. É baseado nas boa práticas sugeridas pela *Sun MicroSystems* [60] para o uso do *framework Hibernate* [64]. No trabalho de Almentero [59] também se faz uso deste padrão em conjunto com programação modular. Nosso caso é diferente, pois

a arquitetura é baseada em orientação a objetos, do trabalho de Almentero nós adotamos a descrição baseada em cenários proposta para a descrição de código.

Na Figura 46 se representa a arquitetura, que desacopla a vista do modelo, com a finalidade de melhorar a re-usabilidade. É desta forma que as modificações nas vistas impactam em menor medida a lógica de negocio ou de dados. Os elementos do padrão são: o **Modelo** que contem dados e regras de negocio, a **Vista** que apresenta a informação do modelo para o usuário e o **Controlador** que gerencia as entradas do usuário.

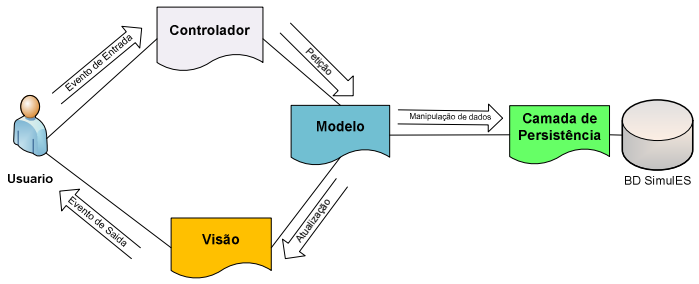


Figura 46 – Arquitetura projetada para o SimulES-W.

A idéia de usar este padrão na implementação do protótipo de SimulES é que o modelo possa ter diversas vistas, cada um com seu correspondente controlador. As responsabilidades identificadas são:

- Modelo: será o encarregado de acessar à camada de persistência e implantar as regras de negocio (funcionalidade do sistema), levar um registro das vistas e controles do sistema.

- Controlador: será o encarregado de receber os eventos de entrada, além de conter as regras de gestão dos eventos.

- Vistas: encarregadas de receber os dados do modelo e mostrar-los aos usuários.

6.3.Método Usado na Construção do SimulES -W

6.3.1. Primeira Etapa

Para a modelagem do SimulES-W foram analisados em conjunto: Léxicos, Cenários, Diagrama SDsituations, Modelo SA e os Diagramas SD e Diagramas SR. Com estes elementos construímos o primeiro diagrama de classes do Jogo.

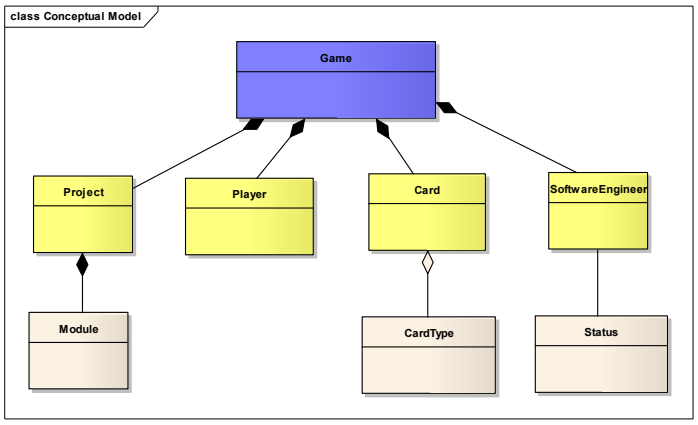


Figura 47 – Primeira versão do modelo de classes para SimulES-W.

Na Figura 47 representamos as primeiras classes que foram identificadas para o jogo, para isso utilizamos a seguinte heurística, todos os símbolos do LAL foram analisados prestando especial atenção aqueles símbolos tipo objeto, aqueles que reuniram as propriedades básicas de uma classe tais como:

- As classes são descritas por substantivos: então analisamos os símbolos nomeados por substantivos.

- Atributos são propriedades nas classes: alguns símbolos possuíam descrições que nos indicavam as suas propriedades. Como na Figura 47 vemos que o símbolo **Projeto** tem características de complexidade, tamanho, qualidade e orçamento candidatos a serem propriedades da classe.

- Classes são susceptíveis de ter operações: que na sua vez são serviços prestados pela classe quando um objeto é solicitado para modificar seu comportamento. Identificamos que os impactos dos símbolos nos auxiliariam nesta tarefa.

- Além disso, identificamos que símbolos tipo verbo podiam representar operações das classes. Mais adiante utilizaremos esta heurística para identificar comportamentos das classes.

Nome:	cartão de projeto
Noção:	Retângulo em papel que informa detalhes sobre o projeto a ser desenvolvido no **jogo**. Um cartão de projeto contém uma descrição em linguagem natural com as principais características do projeto, a **complexidade do projeto**, o **tamanho do projeto**, a **qualidade do projeto** e o **orçamento do projeto**. Um cartão de projeto deve ser selecionado por um **jogador** para ser desenvolvido ao longo do **jogo**. O cartão de projeto selecionado deve ser colocado sobre a **mesa** de forma que todos os **jogadores** tenham acesso as suas informações. A escolha do cartão de projeto é um pré-requisito para dar início ao **jogo**.
Classificação:	objeto
Impacto(s):	1-**jogador** escolhe cartão de projeto que será considerado ao longo do **jogo**. 2-**adversario** concorda com a escolha do projeto. 3-**jogador** obedece as especificações do cartão de projeto.
Sinônimo(s):	cartão, cartões de projeto, cartões, projeto.

Figura 48 – Símbolo SimulES analisado para sua conversão a classe do sistema.

Se bem o léxico foi nossa primeira fonte de informação para identificar candidatos a classe, foi a análise destes símbolos nos diagramas SR e SD o que nos afiançou na nossa premissa, pois estes símbolos representados na modelagem intencional como recursos cumpriam as características de uma classe.

6.3.2.Segunda Etapa

Tendo a primeira versão do modelo do SimulES (Figura 47) e visando o SimulES como uma aplicação Web passamos a analisar as *SDsituations* (Figura 27) do jogo, estas SDSituations descritas tanto nos diagramas SD e SR do trabalho [44] possuíam uma descrição em cenários. Em nossa análise identificamos que cada uma delas podia ser representada em uma tela do sistema (página Web), então criamos a primeira versão do modelo conceitual da aplicação Web (Figura 49). A página principal ou *Main* foi identificada como um ponto de entrada para todos os jogadores e a página *gestão do jogo* seria uma página para gerir elementos de controle que somente um administrador podia manipular. Já as páginas *Rodada de inicio, Rodada de Ações e Rodada de conceitos* que pertenciam às *SDsituations* seriam denominadas como as páginas núcleo do jogo.

Como vemos na Figura 49, os episódios descrevem comportamentos que são sensíveis a implementação, além disso, cada *SDsituations* está totalmente desacoplada, o que faz razoável a análise de implementar separadamente cada uma delas.

Título:	início de jogo
Objetivo:	Descrever os preparativos para inicio do **jogo**.
Contexto:	Tabuleiro de cada **jogador** colocado na **mesa de jogo**. **cartas** embaralhadas sobre a **mesa**. Cada **jogador** tem uma **carta** de **engenheiro de software**.
Atores:	**jogador**
Recursos:	**dado**, **cartas**, **informações do projeto**, tabuleiro.
Exceção:	
Episódios:	**jogador** joga **dado**. Restrição: **jogador** que tirar o maior número no **dado**, inicia o **jogo**. **jogador** escolhe aleatoriamente do **monte** de **cartões de projeto**. Restrição: Outros **jogadores** têm que concordar com **carta** escolhida por **jogador** que inicia **jogo**. Cada **jogador** compra uma **carta** de **engenheiro de software** e coloca-a no tabuleiro. Restrição: **sentido de jogo** é horário.

Figura 49 – *SDSituations* candidata a página no SimulES-W.

SDSituations candidata a página no SimulES

Com estas heurísticas, o modelo inicial da Figura 46 e o modelo de navegabilidade da aplicação, Figura 50, foram insumos para a criação do primeiro protótipo.

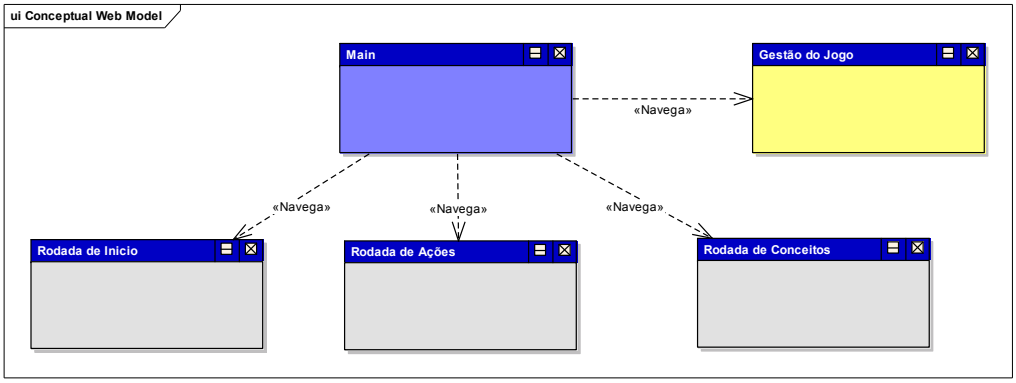


Figura 50 – Modelo de navegabilidade da aplicação Web para o SimulES-W.

6.4. Incorporando Elementos em cada uma das Camadas

Com a implementação das classes identificadas dos recursos e atores na modelagem intencional, que no C&L aparecem como símbolo tipo objeto e sujeito, nós começamos a análise detalhada para a implementação dos comportamentos tanto nas páginas como nas classes. Para isso analisávamos novamente os diagramas SD e SR procurando os comportamentos (intencionalidades) e os objetivos que procuravam cada um dos atores. As informações destes elementos estariam presentes na modelagem como metas e tarefas e seu detalhamento estaria presente no Léxico, o que significa, identificamos que *metas e tarefas* as quais estão representadas no C&L como símbolos tipo *verbo e estado* tinham que ser incorporadas no sistema para dar a este o comportamento esperado e ficariam na camada de controle. As *SDsituations* identificadas foram incorporadas na camada de visão porque possuíam características que nos permitiram implementar-las nesta camada, pois, constituíam cenários bem delimitados com um objetivo identificado, e o cumprimento dos objetivos em cada *SDsituation* era independente das outras. Além disso, possuíam uma correlação em tempo de execução, ou seja, identificamos quais tinham que ser executadas antes e depois. E na medida em que íamos implementando refinamos os modelos *i** e também os léxicos e cenários.

6.4.1.Identificar as *SDsituations* Principais

As *SDsituations* nomeadas como principais foram aquelas que faziam parte do núcleo do jogo ou seja as que estão representadas na Figura 51, as *SDSituations* ressaltadas em amarelo foram aquelas refinadas e implementadas nas primeiras etapas.

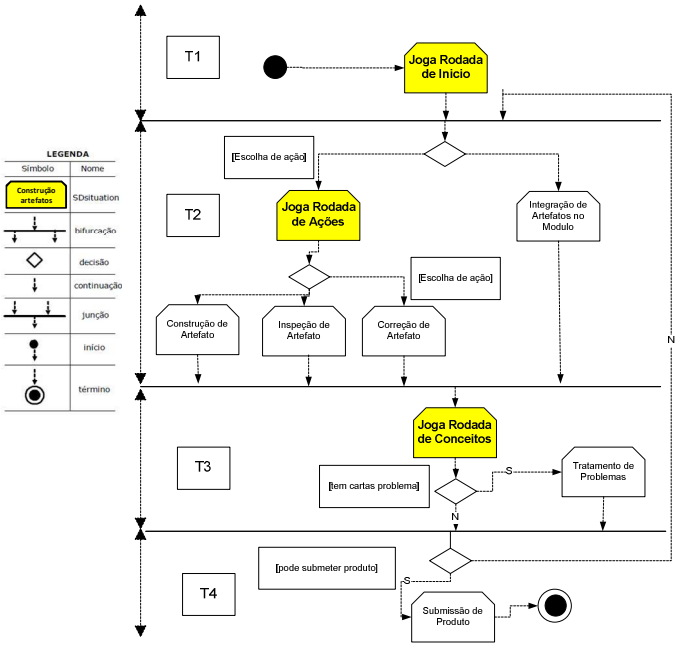


Figura 51 – Diagrama *SDsituations* refinado para o SimulES-W.

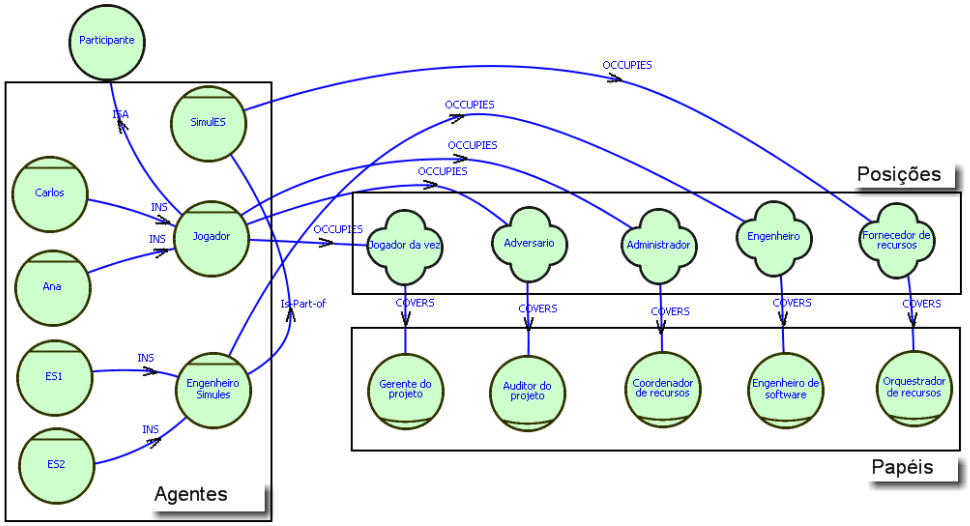


Figura 52 – Modelo SA SimulES-W refinado.

Uma nova posição para jogador foi descoberta *Administrador*, ele é instanciado naqueles casos onde é preciso fazer atividades de gestão dentro do

jogo. Porém, seu papel neste caso particular será de *Coordenador de Recursos* e o SimulES atuara como *Orquestrador* dos mesmos dentro do sistema, na Figura 51 podemos ver estas mudanças dentro do Modelo SA.

A seguir vamos ver o refinamento dos modelos segundo a implementação, alguns dos modelos intencionais apresentados a seguir foram tomados de [44] e refinados conforme a implementação, outros foram criados porque pertencem a SDsituations auxiliares.

SDsituation: Joga Rodada de Início

Nesta SDsituation podemos ver que um novo ator entrou na cena do jogo, o Administrador, que é um jogador que se encarregará de gerir elementos dentro do jogo e solicitar recursos e atividades de controle para o SimulES, Figura 53, vemos que o Administrador é o encarregado de fechar a entrada de jogadores, mas é SimulES como o orquestrador o encarregado de executar a atividade dentro do sistema como vemos na Figura 54. Outra questão importante a ressaltar é a meta *jogador seja registrado*. Para controlar jogadores em uma partida o SimulES deve saber que jogadores estão *registrados*, além disso as atividades e movimentos on-line efetuados pelos jogadores serão publicados para que todos os jogadores fiquem informados sobre detalhes da partida.

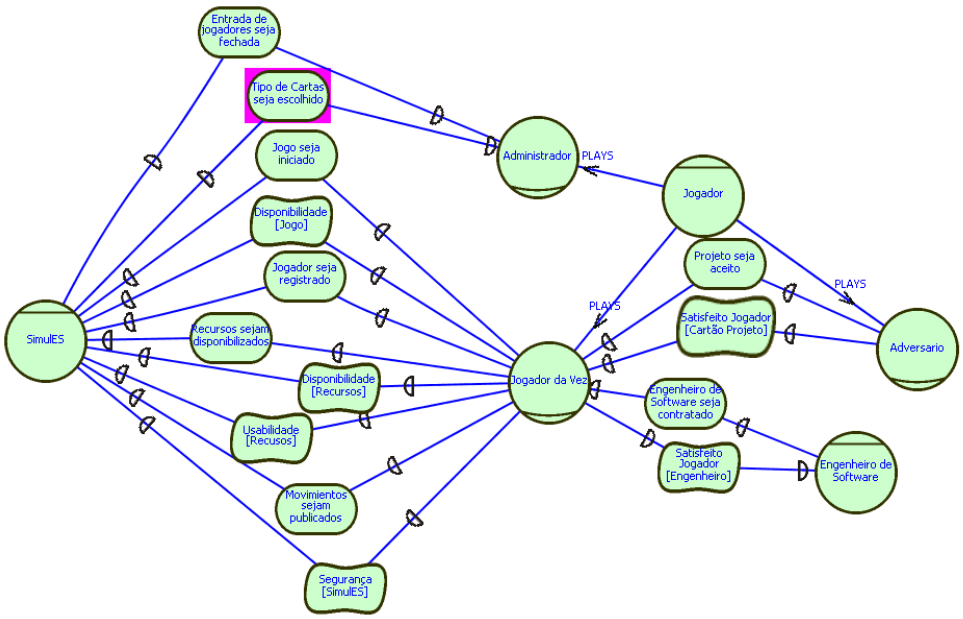


Figura 53 – Modelo SD – Joga Rodada de Início.

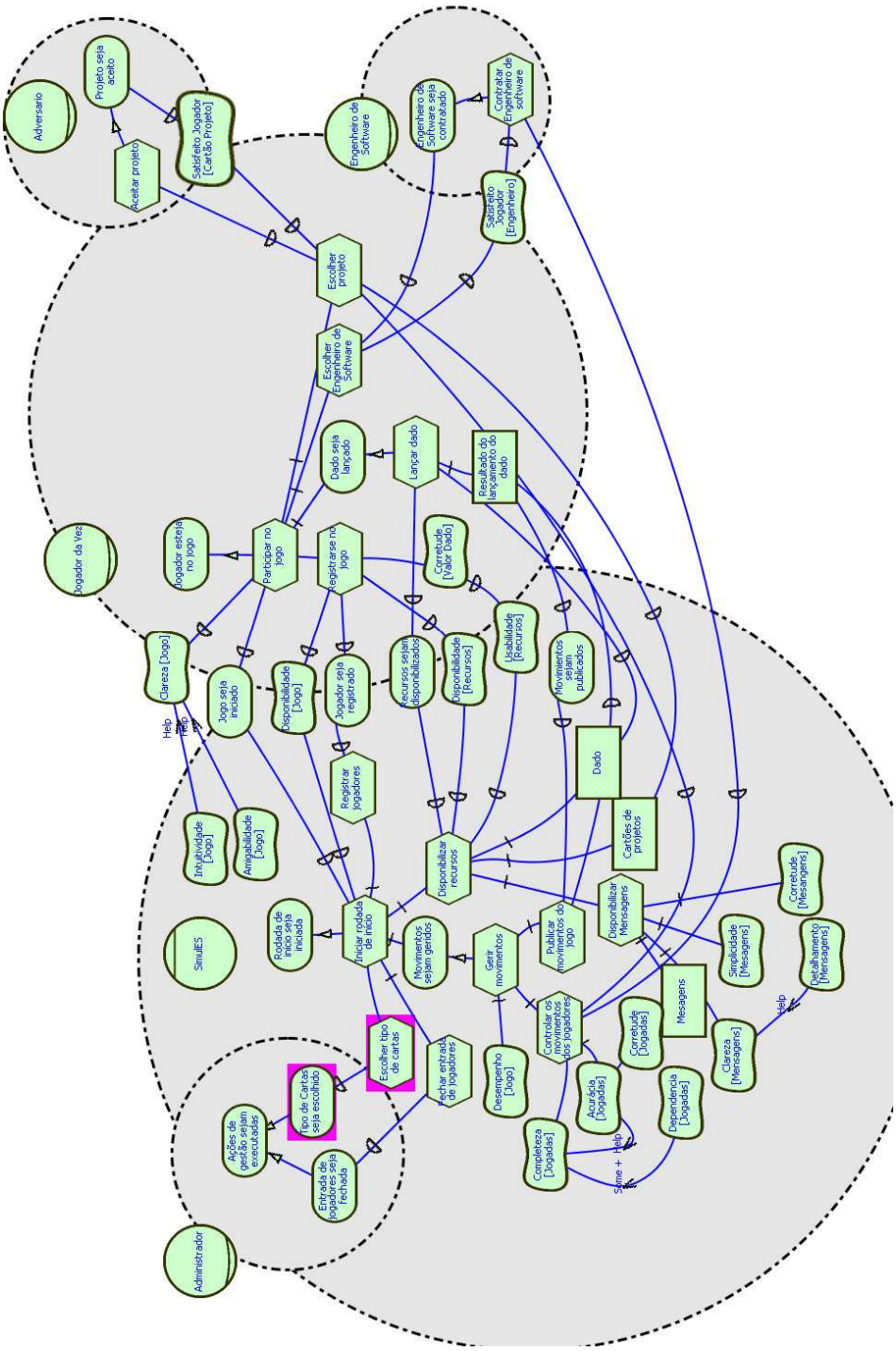


Figura 54 – Modelo SR – Joga Rodada de Início.

SDsituation: Joga Rodada de Ações

Nesta SDsituations temos que o jogador e o SimulES são os atores principais da situação, o engenheiro de software entrará na cena se ações do tabuleiro são escolhidas (Figura 55). O jogador administrador não participa nesta situação, contudo SimulES executará tarefas de controle ao calcular o lançamento do dado e ceder as cartas ao jogador, além de publicar os movimentos e as informações do jogo.

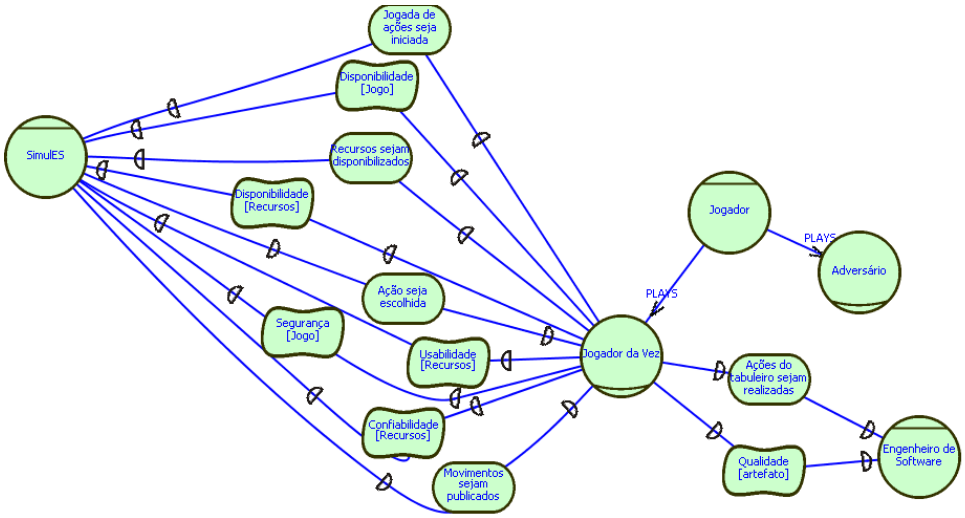


Figura 55 – Modelo SD – Joga Rodada de Ações.

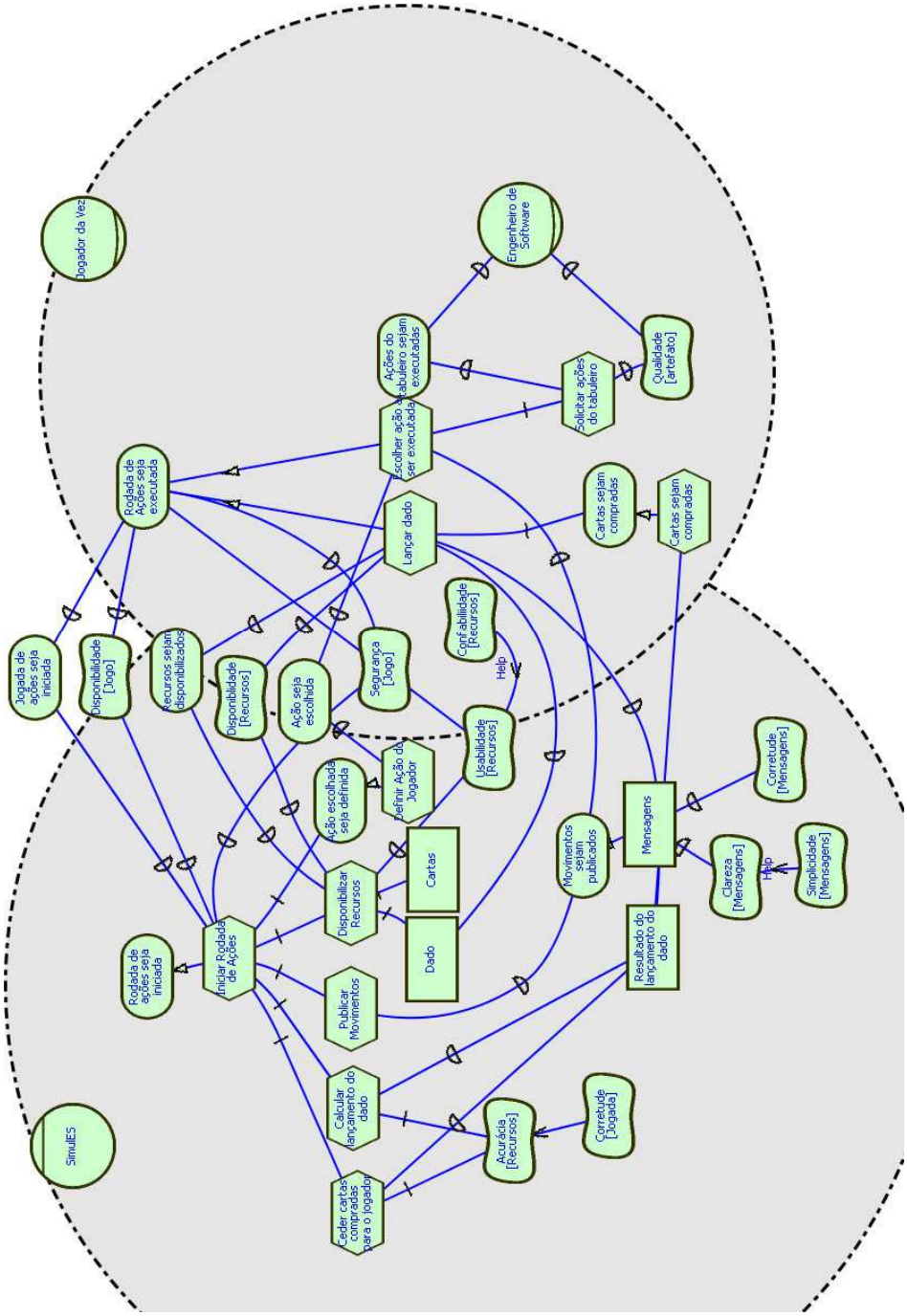


Figura 56 – Modelo SR – Joga Rodada de Ações.

SDsituation: Construção de Artefato

Desta SDsituations podemos resaltar que o jogador, junto com seu engenheiro ou engenheiros, construi os artefatos como vemos na Figura 57 e depois ele os submete. O SimulES se encarregara de guardar a configuração do tabuleiro individual e publica-la depois para ser visto por outros jogadores. Além disso, o SimulES se encarregará de gestionar as mensagens onde apresentará o resumo dos movimentos feitos pelos jogadores (Figura 58).

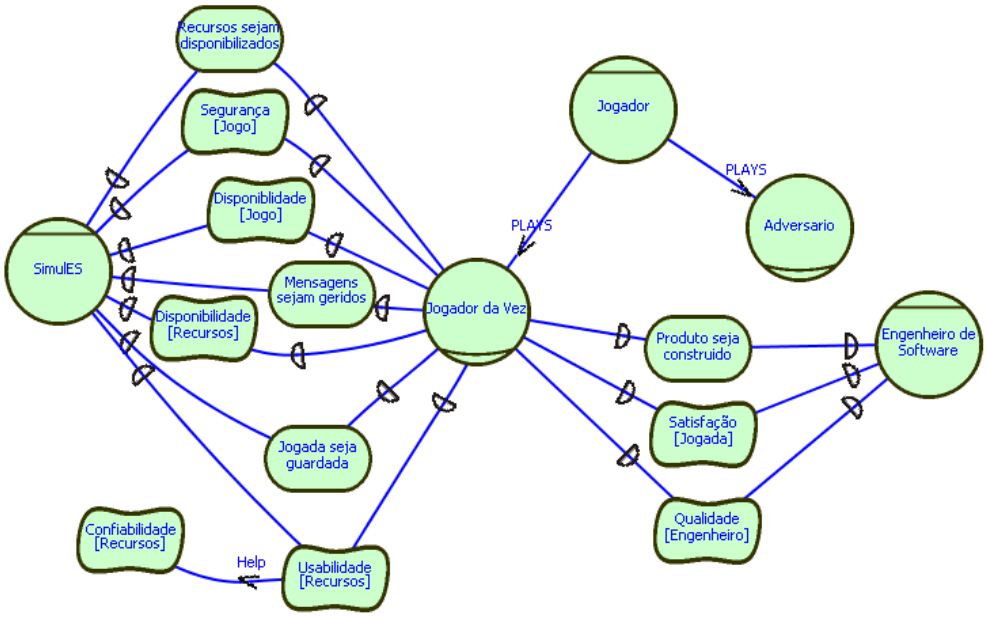


Figura 57 – Modelo SD – Construção de Artefato.

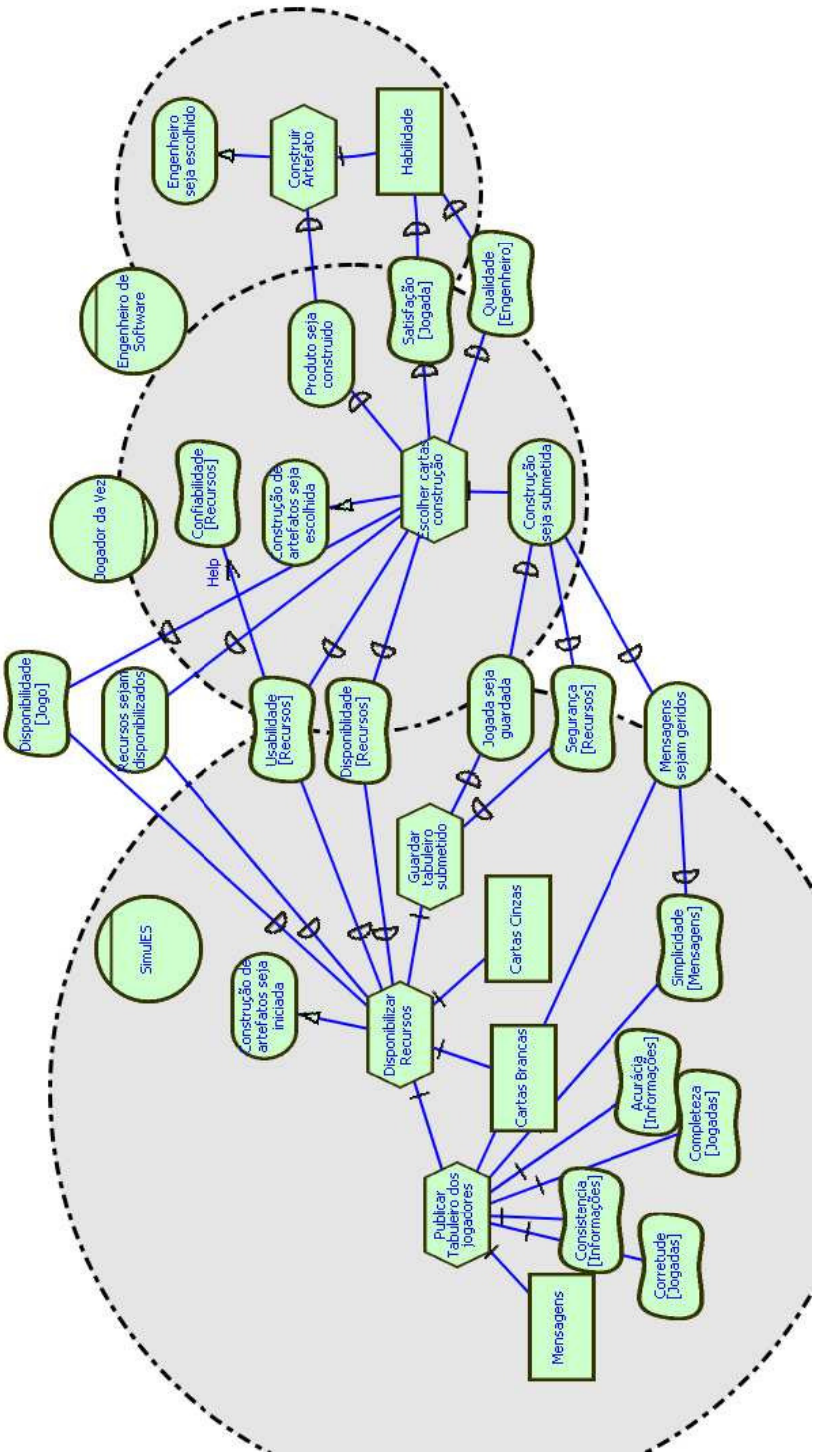


Figura 58 – Modelo SD – Construção de Artefato.

SDsituation: Inspeção de Artefato

Na inspeção de artefato interagem o SimulES, o jogador e o Engenheiro ou Engenheiros de Software. SimulES faz a gestão e fornece os recursos para a inspeção (Figura 59), a inspeção é feita baseada na relação entre o jogador e engenheiro. Já quando a inspeção é finalizada o jogador deve submeter para que o SimulES guarde a configuração (Figura 60) e outros jogadores possam conhecer o resultado da inspeção. Embora algumas metas flexíveis começam a ser identificadas nesta etapa, elas serão tratadas e ou refinadas em etapas posteriores da modelagem com o foco de requisitos não funcionais.

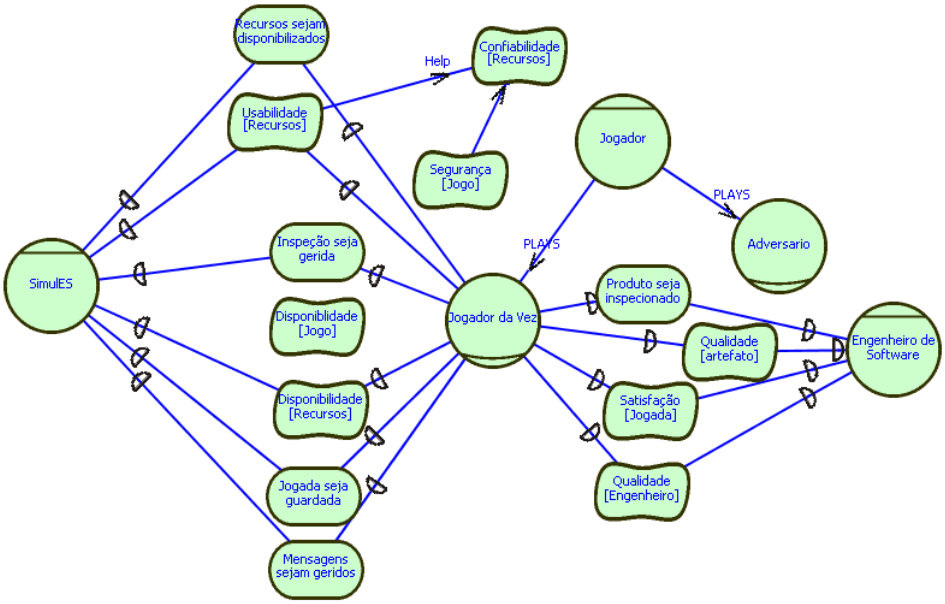


Figura 59 – Modelo SD – Inspeção de Artefato.

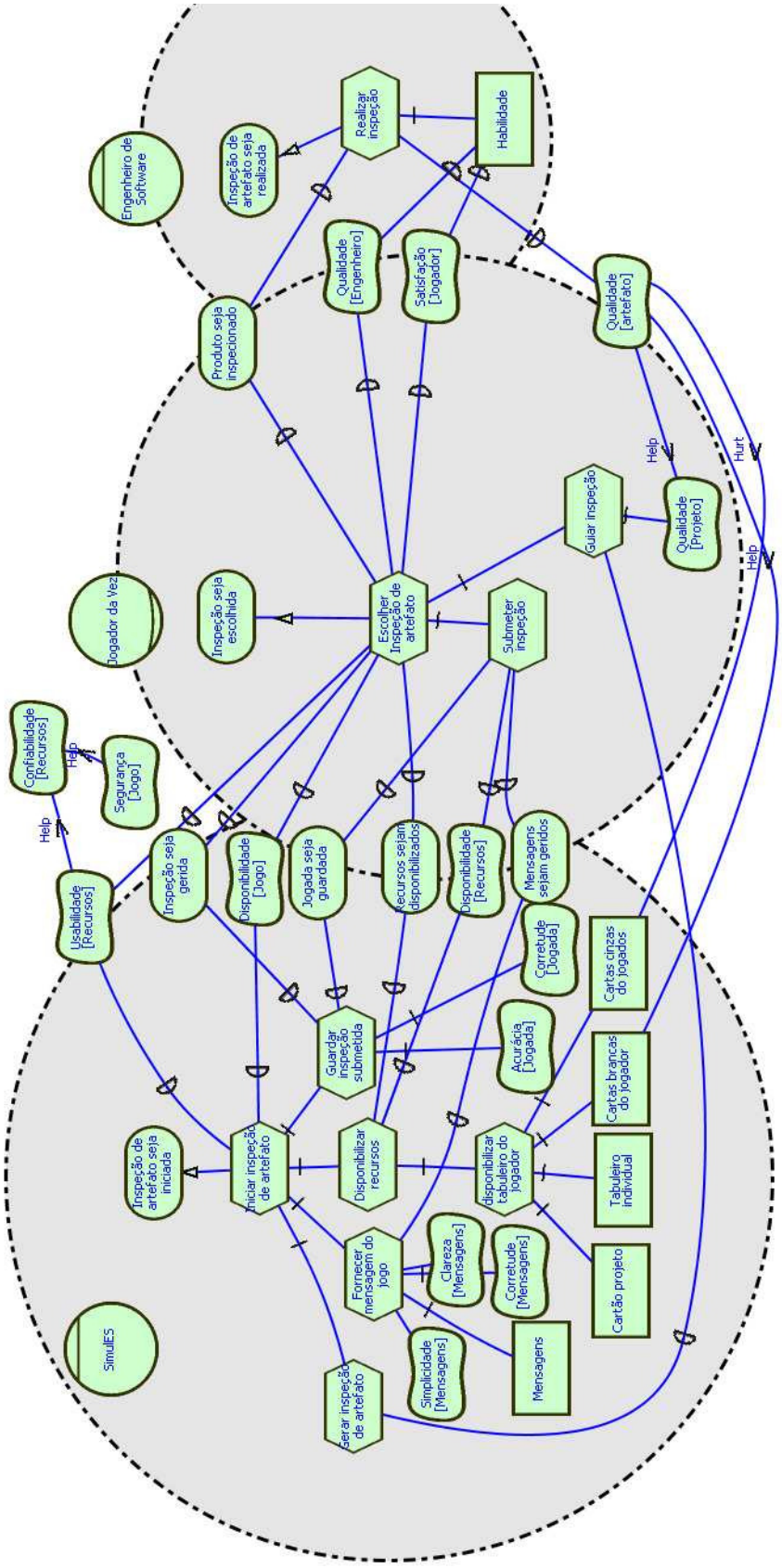


Figura 60 – Modelo SD – Inspeção de Artefato.

SDsituation: Correção de Artefatos

O SimulES deve fornecer o tabuleiro individual com a construção dos artefatos no tabuleiro individual feito previamente pelo jogador. A condição para realizar a correção é que o jogador tenha cartas brancas ou cartas cinzas com defeito e estará condicionado à habilidade dos engenheiros contratados.

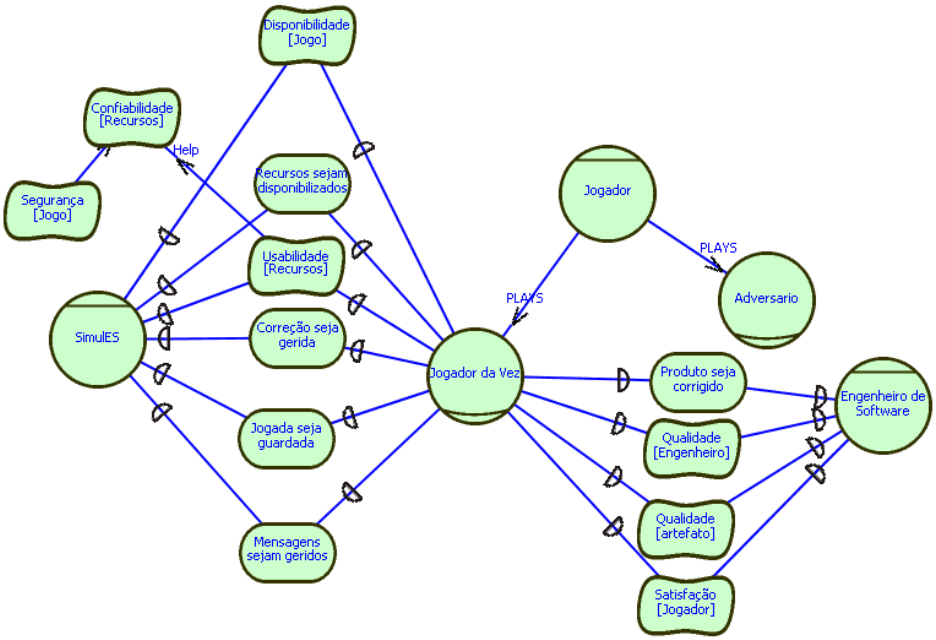


Figura 61 – Modelo SD – Correção de Artefato.

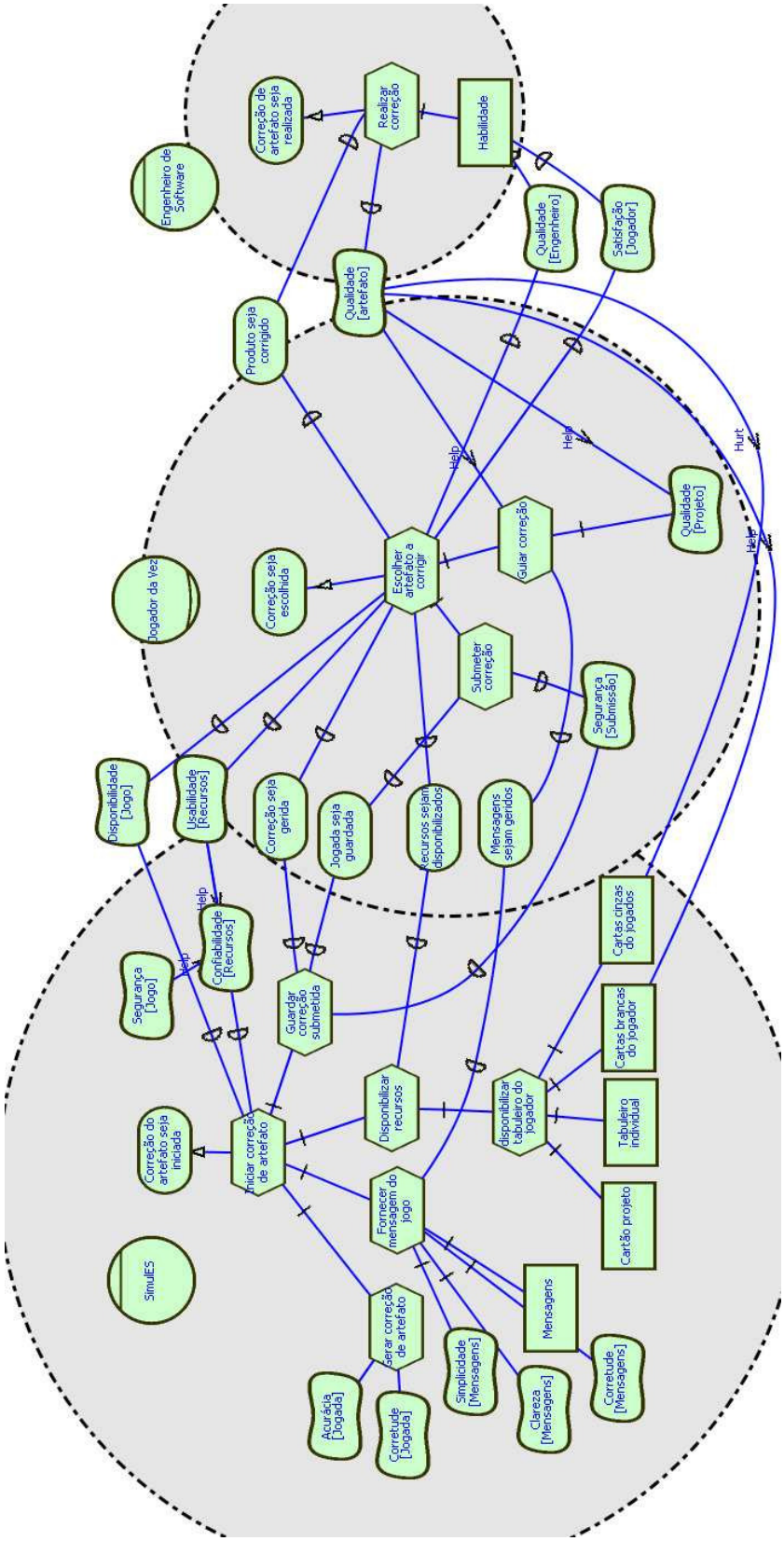


Figura 62 – Modelo SD – Correção de Artefato.

SDsituation: Joga Rodada de Conceitos

Nesta rodada do jogo os jogadores aplicam seus conceitos e jogam problemas a seus adversários como estratégia para ganhar o jogo e danificar o jogo dos outros. Caso não sejam permanentes, o jogador pode livrar-se eventualmente de problemas impostos. Ele pode avaliar a conveniência da participação de seus engenheiros dentro do jogo além de descartar aquelas cartas que não sejam úteis para seu jogo.

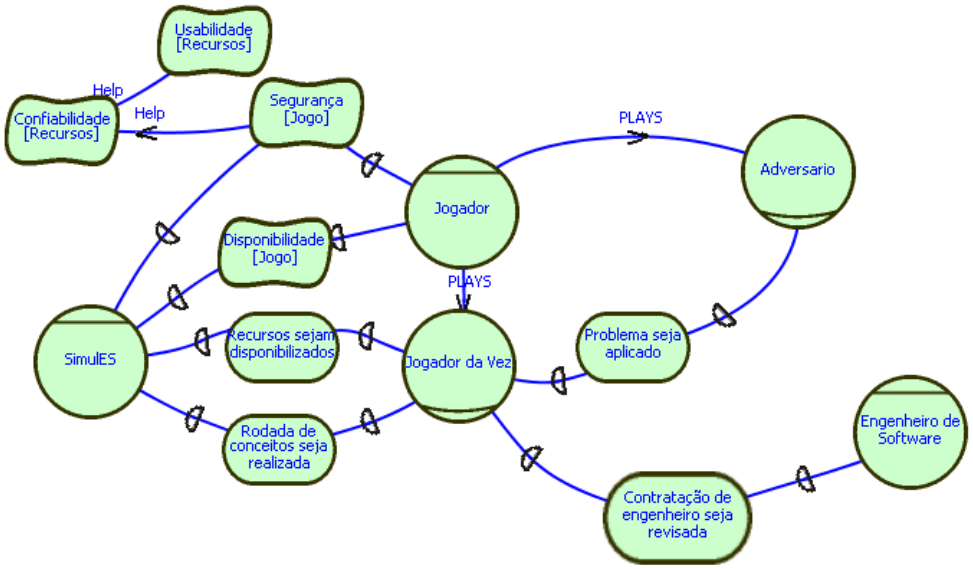


Figura 63 – Modelo SD – Joga Rodada de Conceitos.

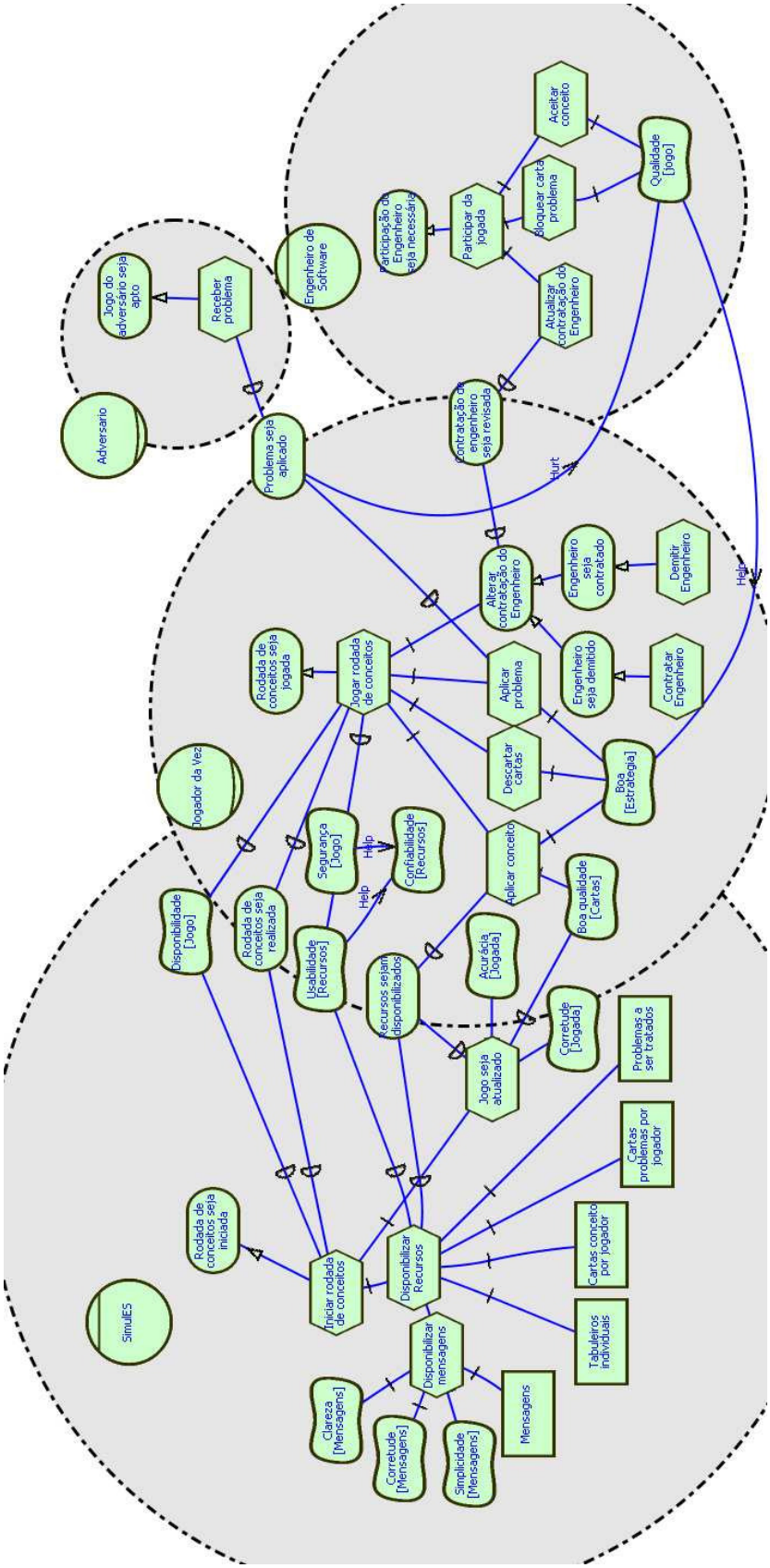


Figura 64 – Modelo SR – Joga Rodada de Conceitos.

SDsituation: Tratamento de Problema

Esta *SDsituations* retrata um novo ator, o SimulES encarregado de disponibilizar os recursos (Figura 65) e informar para todos os jogadores quando os problemas são tratados. Além disso, vemos na Figura 66 que o jogador em seu papel de adversário deve aceitar o tratamento que o *jogador da vez* dá aos problemas, isso para que o fluxo do jogo continue.

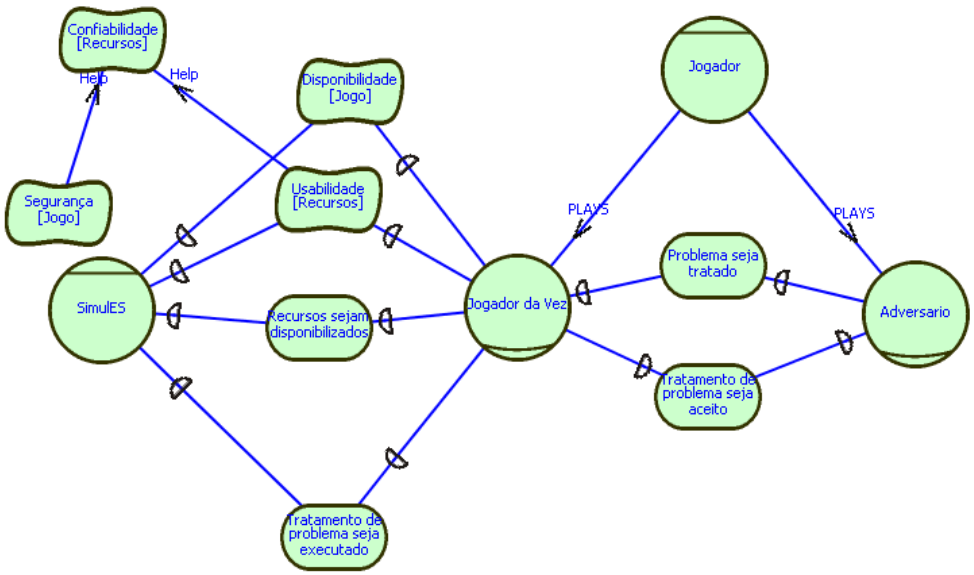


Figura 65 – Modelo SD – Tratamento de Problema.

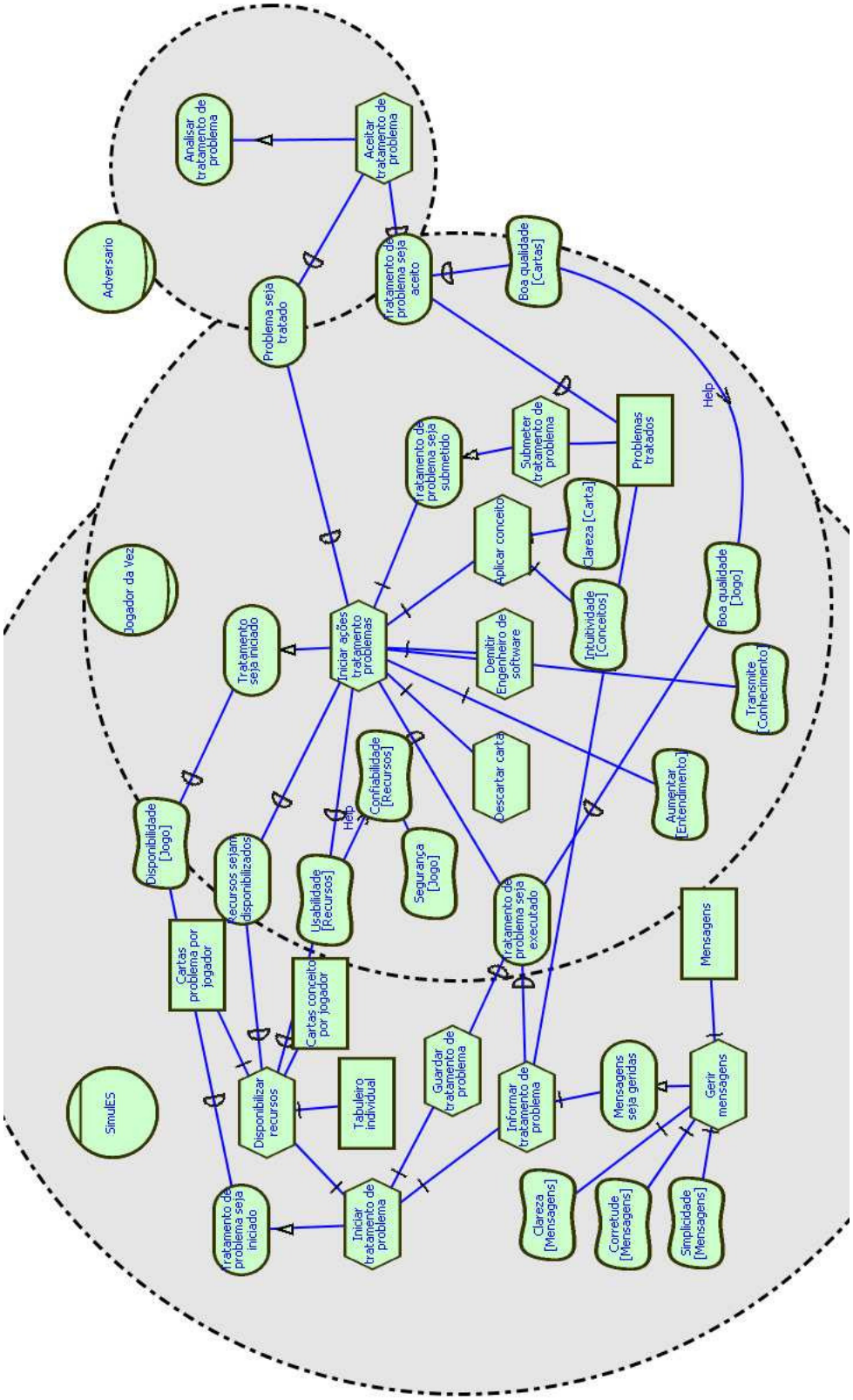


Figura 66 – Modelo SR – Tratamento de Problema.

SDsituation Submissão de Produto

Esta é a ultima *SDsituations* tratada dentro do núcleo do jogo, porém nela é possível tomar a decisão de finalizar o jogo (Figura 67). Se no momento de solicitar a submissão do produto o *jogador da vez* não fizer nenhuma tarefa de inspeção o SimulES, ou outro jogador no papel de adversário, pode realizar a tarefa de inspeção. Além disso, como vemos na Figura 68 um jogador no papel de administrador será quem define se o jogo finaliza e assim o sistema fechara todos os recursos e informara que a partida terminou e, como conseqüência, o jogador que submeteu o produto ganhará a partida.

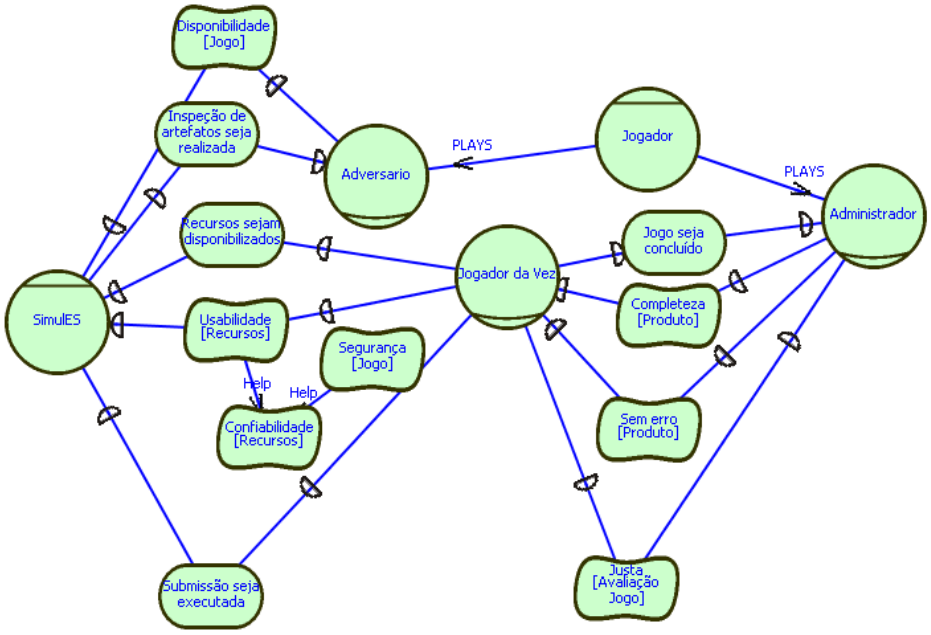


Figura 67 – Modelo SD – Submissão de Produto.

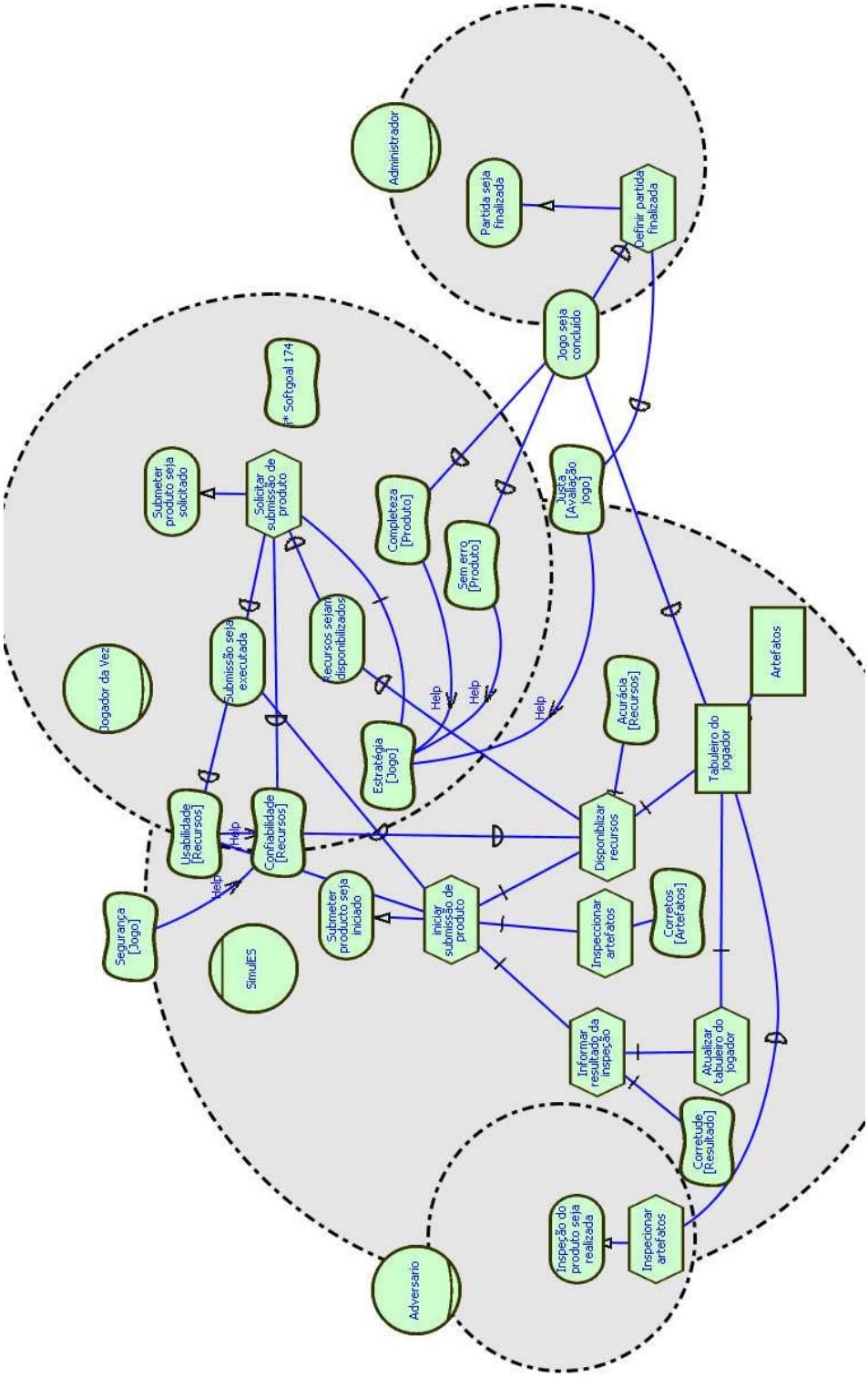


Figura 68 – Modelo SR – Submissão de Produto.

SDsituation: Integração de Artefatos em Módulo

Se bem que esta atividade pertence ao tempo 2 na ordem de execução das SDsituations (Figura 51) ela é uma atividade que está sendo realizada pelo SimulES como vemos na Figura 69, isso porque no momento de submeter o produto o SimulES realiza uma verificação de que todos os módulos estejam construídos e separa internamente os artefatos de cada um. Ele envia uma mensagem do resultado desta integração (Figura 70) para que o produto efetivamente possa ser submetido.

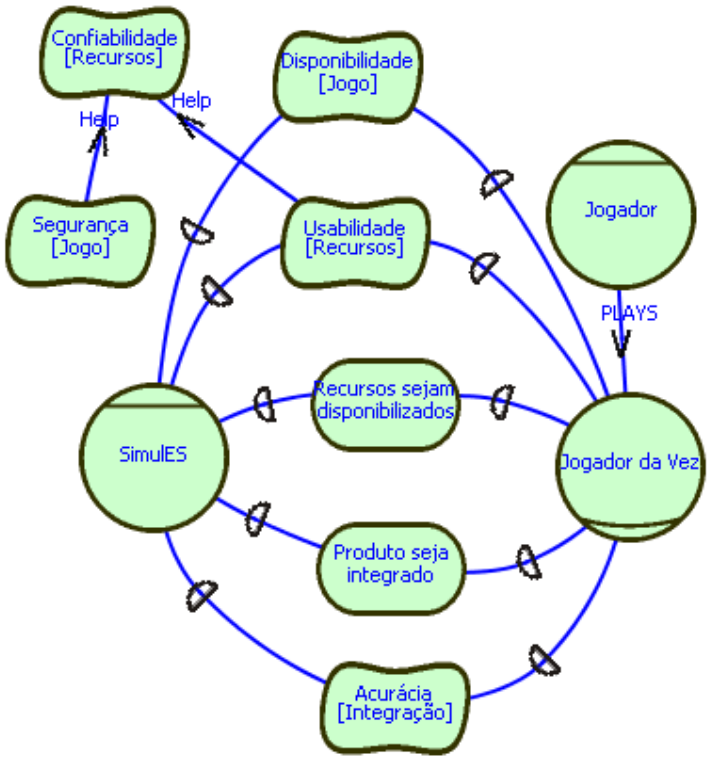


Figura 69 – Modelo SD – Integração de Artefato em Módulo.

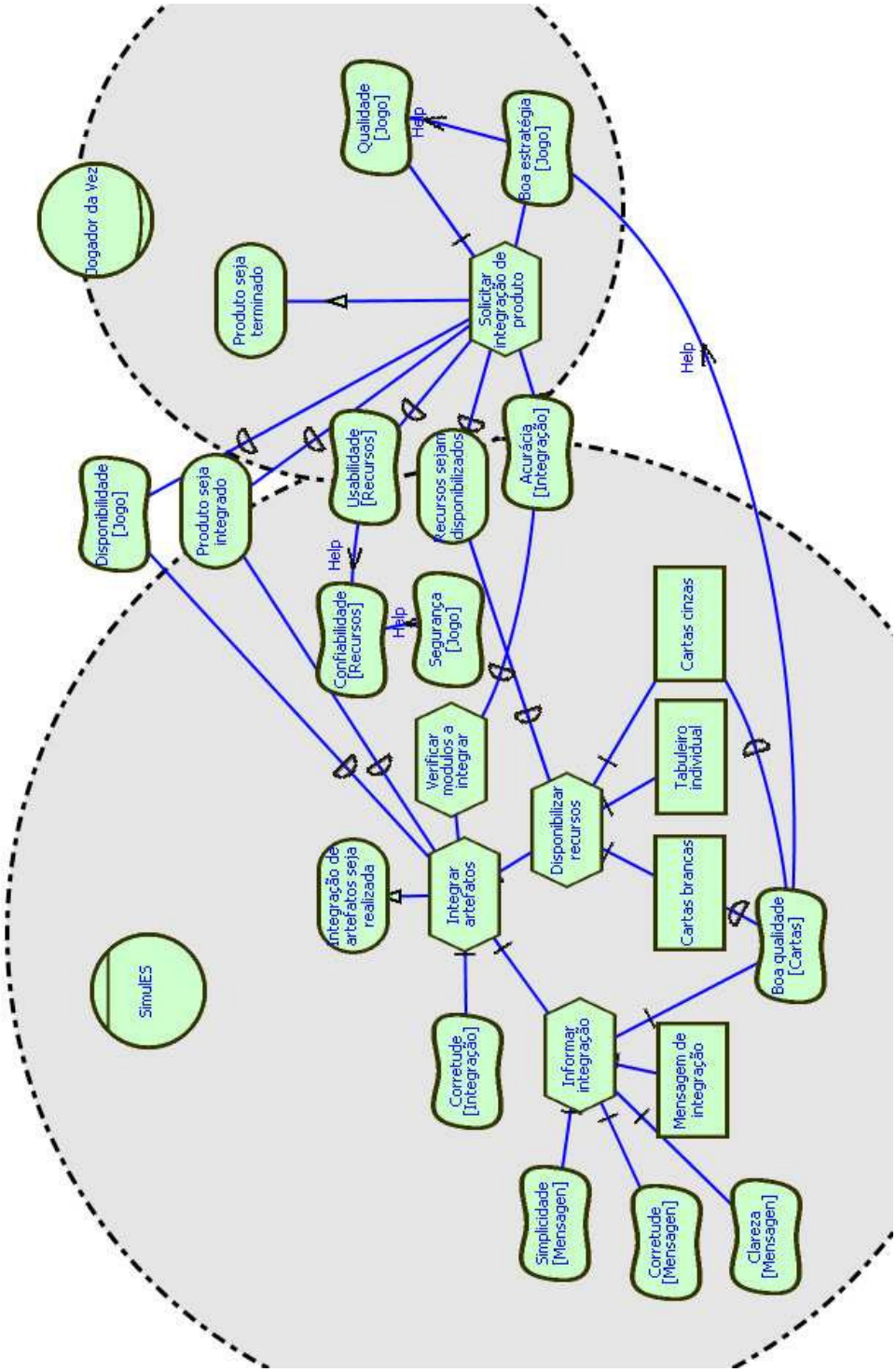


Figura 70 – Modelo SR – Integração de Artefato em Módulo.

6.4.2. *SDsituations* Auxiliares

Outras *SDsituations* que não pertencem ao núcleo do jogo mas ajudam na execução do mesmo e que tiveram sua origem na análise dos símbolos tipo objeto e que demandam comportamento dentro do SimulES-W, são apresentadas a seguir.

SDsituation: Apresentar Dinâmica do Jogo

Esta *SDsituations* surgiu como resposta a análise do comportamento do tabuleiro principal. Na primeira modelagem do SimulES [4] o tabuleiro individual é apresentado como um símbolo tipo objeto, já em [44] nos modelos intencionais preliminares aparece como um recurso. Em nossa análise encontramos que o tabuleiro individual tinha que ter comportamentos e funcionalidades próprias. E um ponto centralizador de informações relevantes do jogo além de ser o encarregado da troca de mensagens entre os jogadores pois jogadores não estariam no mesmo espaço físico.

Assim o tabuleiro central que era descrito no léxico em [44] como "*Área central da mesa definida por um papel impresso*", na versão do léxico para SimulES-W temos uma nova definição "*Página principal do jogo onde são disponibilizadas as informações de cartão do projeto escolhido, mensagens dos jogadores, informação dos movimentos do jogo, jogadores on-line.*". Como também está associado à *SDSituation Apresentar Dinâmica do Jogo*, onde jogador e SimulES são os atores desta interação (Figura 71). SimulES fornece os recursos e as informações do jogo, as quais serão administradas conforme às mensagens trocadas pelos jogadores e suas jogadas nas diferentes rodadas do jogo (Figura 72).

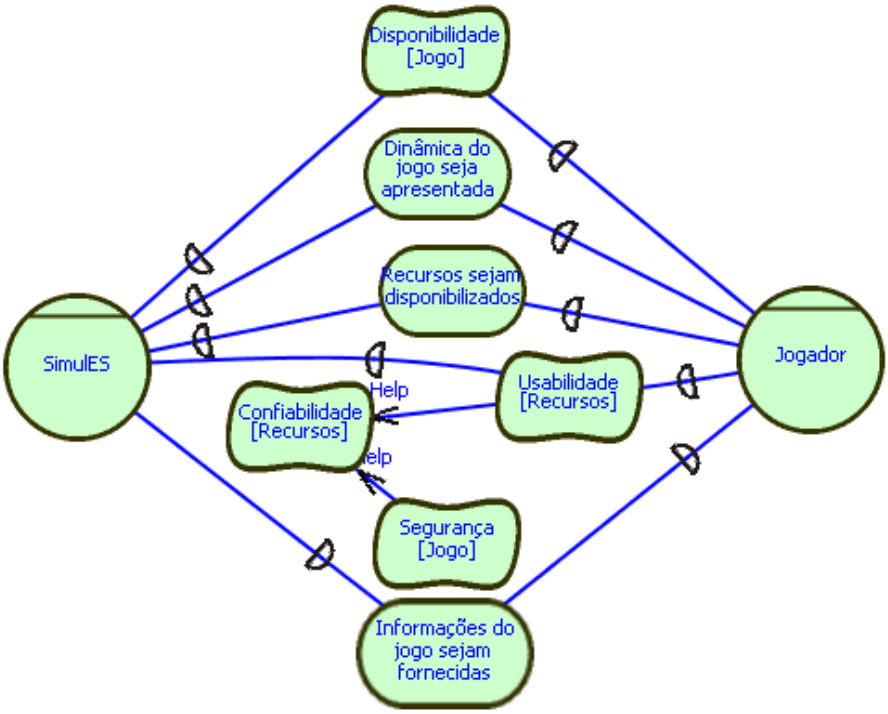


Figura 71 – Modelo SD – Apresentar Dinâmica do Jogo.

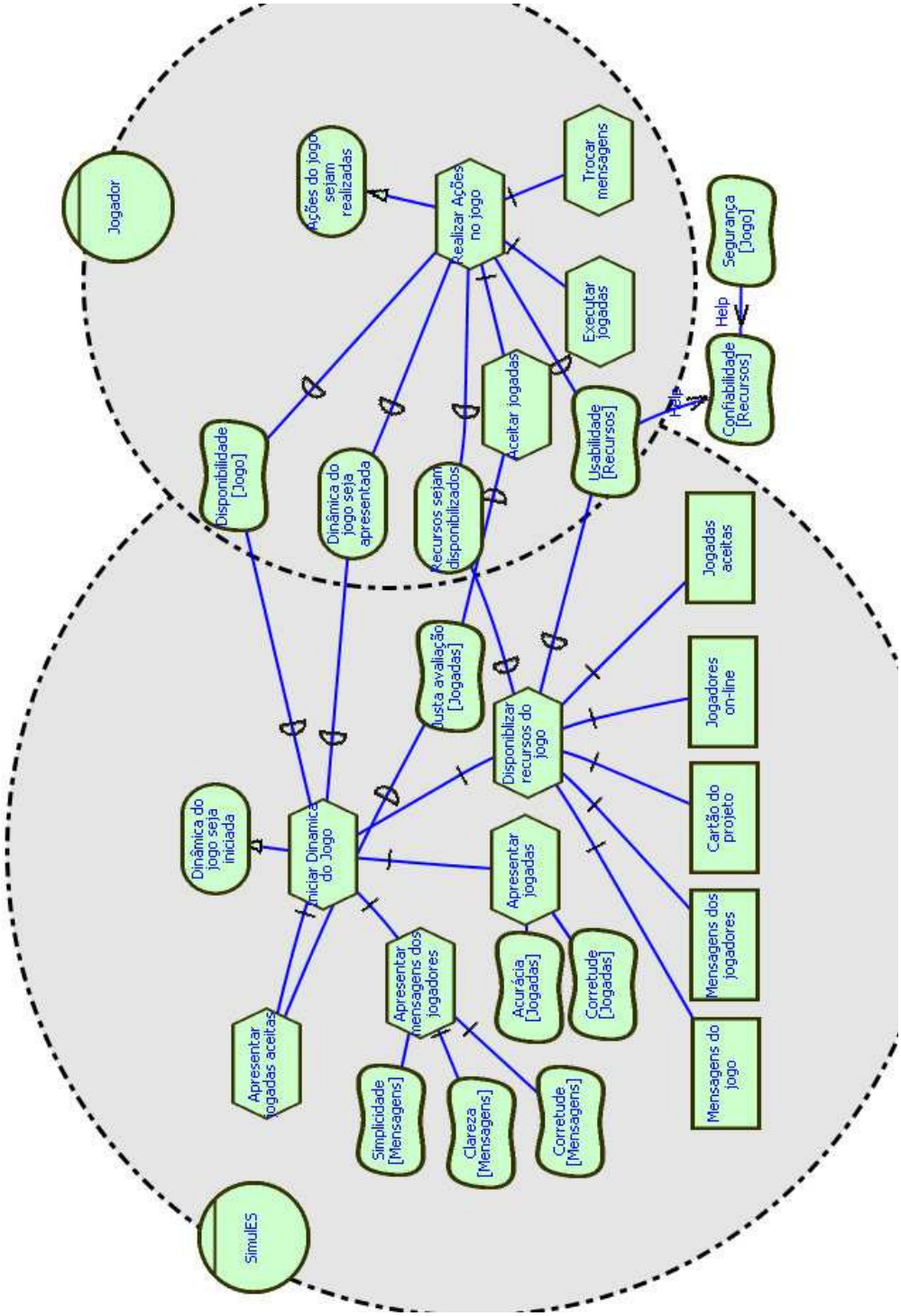


Figura 72 – Modelo SR – Apresentar Dinâmica do Jogo.

SDsituation: Gestão de Material de Apoio

O material de apoio permitirá tipificar o material que será usado durante o jogo, ou seja, se o interesse do instrutor é que temas específicos da área da engenharia de software sejam tratados no jogo ele pode criar o **material de apoio** e escolher este no inicio do jogo, para que as cartas a serem apresentadas sejam especializadas e relacionadas com o tópico escolhido. Como apresentamos nas Figuras 73 e 74 os atores relacionados nesta *SDsituation* são SimulES e Jogador no seu papel de administrador.

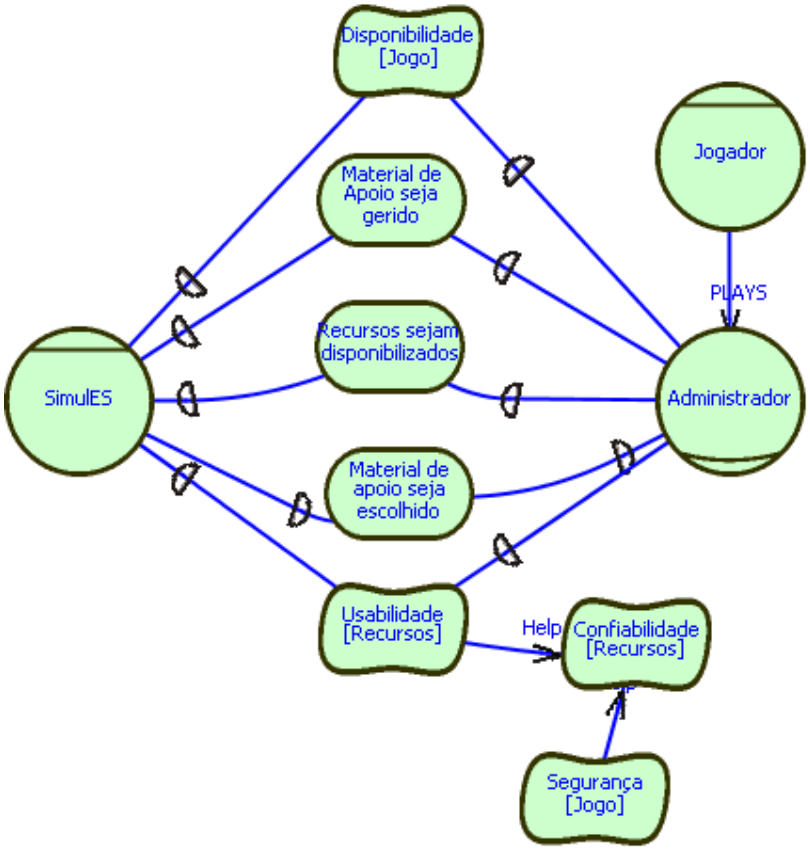


Figura 73 – Modelo SD – Gestão de Material de Apoio.

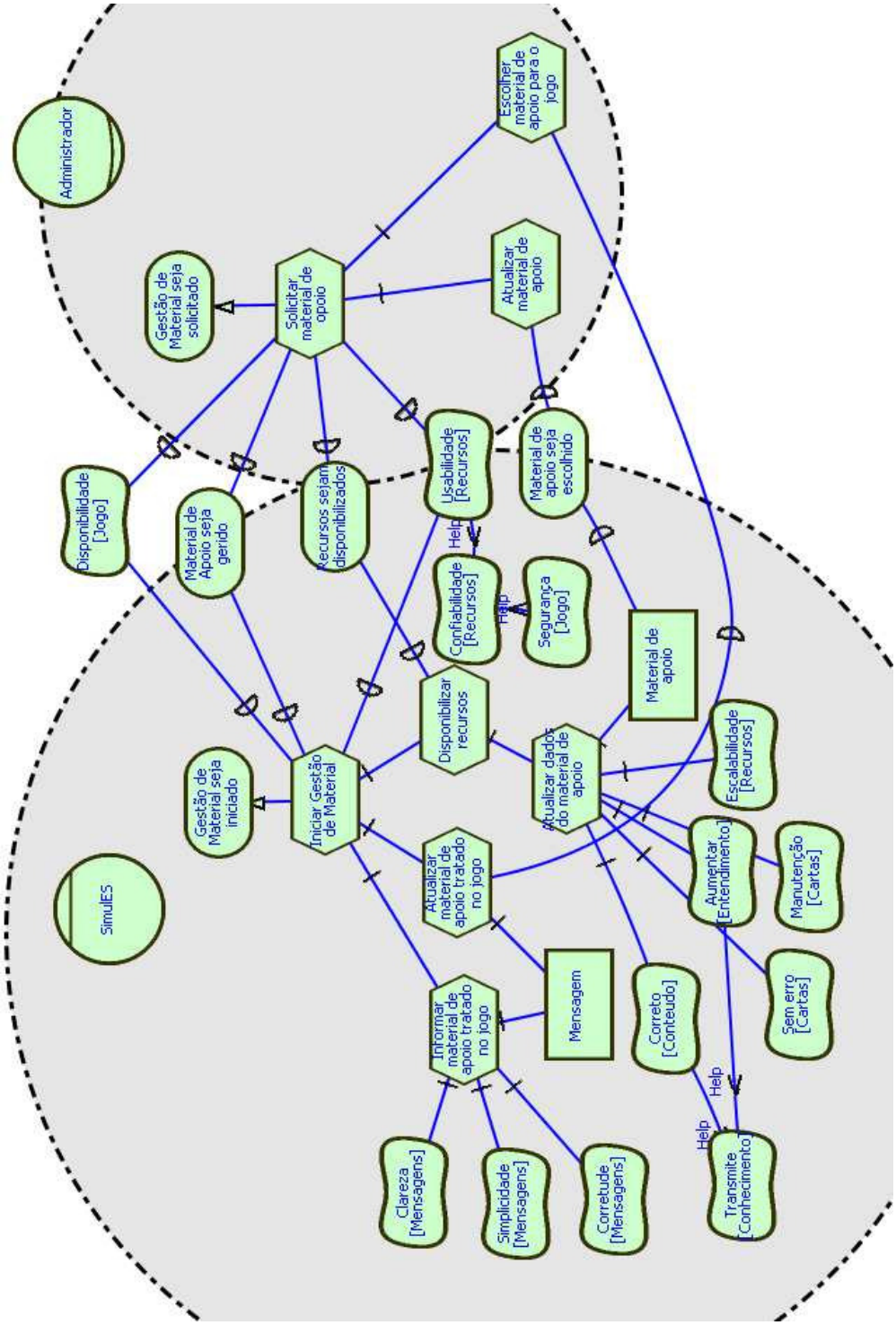


Figura 74 – Modelo SR – Gestão de Material de Apoio.

SDsituation: Gestão do Jogo

Esta *SDsituation* auxilia nos relacionamentos de colaboração entre o SimulES e o Jogador no seu papel de Administrador (Figura 75) onde são mostradas as atividades que levam tanto a dar inicio à sessão do jogo como ao fechamento do mesmo. Mas para que essas atividades sejam possíveis é necessário ter disponíveis todos os recursos a ser inicializados no inicio e no fim do jogo, além disso, o administrador deve decidir quando fechar a entrada de jogadores e fazer a solicitação ao SimulES. Outra atividade é escolher que tipo de material de apoio deve ser usado no decorrer do jogo todo isso é representado na Figura 76.

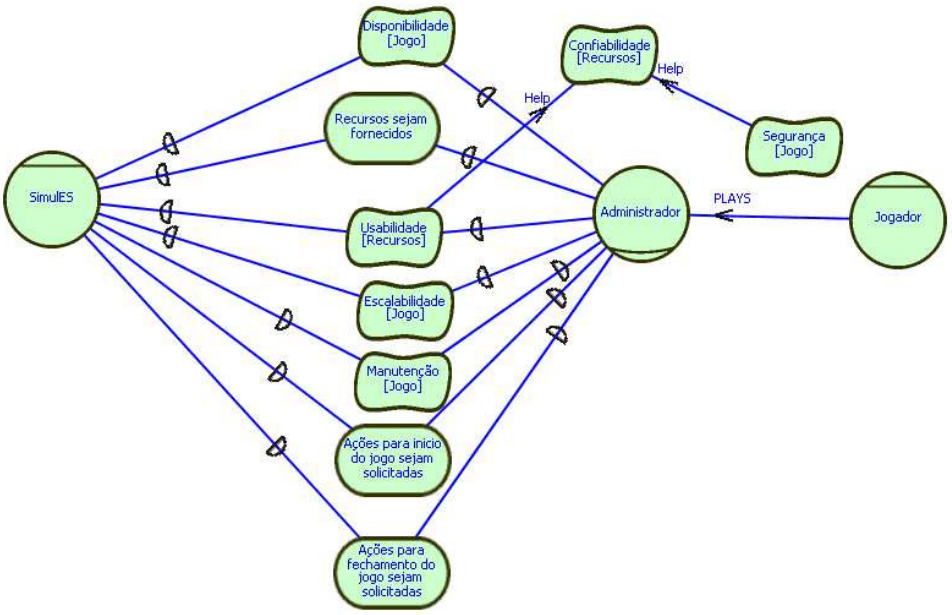


Figura 75 – Modelo SD – Gestão do Jogo.

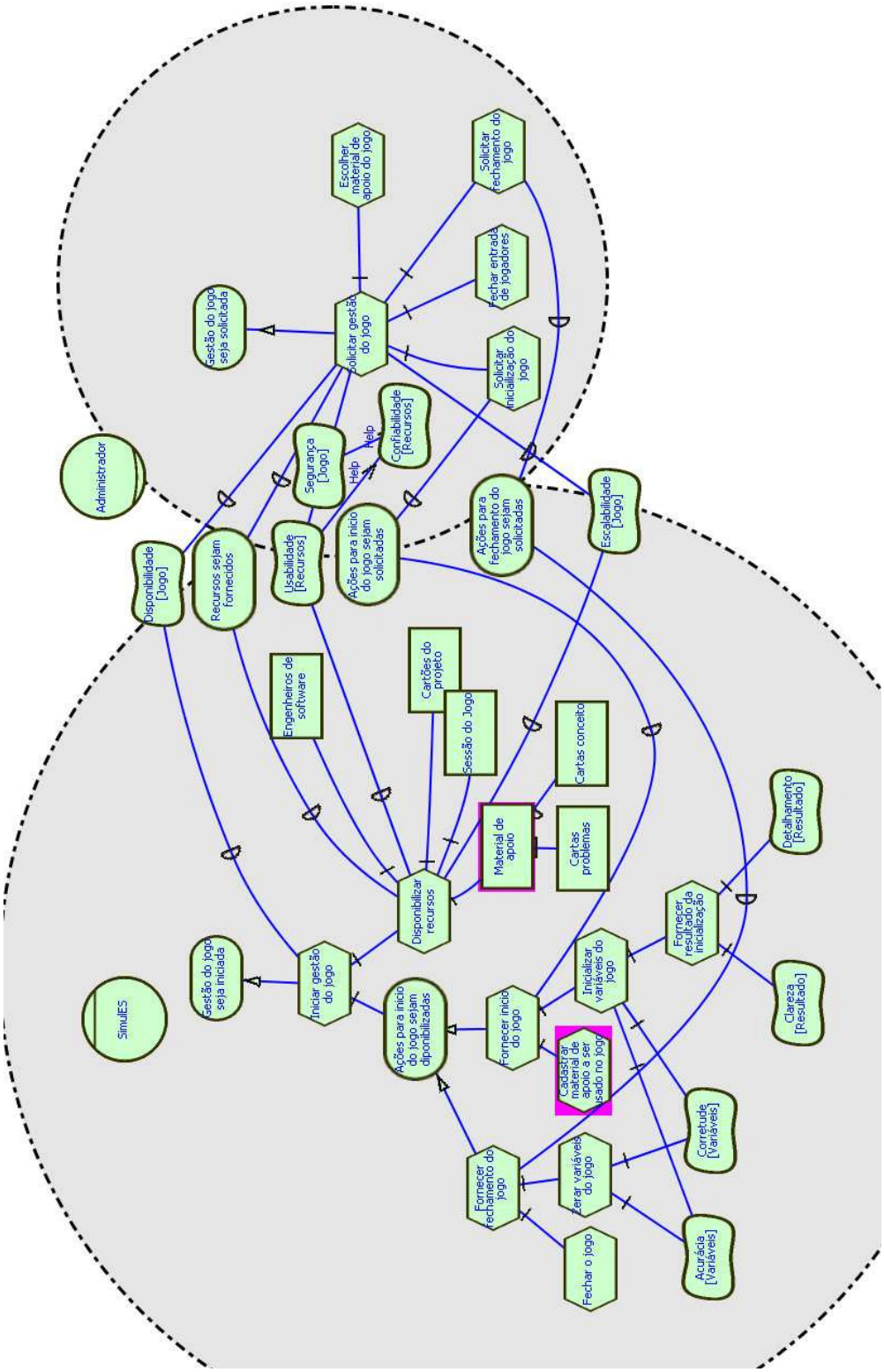


Figura 76 – Modelo SR – Gestão do Jogo.

6.4.3.Descrição das *SDsituations* do Sistema Através de Cenários

Com cenários podemos fazer a descrição passo a passo da operação do jogo e sua interação com os diferentes atores. Estes cenários constituem a descrição dos aspectos relevantes em quanto ao comportamento e o ambiente do SimulES-W, isso é feito através dos episódios, a vantagem e que é usada a linguagem natural semi-estruturada que facilita a leitura e o entendimento.

A evolução do SimulES que inclui modelos intencionais e a criação de um protótipo baseado nestes modelos, refinou o Léxico e descreveu as *SDSituations* (apresentar dinâmica do jogo, construção de artefatos, correção de artefato, gestão de material de apoio, gestão do jogo, inspeção de artefato, integração de artefatos no módulo, joga rodada de ações, joga rodada de conceitos, joga rodada de inicio, submissão de produto e tratamento de problema) em cenários segundo à implementação. Portanto estaremos primeiro mostrando o léxico do SimulES-W (Sub-seção 6.4.3.1) seguidos dos cenários do SimulES-W (Sub-seção 6.4.3.2).

6.4.3.1. Léxico do SimulES-W

Nome:	administrador
Noção:	Papel de **jogador** que é instanciado se realizar atividades de gestão do **jogo**. Observação: **símbolo do domínio**
Classificação:	sujeito
Impacto(s):	1- Reinicia as variáveis para inicio do **jogo** 2- Fecha a entrada de **jogadores** ao **jogo** 3- Escolhe o **material de apoio** usado no **jogo** 4- Gerencia o **material de apoio** do **jogo** 5- Fecha o **jogo**
Sinônimo(s):	

Nome:	adversário
Noção:	Papel de **jogador** que é instanciado se não é **jogador da vez** Observação: **símbolo do domínio**
Classificação:	sujeito
Impacto(s):	1-Recebe problemas através das cartas de problemas 2-Escolhe **módulo** 3-Inspeciona **módulo**
Sinônimo(s):	concorrente.

Nome:	aplicar conceito
Noção:	Procedimento ou execução de ação do **jogador da vez** para contrapor outra ação que perjudica seu **jogo**, ou para melhorar seu **jogo**. Observação: **símbolo do domínio**
Classificação:	verbo
Impacto(s):	1- **jogador da vez** neutraliza problema
Sinônimo(s):	

Nome:	aplicar problema
Noção:	**jogador da vez** pode escolher até 3 adversários e submeter a cada um deles uma **carta de problema**. Observação: **símbolo do domínio**
Classificação:	verbo
Impacto(s):	**adversário** recebe problema. **jogador da vez** trata problema.
Sinônimo(s):	

Nome:	artefato
Noção:	Recursos que simbolizam os produtos que são produzidos pelos engenheiros de sofware e que podem ou não ter defeitos. Artefatos com defeito possuem um smile vermelho. Artefatos sem defeito possuem um smile amarelho. **artefatos brancos** tem melhor qualidade que **artefatos cinza**, porém a probabilidade de erro é menor em **artefatos brancos**. São usados para construir os **módulos** de um produto. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	1-**engenheiro de software** constrói artefato 2-**engenheiro de software** inspeciona artefato 3-**engenheiro de software** corrige artefato
Sinônimo(s):	artefatos.

Nome:	artefato branco
Noção:	possuem uma proporção maior de **artefatos** sem defeito. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	
Sinônimo(s):	artefatos brancos.

Nome:	artefato cinza
Noção:	possui uma proporção maior de **artefatos** com defeito. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	
Sinônimo(s):	artefatos cinza.

Nome:	artefato construído
Noção:	1- Usado para construir os documentos de requisitos, desenho, código, rastro ou ajuda. 2-Um conjunto de **artefatos** construídos compõem um **módulo**. 3- Pode ser branco ou cinza. 4-Pode ter defeito ou não. 5-**artefato** construído esta na **tabuleiro individual** sem ser desvirado. Observação: **símbolo do domínio**
Classificação:	estado
Impacto(s):	1- **engenheiro de software** pode **inspecionar artefato**. 2- **engenheiro de software** pode **corrigir artefato**.
Sinônimo(s):	

Nome:	artefato corrigido
Noção:	1-É um **artefato**. 2-Pode ter defeito ou não. 3-permanece com a mesma cor do **artefato inspecionado** que apresentou defeito. 4-**artefato construído** esta na **tabuleiro individual** sem ser desvirado. Observação: **símbolo do domínio**
Classificação:	estado
Impacto(s):	1-**engenheiro de software** pode inspecionar o **artefato** corrigido.
Sinônimo(s):	

Nome:	artefato defeituoso
Noção:	1-É um **artefato** 2-Pode ser branco ou cinza 3-Possui um smile vermelho simbolizando um defeito. Observação: **símbolo do domínio**
Classificação:	estado
Impacto(s):	1-**engenheiro de software** pode **corrigir artefato** defeituoso.
Sinônimo(s):	

Nome:	artefato inspecionado
Noção:	1-É um **artefato** 2-Pode possuir defeito ou não. Observação: **símbolo do domínio**
Classificação:	estado
Impacto(s):	1-avalia a qualidade do **artefato**.
Sinônimo(s):	

Nome:	artefato livre de defeito
Noção:	1-É um **artefato** 2-Pode ser branco ou cinza 3-Possui um smile amarelho representando um **artefato** sem defeitos. Observação: **símbolo do domínio**
Classificação:	estado
Impacto(s):	1- **artefato** não precisa de nova inspeção.
Sinônimo(s):	

Nome:	artefatos de software
Noção:	Conjunto de **módulos** documentos construídos dentro do **jogo** requisitos (RQ), desenho (DS), código (CD), rastro (RT) ou ajuda (AJ). Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	1- **módulos** contem **artefatos** de software requisitos (RQ), desenho (DS), código (CD), rastro (RT) ou ajuda (AJ).
Sinônimo(s):	documento.

Nome:	carta
Noção:	Recurso que representa uma **carta de problema**, uma **carta de conceito**. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	1-**jogador da vez** compra cartas 2-**jogador da vez** descarta cartas 3-**jogador da vez** joga **carta de problema** para **adversário**
Sinônimo(s):	

Nome:	carta de conceito
Noção:	- Recurso do sistema que representam boas práticas de Engenharia de Software. - Possuem identificador, nome, referencia, descrição, tipo de **material de apoio**, regra, link - Podem neutralizar cartas de problemas. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	1-**jogador** compra **carta** de conceito. 2-**jogador** descarta **carta** de conceito. 3-**jogador** aplica conceito através da **carta** de conceito. 4-**jogador** coloca **carta** de conceito no tabuleiro.
Sinônimo(s):	cartas de conceito.

Nome:	carta de problema
Noção:	- Recursos do sistema que descrevem problemas clássicos de Engenharia de Software resultantes de faltas no processo de produção. - Possuem identificador, nome, referencia, descrição, tipo de **material de apoio**, regra, link - São utilizadas para criar obstáculos ao progresso dos adversários. - Podem ser neutralizadas por **cartas de conceito**. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	1-**jogador** compra **carta** de problema. 2-**jogador** descarta **carta** de problema. 3-**jogador** submete problema através de **carta** de problema.
Sinônimo(s):	cartas de problema.

Nome:	cartão de projeto
Noção:	Recurso no Sistema que informa detalhes sobre o projeto a ser desenvolvido no **jogo**. Um cartão de projeto contém uma descrição em linguagem natural com as principais características do projeto, um identificador único, um nome, a **complexidade do projeto**, o **tamanho do projeto**, a **qualidade do projeto** e o **orçamento do projeto** e um status para identificar qual projeto esta sendo usado no **jogo**. Um cartão de projeto deve ser selecionado por um **jogador** para ser desenvolvido ao longo do **jogo**. Cartão do projeto tem que ter **módulos** associados. O cartão de projeto selecionado deve ser no tabuleiro principal do **jogo** de forma que todos os **jogadores** tenham acesso as suas informações. A escolha do cartão de projeto é um pré-requisito para dar início ao **jogo**. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	1-**jogador** escolhe cartão de projeto que será considerado ao longo do **jogo**. 2-Adversario concorda com a escolha do projeto. 3-**jogador** obedece as especificações do cartão de projeto.
Sinônimo(s):	

Nome:	complexidade do projeto
Noção:	Indica quantos pontos de tempo um **engenheiro de software** precisa gastar para completar um **artefato** de boa qualidade. **artefatos** de má qualidade custam metade deste valor. Possui valor 2 ou 4.
Classificação:	objeto
Impacto(s):	1-**jogador** produz **artefatos** de acordo com as informações de complexidade do **cartão de projeto**.
Sinônimo(s):	

Nome:	comprar carta
Noção:	Pegar **carta** de um **monte** (**engenheiro de software**, **carta de problema**, **carta de conceito**) e reservada, ou seja, sem ser jogadas no **jogo** publico do **jogador**. As cartas são compradas pelo **jogador da vez**. As cartas são compradas de acordo com o **resultado do lançamento do dado**. Observação: **símbolo do domínio**
Classificação:	verbo
Impacto(s):	**jogador da vez** adquire **carta de conceito**. **jogador da vez** adquire **carta de problema**. **jogador da vez** adquire **engenheiro de software**. **jogador da vez** adquire **artefato**.
Sinônimo(s):	

Nome:	construir artefato
Noção:	**jogador da vez** através de seu(s) **engenheiro**(s) de software coloca em seu tabuleiro as cartas de **artefato**(s) (brancas ou cinzas), respeitando os pontos de **habilidade** do **engenheiro** e **complexidade do projeto**. o jogado da vez pode escolher construir com os **artefato**(s) documentos de requisitos, desenho, código, ajuda e rastro. **artefato** é construído por **engenheiro de software**. **artefato**(s) construído(s) podem ser de qualidade (branco) ou sem qualidade (cinza). Observação: **símbolo do domínio**
Classificação:	verbo
Impacto(s):	**artefato** é colocado no **tabuleiro individual** do **jogador da vez** de acordo com seu tipo e **engenheiro de software** que o produziu.
Sinônimo(s):	

Nome:	contratar engenheiro de software
Noção:	**jogador da vez** seleciona o **engenheiro de software** disponível que deseja contratar e atualiza seu estado para contratado, isso atualiza os engenheiros que aparecem no **tabuleiro individual**. No início do **jogo** é obrigatório que cada **jogador** contrate um **engenheiro de software**. Exceção: **engenheiro de software contratado** só poderá produzir **artefatos** no próximo turno exceto depois da primeira **rodada** quando os **jogadores** somente têm um **engenheiro de software**. Observação: simbolo do domínio
Classificação:	verbo
Impacto(s):	**engenheiro de software** pode **construir artefato**. **engenheiro de software** pode **inspecionar artefato**. **engenheiro de software** pode **corrigir artefato**. **jogador da vez** pode **demitir engenheiro de software**.
Sinônimo(s):	

Nome:	corrigir artefato
Noção:	O **artefato** a ser corrigido é substituído por um novo **artefato** da mesma cor não inspecionado. Observação: **símbolo do domínio**
Classificação:	verbo
Impacto(s):	**jogador da vez** possui um novo **artefato** em seu tabuleiro que fica sem ser desvirado para nova inspeção.
Sinônimo(s):	

Nome:	dado
Noção:	Recurso no sistema que é utilizado para sortear aleatoriamente números entre 1 e 6. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	1-**jogador** joga o dado. 2-**jogador** obtêm **resultado do lançamento do dado**. 3- Lançamento do dado permite **escolher projeto**. 4- Lançamento do dado permite comprar **carta de conceito** e **carta de problema**.
Sinônimo(s):	

Nome:	demitir engenheiro de software
Noção:	**jogador da vez** escolhe o **engenheiro** da lista de engenheiros e muda seu estado para demitido, esta faze com que ele volte para o **monte** de cartas de **engenheiro de software**. **engenheiro** demitido fica disponível dentro dos recursos do **simules_** e pode ser contratado por outro **jogador**. Observação: **símbolo do domínio**
Classificação:	verbo
Impacto(s):	**jogador da vez** pode usar orçamento novamente para a contratação de um novo **engenheiro**
Sinônimo(s):	

Nome:	descartar carta
Noção:	**jogador da vez** retorna a(s) **cartas de conceito** ou **cartas de problema** de sua lista de cartas, as seleciona e atualiza seu estado para descartar. Observação: **símbolo do domínio**
Classificação:	verbo
Impacto(s):	Cartas descartadas são disponibilizadas pelo **simules_** e podem ser escolhidas aleatoriamente por outros **jogadores**.
Sinônimo(s):	

Nome:	empacotar produto
Noção:	produzir todos os **módulos** necessários de acordo com o **cartão de projeto**.
Classificação:	verbo
Impacto(s):	Adversários escolhem **módulos** de acordo com a **qualidade do projeto**. Adversários inspecionam **módulos** selecionados. Se **módulos** verificados não apresentarem defeitos, **jogador** vence o **jogo**.
Sinônimo(s):	

Nome:	engenheiro de software
Noção:	Recurso no sistema que representa o profissional responsável por desempenhar ações. Possui **habilidade** e **maturidade**. Observação: **símbolo do domínio**
Classificação:	sujeito
Impacto(s):	1- Se encarrega da construção de **artefatos** 2- Se encarrega da correção de **artefato** 3- Se encarrega da inspeção de **artefato** 4- Se encarrega da integração de **artefatos** no modulo
Sinônimo(s):	engenheiros de software, engenheiro.

Nome:	engenheiro de software contratado
Noção:	1-É um **engenheiro de software** 2-**engenheiro** contratado é colocado no **tabuleiro individual** do **jogador** Observação: **símbolo do domínio**
Classificação:	estado
Impacto(s):	1-**engenheiro de software** pode **construir artefato**. 2-**engenheiro de software** pode **inspecionar artefato**. 3-**engenheiro de software** pode **corrigir artefato**.
Sinônimo(s):	

Nome:	engenheiro de software demitido
Noção:	1-É um **engenheiro de software** 2-**engenheiro** demitido retorna o grupo de **engenheiro** disponíveis para ser contratados. Observação: **símbolo do domínio**
Classificação:	estado
Impacto(s):	1- **jogador da vez** pode contratar outro **engenheiro de software**. 2- **engenheiro** demitido pode ser contratado por outro **jogador**.
Sinônimo(s):	

Nome:	escolher projeto
Noção:	**jogador da vez** na primeira **rodada** escolhe o **cartão de projeto** que será utilizado no **jogo**. **concorrente**(s) devem concordar com projeto escolhido. Observação: **símbolo do domínio**
Classificação:	verbo
Impacto(s):	1- Os **jogadores** usam as informações do **cartão de projeto**. 2- **engenheiros de software** usam as informações do **cartão de projeto**.
Sinônimo(s):	

Nome:	habilidade
Noção:	Número que indica o quanto o **engenheiro de software** possui para, a cada **rodada**, desempenhar ações no **jogo**. Número que limita as ações a serem desempenhadas pelo **engenheiro de software**. Número inteiro no intervalo [1,5]. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	1-**engenheiro de software** usa habilidade para **construir artefato**. 2-**engenheiro de software** usa habilidade para **corrigir artefato**. 3-**engenheiro de software** usa habilidade para **inspecionar artefato**.
Sinônimo(s):	

Nome:	inspecionar artefato
Noção:	Desvirar os **artefatos** selecionados para verificar a existência ou não de defeito. o custo de inspecionar o **artefato** é metade do custo de construir um **artefato**. **artefato** é inspecionado por **engenheiro de software**. Observação: **símbolo do domínio**
Classificação:	verbo
Impacto(s):	1- **jogador da vez** conhece o **artefato inspecionado**. 2- **jogador da vez** pode **corrigir artefato**, caso ele seja defeituoso. 3- O **simules_** também pode fazer inspeção quando o **jogador** submete o produto.
Sinônimo(s):	

Nome:	integrar artefatos no modulo
Noção:	**simules_** forma os mesmos **módulos** descritos no cartão do projeto com os **artefatos** utilizados pelo **jogador** para construir o produto no seu **tabuleiro individual**. Observação: **símbolo do domínio**.
Classificação:	verbo
Impacto(s):	Se **jogador** possuir todos os **módulos** o **simules_** integrara o produto e assim o **jogador** poderá **submeter produto**.
Sinônimo(s):	

Nome:	jogador
Noção:	Participante do **jogo simules_**. Tem por objetivo vencer o **jogo**. Jogador pode asumir o papel de **jogador da vez** Jogador pode asumir o papel de adversario Jogador pode asumir o papel de **administrador** Observação: **símbolo do domínio**
Classificação:	sujeito
Impacto(s):	1- Jogador se registra no **jogo** 2- Jogador um papel no **jogo**
Sinônimo(s):	jogador on-line, jogadores, jogadores on-line.

Nome:	jogador da vez
Noção:	Papel de **jogador** que esta executando uma ação ou jogada dentro de **jogo**. Observação: **símbolo do domínio**
Classificação:	sujeito
Impacto(s):	1- joga **rodada** de inicio. 2- joga **rodada** de ações. 3- aplica conceitos (joga **rodada** de conceitos). 4- integração de **artefatos** no **módulo**. 5- submissão de produto. 6- aplica problema (joga **rodada** de conceitos). 7- **contratar engenheiro de software**. 8- **demitir engenheiro de software**.
Sinônimo(s):	

Nome:	jogar dado
Noção:	arremeçar o **dado** no joga **rodada** de inicio ou joga **rodada** de ações. Observação: **símbolo do domínio**
Classificação:	verbo
Impacto(s):	1- **jogador da vez** obtém **resultado do lançamento do dado**. 2- **jogador** com melhor resultado do **dado** em joga **rodada** de inicio escolhe o projeto. 3- O resultado do **dado** em joga **rodada** de ações permite a compra de **cartas de conceito** e/ou **cartas de problema**
Sinônimo(s):	

Nome:	jogar rodada de ações
Noção:	Nesta **rodada** o **jogador** pode adquirir cartas de conceitos, cartas de problemas, e/ou cartas de **engenheiros de software**, de acordo com o **resultado do lançamento do dado**. O **jogador** também poderá construir **artefatos**, inspecionar **artefatos** e/ou corrigir **artefatos**.
Classificação:	verbo
Impacto(s):	**simules** inicia **rodada** de ações **jogador da vez** joga **rodada** de ações **jogador da vez** obtém **resultado do lançamento do dado**
Sinônimo(s):	

Nome:	jogar rodada de conceitos
Noção:	Ações que o **jogador** realiza dentro do **jogo**, onde pode descartar cartas excedentes, contratar e/ou demitir **engenheiros de software**, submeter problemas e tratar problemas. Observação: **símbolo do domínio**
Classificação:	verbo
Impacto(s):	**simules_** inicia **rodada** de conceitos Se o **jogo** do **jogador** esta completo pode solicitar **empacotar produto** e **submeter produto**.
Sinônimo(s):	

Nome:	jogo
Noção:	Ferramenta lúdico-pedagógica desenvolvida para facilitar o ensino da Engenharia de Software. Observação: **símbolo do domínio** Possui de 1 a 8 **jogadores**. Possui **cartas de conceito** e cartas de problemas. Possui **tabuleiro individual** e tabuleiro principal. Possui **cartão de projeto**. Possui joga **rodada** de inicio quando o Jogo começa. Possui submissão de produto quando o jogo finaliza Possui rodadas. Observação: **símbolo do domínio**.
Classificação:	objeto
Impacto(s):	1- **jogador** joga o jogo. 2- **simules_** é o orquestrador do jogo
Sinônimo(s):	

Nome:	material de apoio
Noção:	Classificação do material a ser tratado dentro do **jogo**. Fontes de informação das quais serão tomadas as cartas conceito e cartas problema, conforme seja o interes do instrutor sobre os conteúdos nestas cartas (conceito e cartas).
Classificação:	objeto
Impacto(s):	1- No inicio do **jogo** o material de apoio deve ser escolhido 2- Material de apoio indica a os temas a serem tratados no **jogo**
Sinônimo(s):	

Nome:	maturidade
Noção:	Número inteiro entre [1,5] e é uma das características do **engenheiro de software**. Quanto maior o número, maior a facilidade do **engenheiro** em trabalhar com os outros. Indica um **engenheiro** com personalidade que facilita o trabalho em conjunto. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	**cartas de problema** podem usar a maturidade como condição para **aplicar problema**.
Sinônimo(s):	

Nome:	módulo
Noção:	Parte que compõe um projeto. Pode ser formado por **documento** de requisitos(RQ), desenho (DS), código (CD), rastro (RT) ou ajuda (AJ). Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	**simules_** integra para **jogador da vez artefatos** em modulos.
Sinônimo(s):	módulos.

Nome:	orçamento do projeto
Noção:	Simboliza a quantidade de dinheiro disponível para gastar com o projeto. Funciona como restrição para contratação de **engenheiro de software** bem como para uso de **cartas de conceito**. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	1- **jogador** usa orçamento do projeto para contratar engenheiros. 2- **jogador** usa orçamento do projeto para usar cartas conceito.
Sinônimo(s):	

Nome:	problema persistente
Noção:	1-O problema acompanha o **jogador** por todo o **jogo**, a não ser se for contraposto por uma **carta de conceito**. 2-Deve ser dentro das cartas problemas do **jogador**. Observação: **símbolo do domínio**
Classificação:	estado
Impacto(s):	1-Adversario trata problema. 2-Pode ser contraposto por **carta de conceito**.
Sinônimo(s):	

Nome:	problema temporário
Noção:	1-Atrapalha o **jogo** do **jogador da vez** 2-O **adversário** sofre a penalidade no momento de sua aplicação. 3-Após ser aplicado, o problema retorna para o **repositório** de cartas de problemas e/ou cartas de conceitos. 4- Geralmente dura uma **rodada** só. Observação: **símbolo do domínio**
Classificação:	estado
Impacto(s):	1- **jogador da vez** trata problema 2- **adversário** concorda com tratamento do problema
Sinônimo(s):	

Nome:	produto de software
Noção:	Conjunto de **módulos** construídos pelos **engenheiros de software**. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	1- Produtos de software possuem **módulos** 2- **módulos** contem **artefatos de software** requisitos (RQ), desenho (DS), código (CD), rastro (RT) ou ajuda (AJ). 3- **artefatos de software** possuem **artefatos brancos** e **artefatos cinza**
Sinônimo(s):	

Nome:	qualidade do projeto
Noção:	Representa o quão livre de defeitos deve estar o produto final. O valor varia de 1 a 5 indicando o número mínimo de **módulos** sem defeitos necessários para vencer o **jogo**. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	**jogador** usa qualidade do projeto. **jogador** deve possuir um número mínimo de **módulos** sem defeito.
Sinônimo(s):	

Nome:	receber problema
Noção:	receber **carta de problema** submetida pelo **jogador da vez** durante a **rodada** de conceitos Observação: **símbolo do domínio**
Classificação:	verbo
Impacto(s):	1- **simules** notifica ao **jogador** que recebeu **carta de problema** 2- **jogador da vez** trata problema. 3- **adversário** concorda com o tratamento à **carta** problema **dado** pelo **jogador da vez**
Sinônimo(s):	

Nome:	repositório
Noção:	Conjunto de cartas ou cartões disponíveis para seleção, compra ou descarte por parte dos **jogadores**. **jogadores** podem comprar cartas do repositório de cartas. **jogadores** podem descartar cartas e estas voltam para o repositório de cartas. **jogadores** podem selecionar um cartão do repositório de cartões. repositórios são geridos pelo **simules_**. **administrador** solicita ações nos repositórios. repositórios devem ser acessíveis por todos os **jogadores**. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	**simules** disponibiliza os repositórios para o **jogo**. **jogador** compra **carta** do repositório de cartas. **jogador** descarta **carta** e **volta** para repositório de cartas. **jogador** seleciona um projeto do repositório de cartões de projetos.
Sinônimo(s):	repositórios, monte.

Nome:	resultado do lançamento do dado
Noção:	Número no intervalo [1,6] resultante do lançamento do **dado**. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	1- Resultado do lançamento do **dado** determina quem escolhe o projeto. 2- Resultado do lançamento do **dado** determina quantas cartas serão compradas. 3- Resultado do lançamento do **dado** determina o tipo de **carta** que será comprada.
Sinônimo(s):	

Nome:	rodada
Noção:	Cada uma das vezes que se executa uma SDsitation.
Classificação:	objeto
Impacto(s):	1- Todos os **jogadores** deve passar pelas diferentes rodadas
Sinônimo(s):	

Nome:	ronda
Noção:	Movimento ou giro completo em todas as rodadas (SDsituations) do inicio ate o fim.
Classificação:	objeto
Impacto(s):	1- A ronda é cíclica ate que algum **jogador** possa **submeter produto** e ganhe o **jogo**. 2- A ronda esta composta de rodadas.
Sinônimo(s):	volta.

Nome:	sentido de jogo
Noção:	Forma pela qual os **jogadores** se sucedem ao longo do **jogo**. O sentido de **jogo** no **simules** é conforme os **jogadores** foram registrados no **jogo**. O sentido de **jogo** não pode ser desrespeitado. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	1- **jogadores** obedecem sentido de **jogo**. 2- **simules** fornece controles para que o sentido do **jogo** seja respeitado
Sinônimo(s):	

Nome:	símbolo do domínio
Noção:	Símbolos que pertence ao domínio do **jogo**.
Classificação:	objeto
Impacto(s):	
Sinônimo(s):	

Nome:	simules
Noção:	É um **jogo** educacional que simula várias situações da Engenharia de Software. Observação: **símbolo do domínio**
Classificação:	sujeito
Impacto(s):	1-Fornece recursos para os **jogadores** 2-Orquestra os recursos do **jogo** 3- Fornece controles dentro do **jogo**. 4- controla e fornece as mensagens dos **jogadores**. 5- Informa sobre os movimentos dentro do **jogo**
Sinônimo(s):	simules_.

Nome:	submeter produto
Noção:	mostrar o conteúdo de alguns ou todos os **módulos** construídos, de acordo com a **qualidade do projeto**. Observação: **símbolo do domínio**
Classificação:	verbo
Impacto(s):	1- **jogador** apresenta os **módulos** exigidos na especificação do projeto. 2- Se **módulos** inspecionados não apresentarem defeitos, **jogador** vence o **jogo**.
Sinônimo(s):	

Nome:	tabuleiro central
Noção:	Pagina principal do **jogo** onde são disponibilizadas as informações de cartão do projeto escolhido, mensagens dos **jogadores**, informação dos movimentos do **jogo**, **jogadores on-line**. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	1- Tabuleiro central orienta as rodadas do **jogo**. 2- No tabuleiro central os **jogadores** trocam mensagens. 3- No tabuleiro central aparece o projeto escolhido a ser tratado no **jogo**. 4- No tabuleiro central é resumido os movimentos no **jogo**.
Sinônimo(s):	index, pagina principal do jogo, mesa.

Nome:	tabuleiro individual
Noção:	Pagina definida no **jogo** e apresenta os **artefatos** de requisito, **artefatos** de código, **artefatos** de ajuda, **artefato** de rastro que o **jogador** deve construir. **jogador** coloca seus **artefatos** braços ou **artefatos cinza** no tabuleiro. cada **jogador** tem um tabuleiro. no tabuleiro se dispõe um espaço para por os **engenheiros de software** contratados. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	1- **jogador** coloca **artefatos brancos** e/ou **artefatos cinza** no tabuleiro individual. 2- **jogador** retira **artefatos** do tabuleiro individual. 3- **jogador** retira cartas do tabuleiro individual. 4- **jogador** desvira **artefatos** do tabuleiro individual.
Sinônimo(s):	

Nome:	tamanho do projeto
Noção:	Indica quantos **módulos** integrados devem ser completados para empacotar o produto e vencer o **jogo**. Um projeto pode ter tamanho máximo 6. Observação: **símbolo do domínio**
Classificação:	objeto
Impacto(s):	**jogador** usa informações de tamanho do projeto.
Sinônimo(s):	

Nome:	tratar problema
Noção:	Procedimento ou execução de ação do **jogador da vez** para contrarestar outra ação que perjudica seu **jogo**. Observação: **símbolo do domínio**
Classificação:	verbo
Impacto(s):	1- **jogador da vez** pode contrapor problema aplicado. 2- **jogador da vez** atende a demanda do problema. 3- **adversário** concordam com tratamento do problema.
Sinônimo(s):	

6.4.3.2. Descrevendo as SDsituations através de Cenários

Segundo [11] para descrever *SDsituations* através da técnica de cenários é preciso o uso do Diagrama *SDsituations* e dos Modelos SR procurando-se maximizar o uso do LAL[10]. Foram utilizadas as *SDsituations* da Sub-seção 6.4.1. (Identificar as *SDsituations* Principais) e da Sub-seção 6.4.2. (*SDsituations* auxiliares).

Nesta descrição damos ênfase aos elementos, regras e dinâmica do jogo através da narrativa oferecida pelos cenários. Conforme [59] o uso de cenários para descrever o sistema torna este mais transparente o que ajuda nosso propósito neste trabalho.

SDsituations principais

Titulo: SDsituations joga rodada de inicio
Objetivo: Descrever os preparativos para inicio do jogo.
Contexto:
Ubicação geográfica: Web
Ubicação temporal: Java PlayStartRoundPage
Precondição:
- cenário executado no inicio do jogo somente
- Variáveis do jogo inicializadas
- Sessão do jogo estabelecida

Pos-condição:
- jogadores registrados.
- A ordem das jogadas dos jogadores já foi definida dentro do jogo.
- Projeto escolhido.
- executar cenário joga rodada de ações, construção de artefatos

Atores: jogador, SimulES Recursos: dado, cartas engenheiro, informações do projeto.
Episódios:
1- Os jogadores se cadastram no jogo
2- SimulES registra os jogares
3- SimulES anuncia os jogadores cadastrados
4- O jogador administrador fecha a entrada de jogadores
5- O jogador lança o dado. Restrição: jogador que tirar o maior número no dado, escolhe o projeto
6- SimulES valida a escolha do projeto
7- SimulES anuncia o projeto escolhido
8- Cada jogador compra uma carta de engenheiro de software
9- SimulES registra o engenheiro de software escolhido
Restrição: a ordem na qual os jogadores jogam será acorde com a ordem na qual os jogadores se cadastraram.

Titulo: SDsituations joga rodada de ações
Objetivo: Descrever as regras da rodada de ações.
Contexto:
Ubicação geográfica: Web
Ubicação temporal: IndividualBoardPage e PlayConceptsRoundPage
Pre-condição:
- projeto escolhido
- um engenheiro de software por cada jogador

Pos-condição:
- Jogadas registradas no jogo

Atores: jogador Recursos: dado, cartas, informações do projeto, página tabuleiro individual e página principal do jogo.
Episódios:
1- jogador usa seus engenheiros de software em CONSTRÓI artefato ou INSPECIONA artefato ou CORRIGE artefato e joga lançamento do dado.
2- jogador lança o dado.
3- Se dado igual a 1,ou 2 ou 3, então jogador pega 1, 2 ou 3 cartas do monte de problemas e conceitos. Restrição: jogador não pode pegar cartas do monte de engenheiros de software.
4- Se dado igual a 4 ou 5 ou 6, então jogador pega 3 cartas do monte de problemas e conceitos e x cartas do monte de engenheiro de software, onde x é o valor do dado menos 3.
Restrição: o primeiro episodio é executado de acordo com a habilidade de cada engenheiro de software.
Restrição: Habilidades e o custo dessas ações podem ser afetados por cartas de conceitos ou cartas de problemas.
Restrição: Se jogador não usou engenheiros de software e nem jogou o dado, então INTEGRA artefatos EM UM módulo.

Titulo: SDsituations construção de artefatos
Objetivo: Descrever as regras da construção de artefatos.
Contexto:
Localização geográfica: Web
Localização temporal: Java IndividualBoardPage
Pre-Condição: Pelo menos um engenheiro no tabuleiro
Pos-condição: artefatos construídos no tabuleiro do jogador.
Atores: jogador da vez e engenheiro de software
Recursos: cartão de projeto na página principal do jogo, engenheiros de software no tabuleiro individual, jogador com cartas brancas e cartas cinzas disponíveis Exceção: jogador penalizado para executar a jogada
Metas flexíveis: qualidade[artefato], completeza[artefato]
Restrições:
A quantidade de artefatos construídos depende da habilidade do engenheiro de software, do tipo de artefato escolhido e da complexidade do projeto.
Cartas brancas custam valor da complexidade do projeto.
Cartas cinzas custam a metade da complexidade do projeto
Episódios:
1- jogador da vez compra cartas brancas ou cartas cinza
2- jogador da vez disponibiliza no tabuleiro individual as cartas brancas ou cartas cinza compradas
3- engenheiro de software constrói produto
4- O jogador submete a jogada realizada
5- SimulES guarda as informações de tabuleiro individual criada pelo jogador
6- Informações do tabuleiro individual podem ser vistas por todos os jogadores

Titulo: SDsituations inspeção de artefato
Objetivo: Descrever as regras da inspeção de artefatos.
Contexto: Ubicação geográfica: Web
Ubicação temporal: Java IndividualBoardPage
Pre-condição:
- Cenário construção de artefatos já executado
- informações do projeto na página central do projeto.
- jogador tem cartas brancas e/ou cartas cinzas no tabuleiro.
Pos-condição:
- artefatos com o resultado da inspeção
Atores: jogador, engenheiro de software Recursos: cartas brancas, cartas cinzas, informações do projeto na página principal do jogo, tabuleiro individual.
Episódios:
1- jogador escolhe artefato do tabuleiro.
2- Se o engenheiro de software responsável pelo artefato faz a inspeção, então o mesmo gasta um ponto de tempo.
3- Se outro engenheiro de software faz a inspeção, então ele gasta dois pontos de tempo.
4- O resultado da inspeção é fornecido pelo SimulES
5- O usuário submete sua inspeção
6- artefatos inspecionados são visíveis para todos os jogadores

Titulo: SDsituations correção de artefato
Objetivo: Descrever as regras da correção de artefatos.
Contexto: Localização geográfica: Web
Localização temporal: Java IndividualBoardPage
Pre-condição:
- informações do projeto a página principal do jogo.
- jogador tem artefatos inspecionados com defeito (INSPECIONA artefato) no tabuleiro individual.
Pos-condição: SimulES fornece para todos os jogadores o resultado da inspeção
Atores: jogador, engenheiro de software
Recursos: Cartas inspecionadas com defeito, informações do projeto
Episódios:
1- jogador escolhe artefato com defeito do tabuleiro.
2- jogador descarta carta do tabuleiro individual.
3- SimulES apaga a carta do tabuleiro individual do jogador.
4- Se o engenheiro de software responsável pelo artefato faz a correção, então o mesmo gasta um ponto de tempo.
5- Se outro engenheiro de software faz a correção, então ele gasta três pontos de tempo.
6- artefato inspecionado é substituído por artefato da mesma cor não inspecionada.
7- jogador submete o resultado da correção.
8- SimulES guarda as informações submetidas pelo jogador.
9- SimulES fornece as informações da correção

Titulo: integração de artefatos no módulo
Objetivo: Descrever as regras da integração de artefatos.
Contexto: Localização geográfica: Web
Localização temporal: Java IndividualBoardPage
Pre-condição:
- informações do projeto na página central do jogo.
-jogador tem artefatos no tabuleiro individual.
Pos-condição:
-jogador pode submeter produto.
Atores: SimulES, jogador da vez
Recursos: cartas brancas e/ou cartas cinza, informações do projeto na página principal do jogo.
Episódios:
1- jogador escolhe submissão de produto
2- SimulES escolhe módulo ou módulos do projeto.
3- SimulES seleciona artefatos do tabuleiro, independente do responsável (engenheiro de software) de acordo com as informações de projeto.
4- SimulES integra o módulo
5- SimulES informa integração no modulo.

Titulo: joga rodada de conceitos
Objetivo: Descrever as regras da rodada de conceitos.
Contexto: Localização geográfica: Web
Localização temporal: Java PlayConceptsRoundPage.
Pre-Condição: apos de terminada joga rodada de ações.
Atores: jogador, adversário Recursos: cartão de projeto na página principal do jogo, engenheiros de software que pertence ao jogador, jogador com cartas conceito e cartas problema pertence ao jogador
Episódios:
1- jogador aplica conceitos caso sejam permanentes, colocando-os ao lado do tabuleiro.
Restrição: Não pode exceder o orçamento disponível no projeto.
2- Se jogador tem cartas de engenheiro de software, então jogador contrata engenheiro de software de acordo com o orçamento disponível (que consta nas informações do projeto), a contratação será reflexa no tabuleiro individual.
3- jogador descarta cartas, SimulES retorna estas aos montes apropriados.
4- jogador escolhe até 3 concorrentes e pode submeter para cada um uma carta de problema.
Restrição: concorrente não pode ter mais de 2 cartas de problemas permanentes e mais de 3 cartas de problemas temporários.
5- problema que submete aos demais concorrentes são visíveis para todos os jogadores.
Restrição: Quem escolhe as cartas do concorrente afetadas pelos problemas impostos é o próprio jogador.

Titulo: tratamento de problema
Objetivo: Descrever as regras do tratamento de problemas.
Contexto: Localização geográfica: Web
Localização temporal: Java PlayConceptsRoundPage.
Pre-condição:
- informações do projeto na página principal do jogo.
- Todos os jogadores terminaram joga rodada de conceitos.
- jogador recebeu cartas problema.
- O tratamento de problema é visível para todos os jogadores
Pós-condição:
- problemas tratados.
Atores: jogador, SimulES, adversário.
Recursos: Cartas conceito, cartas problema, tabuleiro individual, engenheiro de software
Episódios:
1- jogador estuda como atender a demanda da carta de problema colocada pelo concorrente.
2- jogador atende a demanda da carta de problema, escolhendo as opções que tenha para tratar o problema.
Restrição: A carta problema pode ser uma penalidade, ou seja,
se o problema é temporário então jogador descarta a carta de problema. Restrição: carta de problema pode ser contraposta por carta de conceito.
Restrição: Se o problema é permanente então jogador mantém carta de problema.
Restrição: se carta de problema pode ser contraposta por carta de conceito. jogador pode demitir um engenheiro de software, descartando-o.
3- O jogador submete seu tratamento de problema.
4- SimulES atualiza as informações de tratamento de problema.
5- jogadores aceitam o rejeitam o tratamento dado por o jogador ao problema em questão.
6- SimulES informa sobre o conceito dos jogadores sobre o problema tratado

Titulo: submissão de produto
Objetivo: Descrever as regras da submissão de produto.
Contexto:
Localização geográfica: Web
Localização temporal: Java IndividualBoardPage
Pre-condição:
- informações do projeto na página principal do jogo.
-construção do produto
- execução do cenário integração de artefatos no módulo.
Pos-condição:
- pode se ganhar o jogo
- pode se terminar a partida
Atores: jogador, SimulES, adversário. Recursos: cartas brancas, cartas cinzas, informações do projeto na página principal do jogo, módulos.
Episódios:
1- jogador mostra que produziu todos os módulos, de acordo com as informações de projeto.
2- SimulES verifica todos os artefatos de x módulos, onde x é o nível de qualidade do projeto.
Restrição: concorrente não pode selecionar módulo protegido por carta conceito do jogador.
3- SimulES fornece as informações da inspeção aleatória feita.
4- concorrente pode inspecionar aleatoriamente artefatos.
Pos-condição: Se artefatos escolhidos (desvirados) forem livres de defeito, então jogador ganha o jogo. Se artefatos escolhidos (desvirados) forem defeituosos, então jogador pode corrigir um por turno de jogo

SDsituations Auxiliares

Titulo: SDsituations apresentar dinâmica do jogo
Objetivo: Descrever e disponibilizar as informações gerais do jogo.
Contexto: Localização geografica: Web
Localização temporal: Java index
Precondições: Serviços Web disponíveis
Poscondições: Informações gerais do jogo disponíveis Atores: jogador, SimulES
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:
1 – jogador entra no jogo
2 – jogador pode trocar mensagens com os jogadores;
3 – SimulES disponibiliza as informações do projeto escolhido;
4 – SimulES disponibiliza as informações de aceitação de movimentos;
5 – SimulES disponibiliza as mensagens trocadas entre os jogadores;
6 – SimulES disponibiliza as informações dos movimentos do jogo;
Restrições: tempo de resposta adequado

Titulo: SDsituations gestão de material de apoio
Objetivo: Descrever as regras do material de apoio.
Contexto:
Localização: Web Java: AdminPage.
jogador administrador cria tipos de material de apoio e fornece as informações do material de apoio que serão utilizadas nas cartas conceito cartas problema.
Atores: jogador administrador
Recursos: informações do projeto, cartas brancas, cartas cinzas, engenheiros de software, jogadores, movimentos do jogo.
Episódios:
1- jogador administrador solicita a SimulES inicializar as variáveis do jogo.
2- SimulES inicializa todas a variáveis do jogo.
3- SimulES fornece o resultado a inicialização.
4- jogador administrador estabelece uma nova sessão de jogo.
5- jogador administrador fecha a entrada de novos jogadores na sessão
Precondição: jogadores cadastrassem no jogo (joga rodada de inicio).
6- jogador administrador escolhe o material de suporte para o jogo.
7- SimulES cadastra material de apoio que será utilizado no jogo.
8- jogador solicita que jogo seja fechado
Precondição: jogador submete produto e gana o jogo (submissão de produto).
9- SimulES fecha o jogo.
10-Informa para todos os jogadores que jogo foi finalizado.

Titulo: gestão do jogo
Objetivo: Descrever as regras da gestão do jogo.
Contexto:
Localização: Web Java: AdminPage.
jogador administrador inicializa as variáveis do jogo e fecha a entrada de jogadores para iniciar o jogo.
Atores: jogador administrador Recursos: informações do projeto, cartas brancas, cartas cinzas, engenheiros de software, jogadores, movimentos do jogo.
Episódios:
1- jogador administrador solicita a SimulES inicializar as variáveis do jogo.
2- SimulES inicializa todas a variáveis do jogo.
3- SimulES fornece o resultado a inicialização.
4- jogador administrador estabelece uma nova sessão de jogo.
5- jogador administrador fecha a entrada de novos jogadores na sessão
Precondição: jogadores cadastrassem no jogo (joga rodada de inicio).
6- jogador administrador escolhe o material de suporte para o jogo.
7- SimulES cadastra material de apoio que será utilizado no jogo.
8- jogador solicita que jogo seja fechado
Precondição: jogador submete produto e gana o jogo.
9- SimulES fecha o jogo
10- SimulES zera as variáveis do jogo
11-Informa para todos os jogadores que jogo tem finalizado

6.4.4.Refinamento das SDsituations e dos Cenários

Para o refinamento das SDsituations e cenários começamos pela análise dos atributos de transparência que deviam estar presentes nos modelos e refletidos na implementação, para isso fizemos uma análise dos atributos da transparência propostos em [53] e que se encontram detalhados em [22]. Estes foram tratados

conjuntamente com os atributos de qualidade para aplicações Web propostos em [66] como atributos mais importantes a serem levados em consideração no momento de abordar um desenvolvimento Web. Segundo [66] estes atributos são: confiabilidade, usabilidade, segurança, disponibilidade, escalabilidade, manutenção e tempo para o mercado, atributos que de uma ou outra maneira são tratados pela transparência de software, como resultado desta análise apresentamos os termos da transparência extraídos do LEL para o SimulES-W:

Nome:	acessibilidade
Noção:	capacidade de obtenção. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- refere-se à capacidade de acessar o **jogo** e seu conteúdo para todos os **jogadores** que interagem com ele. - é composta de **portabilidade**, **disponibilidade** e **publicidade**.
Sinônimo(s):	

Nome:	portabilidade
Noção:	capacidade de ser usado em diferentes ambientes. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **acessibilidade**. - permitir acesso ao **jogo** não importando a plataforma do **jogador**. - permitir acesso às informações do **jogo** não importando a plataforma do interessado. - executar o **jogo** em diferentes plataformas. **contribuição**: confiança no uso do produto.
Sinônimo(s):	

Nome:	disponibilidade
Noção:	capacidade de ser utilizado no momento em que se fizer necessário. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **acessibilidade**. - disponibilizar o **jogo** para que seja publico. - disponibilizar a documentação do **jogo**. **contribuição**: satisfação do usuário por ter o produto utilizável.
Sinônimo(s):	

Nome:	publicidade
Noção:	Capacidade de se tornar público. Observação: RNF **atributo de transparência**.
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **acessibilidade**. - indexar os cenários do **jogo** por palavra chave. - criar um menu principal para disponibilizar aos cenários principais do **jogo**. - disponibilizar informações de ajuda sobre o **jogo**. - construir um catálogo de uso do **jogo**. **contribuição**: divulgação do conhecimento e experiências.
Sinônimo(s):	

Nome:	auditabilidade
Noção:	capacidade de exame analítico. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- é composto de **validade**, **controle**, **verificabilidade**, **rastreabilidade** e **explicação**.
Sinônimo(s):	

Nome:	validade
Noção:	capacidade de ser testado por experimento ou observação para identificar se o que está sendo feito é correto. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **auditabilidade**. - relacionar modelos x código - testar o codificado. - relacionar objetivos do software X as expectativas dos usuários. - simular o processo. - identificar fontes das informações. **contribuição**: futuras evoluções do **jogo**. **contribuição**: testar o **jogo**.
Sinônimo(s):	

Nome:	verificabilidade
Noção:	capacidade de identificar se o que está sendo feito é o que deve ser feito. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **auditabilidade**. - relacionar objetivos e o **jogo**. - relacionar entradas saídas e atividades. - relacionar eventos e processos. - relacionar atores e atividades. - relacionar informações e instâncias dos cenários. - relacionar requisitos de software e implementação. - fornecer descrições dos cenários do **jogo**. **contribuição**: requisitos x implementação. **contribuição**: testar o **jogo**.
Sinônimo(s):	

Nome:	rastreabilidade
Noção:	capacidade de seguir o desenvolvimento de uma ação ou a construção de uma informação, suas mudanças e justificativas. Observação: RNF **atributo de transparência**.
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **auditabilidade**. - identificar requisitos de software X implementação. - identificar contexto da mudança. - identificar quando sã feitas mudanças. - identificar local da mudança. - identificar informações X instâncias dos cenários. - identificar motivo das mudanças. - identificar precondições na implementação ou cenário. - identificar poscondições na implementação ou cenário. - identificar as mudanças nos **artefatos** e no código. - identificar dependências entre cenários. **contribuição**: futuras evoluções do **jogo**. **contribuição**: modelos x código. **contribuição**: **entendimento** de precondições e poscondições da dinâmica do **jogo**.

Nome:	explicação
Noção:	capacidade de informar a razão de algo. Observação: RNF **atributo de transparência**.
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **auditabilidade**. - identificar como os objetivos do **jogo** são atingidos. - identificar a seqüência lógica do **jogo**. - identificar como os eventos e quais finalizam o **jogo**. - justificar necessidade de entradas. - documentar decisões existentes dentro do **jogo**. - justificar decisões existentes dentro do **jogo**. - justificar as saídas geradas nas atividades do **jogo**. - justificar a seqüência de atividades do **jogo**. - identificar requisitos de software X funcionalidades implementadas. - justificar organização da estrutura do **jogo**. - identificar definição do conteúdo das informações. - justificar necessidade do uso das informações. - justificar existência de cada atividade dentro do progresso do **jogo**. - justificar necessidade de cada ator envolvido no **jogo**. **contribuição**: **entendimento** das partes do **jogo**. **contribuição**: avaliar o nível de atendimento dos objetivos do **jogo**. **contribuição**: avaliar a aceitação do **jogo**.
Sinônimo(s):	

Nome:	entendimento
Noção:	capacidade de alcançar o significado e o sentido.
Classificação:	estado
Impacto(s):	é composto de **concisão**, **composição**, **divisibilidade**, **dependência** e **detalhamento**
Sinônimo(s):	

Nome:	concisão
Noção:	capacidade de ser resumido. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **entendimento**. - contribui muito negativamente com **completeza**. - resumir a documentação do **jogo**. - resumir passos na implementação. - escolher informações a serem apresentados aos diferentes usuários. - reduzir nível de granularidade para simplificar o **entendimento** do **jogo**. **contribuição**: agiliza o treinamento para entender o **jogo**. **contribuição**: positiva para o **entendimento** dos documentos gerados.
Sinônimo(s):	

Nome:	composição
Noção:	capacidade de construir ou formar a partir de diferentes partes. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **entendimento**. - possibilidade de reunir partes da implementação. - reunir informações criadas. - relacionar as partes. - relacionar informações (documentação e código). **contribuição**: positiva para futuras evoluções do **jogo**. **contribuição**: positiva para o **entendimento** da dinâmica do **jogo** por parte dos interessados.
Sinônimo(s):	

Nome:	divisibilidade
Noção:	capacidade de ser particionado. Observação: RNF **atributo de transparência**.
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **entendimento**. - utilizar padrões de projetos. - decompor em classes a implementação do **jogo**. - identificar regras de formação das informações. - decompor informações. **contribuição**: futuras evoluções do **jogo**. **contribuição**: **entendimento** das partes do **jogo**.
Sinônimo(s):	

Nome:	dependência
Noção:	capacidade de identificar a relação entre as partes de um todo. Observação: RNF **atributo de transparência**.
Classificação:	estado
Impacto(s):	- ajudar a satisfazer do conceito de **entendimento**. - contribui muito positivamente com **rastreabilidade**. - na implementação do **jogo** identificar atividades que o compõem. - identificar interfaces na implementação. - identificar **composição** das informações. - identificar relações entre informações documentação x código. **contribuição**: futuras evoluções do **jogo**.
Sinônimo(s):	

Nome:	detalhamento
Noção:	capacidade de descrever em minúcias. Observação: RNF **atributo de transparência**.
Classificação:	estado
Impacto(s):	- ajuda a satisfazer o conceito de **entendimento**. - nomear conforme os usuários alvo. - comentar o código adequadamente. - Usar taxonomia de dados(variáveis)/funções (comandos) **contribuição**: **entendimento** da dinâmica do **jogo**. **contribuição**: **entendimento** dos diferentes documentos do **jogo**.
Sinônimo(s):	

Nome:	informativo
Noção:	capacidade de prover informações com qualidade. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	é composto de **clareza**, **completeza**, **corretude**, **consistência**, **integridade**, **acurácia**, **comparabilidade**, **atualidade**.
Sinônimo(s):	

Nome:	clareza
Noção:	Capacidade de nitidez e compreensão. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **informativo**. - usar termos no **jogo** pertencentes ao domínio da engenharia de software. - oferecer fontes alternativas de informação que possam ser abordadas no **jogo**. - possuir descrições para os **jogadores**. - fornecer ajuda sobre execução das rodadas. - tratar somente atividades pertencentes ao objetivo do **jogo** engenharia de software. - detalhar informações **contribuição**: positiva para identificar os objetivos do **jogo**. **contribuição**: positiva frente ao que o **jogo** pretende ensinar.
Sinônimo(s):	

Nome:	completeza
Noção:	Capacidade de não faltar nada do que pode ou deve ter. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **usabilidade**. - identificar se o conjunto de atividades atinge o objetivo do **jogo**. - verificar se jogos similares tem as mesmas atividades. - possuir todas as atividades necessárias a sua execução. - verificar se o **jogo** cumpre as expectativas dos **jogadores**. **contribuição**: positiva medida em níveis de aprendizado dos **jogadores**.
Sinônimo(s):	

Nome:	corretude
Noção:	capacidade de ser isento de erros. Observação: RNF **atributo de transparência**.
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **informativo**. - contribui muito positivamente com **integridade**. - O **jogo** deve ter capacidade de cumprir com a sintaxe do ambiente onde esta sendo desenvolvido. - Verificar que as pré-condições estejam estabelecidas, as pós-condições sejam atendidas, aquelas que foram definidas nas SDsituations. **contribuição**: futuras evoluções do **jogo**. **contribuição**: relacionar documentação e código. **contribuição**: avaliar requisitos e implementação.
Sinônimo(s):	

Nome:	consistência
Noção:	capacidade de resultado aproximado de várias medições de um mesmo item. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **informativo**. - verificar padrões implementados. - comparar etapas do processo com dados armazenados. - comparar resultados com objetivos esperados. - comparar nome das atividades com suas descrições. **contribuição**: futuras evoluções do **jogo**. **contribuição**: avaliar objetivos do **jogo**.
Sinônimo(s):	

Nome:	integridade
Noção:	capacidade de ser correto e imparcial. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **informativo**. - fornecer as informações do **jogo** de forma imparcial. - apresentar fontes das informações. **contribuição**: equitativo com todos os interessados. **contribuição**: gera confiança nos interessados.
Sinônimo(s):	

Nome:	acurácia
Noção:	capacidade de isenção de erros sistemáticos. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **informativo**. - verificar se os resultados atendem os objetivo do **jogo**. - verificar possibilidade de erros na geração das informações do **jogo**. - **contribuição**: gera confiança no uso do produto software do **jogo**
Sinônimo(s):	

Nome:	comparabilidade
Noção:	capacidade de ser comparado. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **informativo**. - definir parâmetros de comparação para informações fornecida aos **jogadores**. - definir padrões de execução de atividades. - definir padrões de comparação com outros jogos de ensino na engenharia de software. **contribuição**: competitividade do **jogo** frente a outros jogos.
Sinônimo(s):	

Nome:	atualidade
Noção:	Capacidade de estar no estado atual. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **informativo**. - manter o software do **jogo** associado com informações de versionamento/configuração - apresentar as fontes de informação. - permitir **controle** de versão da aplicação. - permitir **controle** de versão das informações. **contribuição**: positiva para futuras evoluções do **jogo**. **contribuição**: **jogo** competitivo frente a outros jogos implementados.
Sinônimo(s):	

Nome:	uniformidade
Noção:	capacidade de manter uma única forma. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **usabilidade**. - contribui pouco positivamente para **clareza**. - padronizar notação tanto na implementação como na documentação. - usar padrões de projeto na implementação. - padronizar sintaxe e semântica nos modelos e no código. - padronizar as interfaces. - padronizar cores. - padronizar tipos de diagramas. - definir termos do domínio. - padronizar forma das descrições. **contribuição**: futuras evoluções do **jogo**. **contribuição**: modelos x código. **contribuição**: **entendimento** dos documentos e implementação do **jogo**.
Sinônimo(s):	

Nome:	simplicidade
Noção:	capacidade de não apresentar dificuldades ou obstáculos. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **usabilidade**. - contribui muito negativamente com **completeza**. - contribui muito positivamente com **concisão** - padronizar a nomenclatura dos labels - padronizar documentos utilizados. - possuir fontes de informações unificadas. - possuir número reduzido de níveis. **contribuição**: **entendimento** do **jogo**. **contribuição**: permite atingir os objetivos do **jogo**.
Sinônimo(s):	

Nome:	operabilidade
Noção:	capacidade de estar operacional. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **usabilidade**. - descrever os objetivos do **jogo**. - descrever os objetivos das rodadas do **jogo**. - descrever os objetivos das rondas do **jogo**. - descrever principais funções do **jogo**. - definir os elementos utilizados. - indicar interfaces com outros elementos. - indicar cenários e subcenários. - identificar elementos X atividades. **contribuição**: confiança no uso do produto.
Sinônimo(s):	

Nome:	intuitividade
Noção:	capacidade de ser utilizado sem aprendizado prévio. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **usabilidade**. - ter representação gráfica dentro da implementação do **jogo**. - dar nome a cada atividade executadas no **jogo**. - usar termos pertencentes ao domínio. - usar símbolos que representem graficamente as atividades do **jogo**. **contribuição**: agiliza o processo de aprendizado. **contribuição**: objetivos do **jogo** atingidos.
Sinônimo(s):	

Nome:	desempenho
Noção:	capacidade de operar adequadamente. Observação: RNF **atributo de transparência**.
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **usabilidade**. - usar indexadores. - possuir espaço para armazenamento das informações. - **tempo de resposta adequado**. **contribuição**: satisfação do usuários em quanto a uso do software.
Sinônimo(s):	

Nome:	adaptabilidade
Noção:	Capacidade de mudar de acordo com as circunstâncias e necessidades. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **usabilidade**. - modificar elementos no **jogo** que conduza a um objetivo especifico determinado pelo instrutor. - **contribuição**: positiva para os objetivos traçados dentro do **jogo**. - **contribuição**: positiva para futuras evoluções do **jogo**.
Sinônimo(s):	

Nome:	amigabilidade
Noção:	Capacidade de utilização sem esforço. Observação: RNF **atributo de transparência**
Classificação:	estado
Impacto(s):	- ajudar a satisfazer o conceito de **usabilidade**. - validar a representação gráfica do **jogo**. - oferecer informações sobre cada atividade executada. **contribuição**: positiva para aceitação do **jogo** em modo digital
Sinônimo(s):	

Com o jogo focado nos atributos de transparência, retomamos os modelos SD e modelos SR e identificamos a pertinência de incorporar alguns dos termos da transparência acima descritos nesta etapa da modelagem.

Como os atributos de transparência são tratados como requisitos não funcionais eles geralmente estão presentes no sistema todo de forma transversal, porém apresentamos um modelo indicando como eles foram incorporados após a análise.

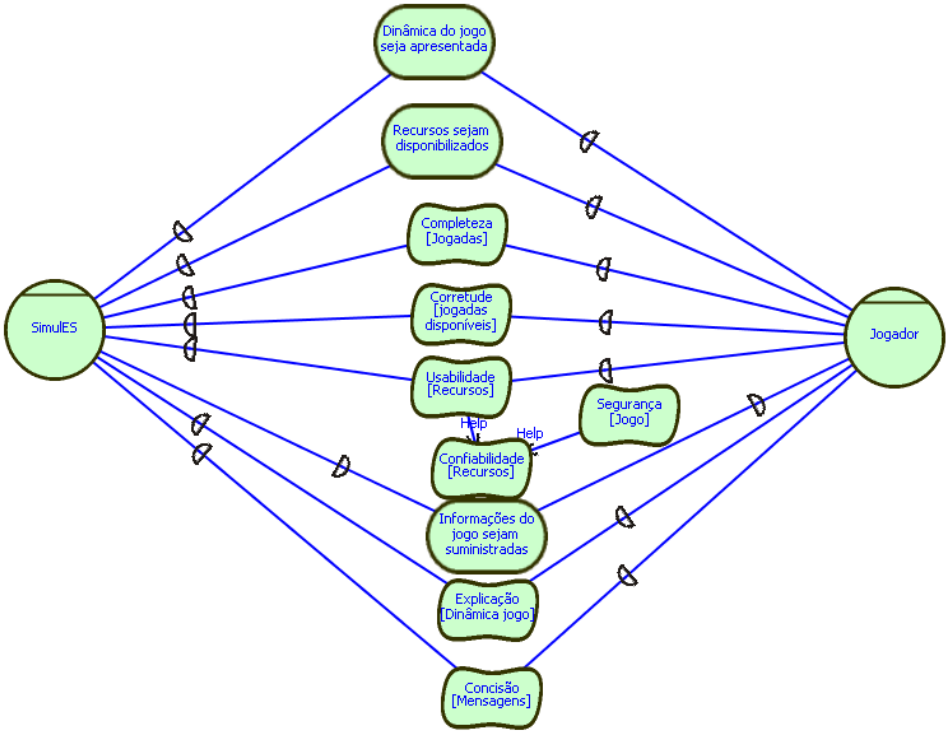


Figura 77 – Modelo SD – Apresentar Dinâmica do Jogo com atributos da Transparência.

Como ilustramos na Figura 77 alguns elementos de transparência foram acrescentados no modelo SD *Completeza*, *Corretude*, *Explicação* e *Concisão* ajudando atingir os objetivos do jogo e o entendimento do mesmo. Eles são expressos como requisitos não funcionais dentro da modelagem.

Como vemos na Figura 78 a usabilidade é um item importante dentro da transparência de software do nosso produto, ele cria a ligação produto e usuário, onde usuário (jogador) será quem avaliará o produto (SimulES) segundo seu uso.

Os atributos de transparência são importantes em nosso empreendimento já que eles contribuem para atingir a facilidade do aprendizado, facilidade para realizar as tarefas, rapidez ao desenvolver as tarefas para o qual o produto foi desenvolvido, e o mais importante a satisfação do usuário *a ser informado sobre o que o jogo faz*, retomando a premissa da transparência de software.

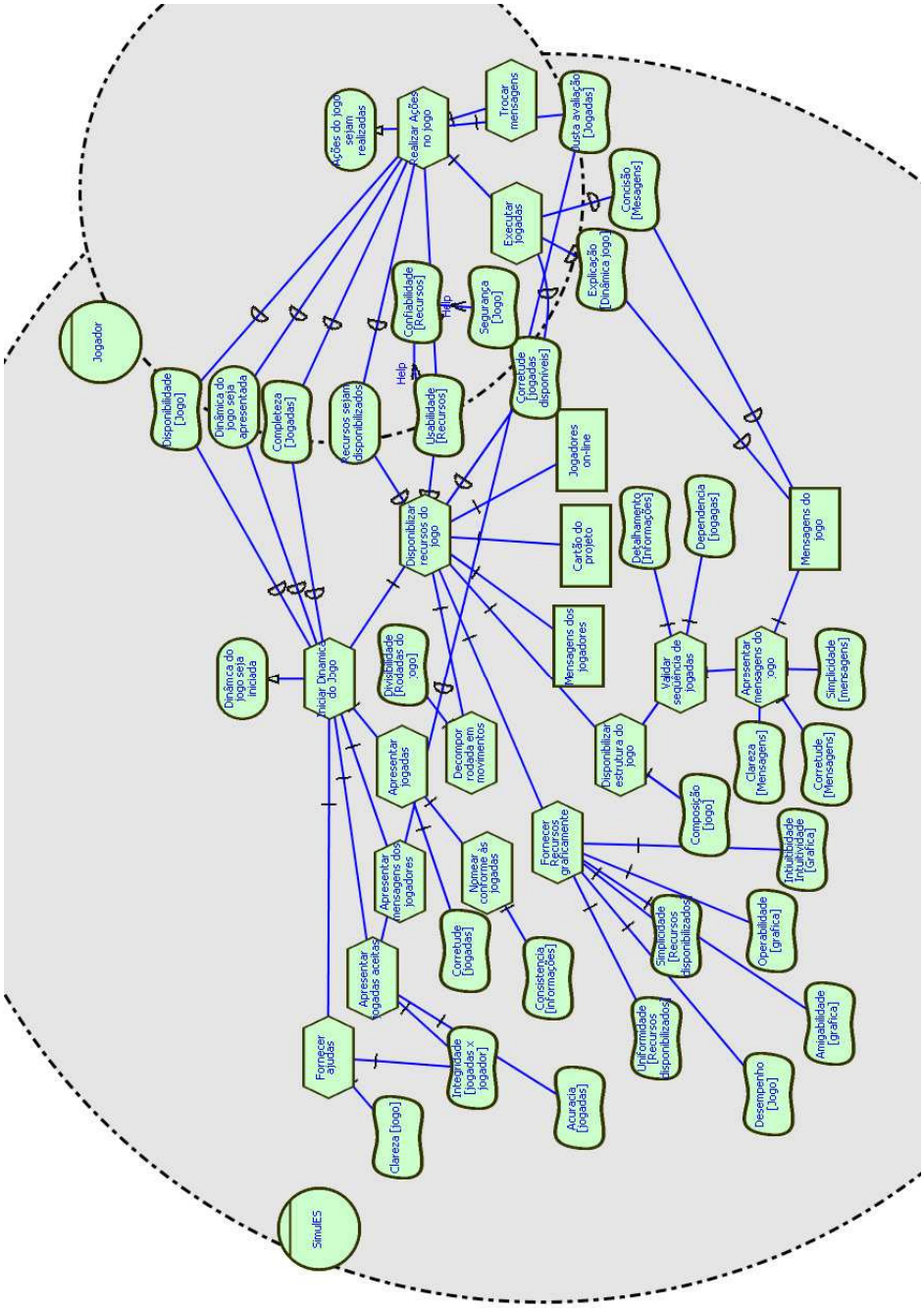


Figura 78 – Modelo SR – Apresentar Dinâmica do Jogo com atributos da Transparência.

A seguir mostramos a descrição mediante cenários da SDsituation *Apresentar Dinâmica do Jogo (*relacionada à Figura 78*)* refinada segundo a análise dos atributos de transparência. Acreditamos na pertinência da incorporação dos atributos da transparência em nosso trabalho, pois elas ajudam a

garantir um produto de software que satisfaze as expectativas expressas pelos usuários. Concordamos com [11] quando expõe que "as metas flexíveis qualificam os elementos da dependência, como conseqüência os elementos desta dependência não podem ser considerados sem aquela qualificação. Como resultado a meta flexível indica que uma operacionalização deve ser desenvolvida".

Operacionalizações das metas flexíveis serão considerados no item *Operacionalização das SDsituations (6.6).*

Titulo: SDsituation apresentar dinâmica do jogo
Objetivo: Descrever e disponibilizar as informações gerais do jogo.
Contexto:
Localização geográfica: Web
Localização temporal: Java index
Pre-condições: Serviços Web disponíveis
Pós-condições: Informações gerais do jogo disponíveis
Atores: simules, jogador
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:
1 – jogador entra no jogo
2 – jogador pode trocar mensagens com os jogadores;
3 - jogador executa jogadas no jogo;
4 – simules disponibiliza as informações do projeto escolhido;
5 - simules da nome às atividades do jogo;
6 - simules decompõe as atividades do jogo;
7 - simules valida seqüência das jogadas;
8 – simules disponibiliza as informações de aceitação de movimentos;
9 – simules disponibiliza as mensagens trocadas entre os jogadores;
10 – simules disponibiliza as informações dos movimentos do jogo;
Restrições: tempo de resposta adequado

Metas flexíveis:
completeza [Jogadas]
corretude [jogadas disponíveis]
explicação [Dinâmica jogo]
concisão [Mensagens]
detalhamento [Informações]
dependência [jogadas]
divisibilidade [Rodadas do jogo]
simplicidade [mensagens]
composição [jogo]
intuitividade [gráfica]
operabilidade [gráfica]
simplicidade [Recursos disponibilizados]
corretude [jogadas]
consistência [informações]
uniformidade [Recursos disponibilizados]
acurácia [jogadas aceitas]
clareza [jogo]
desempenho [jogo]
integridade [jogadas x jogador]

6.5.Cenários do Sistema

A escolha do MVC como arquitetura para O SimulES permitiu a seguinte divisão:

Cada SDsituations tem seu correspondente modelo SR que também possui sua representação em cenário, sua operacionalização situa-se na camada de visão.

Metas e tarefas têm sua representação nos modelos SR que também possuem sua representação como símbolos tipo verbo, e suas operacionalizações estão na camada de controle.

Recursos têm sua representação nos modelos SR e também possuem sua representação como símbolos tipo Sujeito, suas operacionalizações estão na camada de modelo.

6.5.1.Cenários da Camada de Controle

Aqui estão representados os cenários intermediários entre os cenários da camada de visão e os cenários da camada modelo, estes cenários se encarregam da parte lógica responsável do processamento e o comportamento de acordo com às demandas do usuário, constrói o modelo apropriado e o envia para a camada de visão para sua adequada visualização.

Titulo: Aceitar Movimentos
Objetivo: Descrever as ações necessárias para aceitar movimentos.
Localização geográfica: Web
Localização temporal: Java AcceptmoveController
Pre-condições: Serviços Web disponíveis, Jogada de conceitos em execução
Pós-condições: Informações gerais do jogo disponíveis
Atores: jogador, simules
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:

- obter todos os movimentos aceitados
- obter movimento aceitado
- valida que jogador não aceita seu próprio movimento
- adicionar avaliação do movimento realizado
- apagar movimento

Titulo: Controlador de cartas
Objetivo: Descrever as operações de controle nas cartas.
Localização geográfica: Web
Localização temporal: Java CardController
Pre-condições: Serviços Web disponíveis, rodada de inicio já foi executada
Pós-condições: Atualizar estado das cartas no jogo e para jogador.
Atores: jogador, simules
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:

- obter cartas disponíveis
- obter carta especifica
- obter cartas livres por material de apoio
- obter todas as cartas
- apagar todas as cartas do repositório
- adicionar cartas no repositório
- atualizar cartas

Titulo: Controlador de tabuleiro individual
Objetivo: Descrever as operações de controle no tabuleiro individual.
Localização geográfica: Web
Localização temporal: Java IndividualboardController
Pre-condições: Serviços Web disponíveis, rodada de ações em execução
Pós-condições: Atualizar o tabuleiro individual do jogador.
Atores: jogador, simules
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:

- criar o tabuleiro individual para o jogador
- obter os tabuleiros individuais dos jogador
- obter a configuração do tabuleiro individual do jogador
- validar se jogador está registrado no jogo para disponibilizar
- atualiza a configuração do tabuleiro do jogador
- obter o tabuleiro individual por jogador
- apaga os tabuleiros individuais dos jogadores

Titulo: Controlador dos módulos do projeto
Objetivo: Descrever as operações de controle nos módulos do projeto.
Localização geográfica: Web
Localização temporal: Java ModulesProjectController
Pre-condições: Serviços Web disponíveis, projeto para o jogo já escolhido
Pós-condições:
Atores: jogador, simules
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:

- obter os módulos de um projeto especificado
- obter os módulos do projeto com estado escolhido
- obter os módulos de um projeto organizados em um arranjo
- inspecionar módulos especificados
- obter os módulos por um identificador

Titulo: Controlador de movimentos no jogo
Objetivo: Descrever as operações de controle nos movimentos do jogo.
Localização geográfica: Web
Localização temporal: Java MoveController
Pre-condições: Serviços Web disponíveis, rodada de conceitos em execução
Pós-condições:
Atores: jogador, simules
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:

- obter todos os movimentos
- atualizar o movimento para o jogador
- atualizar o próximo movimento
- obter um movimento especifico
- obter o movimento ativo no sistema
- obter o mínimo movimento ativo no sistema
- reiniciar os movimentos
- obter o primeiro movimento no sistema
- fechar um movimento
- habilitar o próximo movimento
- habilitar próximo movimento para o jogador

Titulo: Controlador de cartas por jogador
Objetivo: Descrever as operações de controle nos módulos do projeto.
Localização geográfica: Web
Localização temporal: Java PlayerCardController
Pre-condições: Serviços Web disponíveis
Pós-condições:
Atores: jogador, simules
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:

- obter todas as cartas por jogador
- apagar todas as cartas por jogador
- adicionar cartas para um jogador
- obter cartas de um jogador especifico
- obter as cartas problemas submetidas a um jogador especifico
- obter as cartas problemas do jogador
- obter as cartas conceito por jogador
- apagar carta especifica do repositório
- obter cartas por jogador

Titulo: Controlador de cartas tipo
Objetivo: Descrever as operações de controle nos tipos de cartas.
Localização geográfica: Web
Localização temporal: Java CardtypeController
Pre-condições: Serviços Web disponíveis, rodada de inicio já foi executada
Pós-condições: Atualizar estado das cartas no jogo e para jogador.
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Atores: simules, jogador
Episódios:

- obter os tipos de cartas

Titulo: Controlador do jogador
Objetivo: Descrever as operações de controle para o jogador.
Localização geográfica: Web
Localização temporal: Java PlayerController
Pre-condições: Serviços Web disponíveis
Pós-condições:
Atores: jogador, simules
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:

- criar jogador
- validar se jogador existe
- obter identificador do jogador
- obter nome do jogador
- obter jogador por nome
- obter jogador por identificador
- obter todos os jogadores
- obter quantidade de jogadores
- apagar jogadores
- obter o Maximo lançamento do dado dos jogadores
- atualizar o valor do lançamento do dado dos jogadores
- obter um rango de jogadores
- obter o ultimo jogador registrado
- obter o primeiro jogador registrado

Titulo: Controlador engenheiros de software por jogador
Objetivo: Descrever as operações de controle entre o jogador e seus engenheiros de software.
Localização geográfica: Web
Localização temporal: Java PlayerSoftengineerController
Pre-condições: Serviços Web disponíveis
Pós-condições:
Atores: jogador, simules
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:

- criar a relação entre engenheiro de software e jogador
- obter todas as relações jogador e engenheiros de software
- obter uma relação engenheiro de software jogador especifica
- obter uma relação engenheiro de software jogador especifica onde engenheiro está em estado contratado
- apagar todas as relações engenheiro de software e jogador
- apagar uma relação engenheiro de software e jogador especifica
- obter um engenheiro de software especifico
- apagar um engenheiro de software de uma relação, especificando engenheiro

Titulo: Controlador problemas por jogador
Objetivo: Descrever as operações de controle entre o jogador e os problemas submetidos.
Localização geográfica: Web
Localização temporal: Java PlayersproblemsController
Pre-condições: Serviços Web disponíveis
Pós-condições:
Atores: jogador, simules
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:

- adicionar problema a jogador
- obter os problemas de um jogador especifico
- obter todos os problemas submetidos
- atualizar as observações dos problemas submetidos
- aceitar tratamento de um problema submetido
- rejeitar tratamento de um problema submetido
- atualizar o tratamento dos problemas
- obter todos as cartas problemas dos jogadores
- obter uma carta problema especifica
- apagar todas as relações cartas problemas x jogador

Titulo: Controlador de projeto
Objetivo: Descrever as operações de controle do projeto do jogo.
Localização geográfica: Web
Localização temporal: Java ProjectController
Pre-condições: Serviços Web disponíveis
Pós-condições: operações sobre o cartão do projeto disponíveis
Atores: jogador, simules
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:

- obter todos os projetos do repositório
- obter um projeto especifico
- obter um projeto por identificador
- atualizar o estado do projeto a escolhido
- reiniciar estado dos projetos
- obter a quantidade de módulos a serem construídos por projeto
- obter projeto escolhido
- obter a qualidade do projeto escolhido

Titulo: Controlador de estados dos engenheiros de software
Objetivo: Descrever as operações de controle sobre os estados dos engenheiros de software.
Localização geográfica: Web
Localização temporal: Java SoftwareEngineerStatusController
Pre-condições: repositório com engenheiros de software
Pós-condições: operações com os diferentes estados dos engenheiros de software
Atores: simules, engenheiros de software
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:

- obter todos os estados dos engenheiros de software
- obter estado de um engenheiro de software especifico

Titulo: Controlador de rondas do jogo
Objetivo: Descrever as operações de controle das rondas do jogo.
Localização geográfica: Web
Localização temporal: Java RoundController
Pre-condições: Serviços Web disponíveis, fechar a entrada de jogadores já executada
Pós-condições: rondas do jogo controladas pelo SimulES
Atores: jogador, simules
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:
- obter todas as rondas do jogo
- reiniciar todas as rondas do jogo
- fechar uma ronda especifica
- obter uma ronda especifica
- habilitar a seguinte ronda

Titulo: Controlador de engenheiros de software
Objetivo: Descrever as operações de controle dos engenheiros de software.
Localização geográfica: Web
Localização temporal: Java SoftEngineerController
Pre-condições: Serviços Web disponíveis
Pós-condições: operações com engenheiros de software disponíveis
Atores: jogador, simules, engenheiros de software
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira
Episódios:
- obter todos os engenheiros de software
- obter um engenheiro de software especifico
- obter engenheiros de software disponíveis
- atualizar o estado dos engenheiros de software para disponíveis
- obter engenheiro de software se está disponível
- empregar engenheiro de software
- reiniciar engenheiros de software

Titulo: Controlador de material de apoio
Objetivo: Descrever as operações de controle sobre o material de apoio.
Localização geográfica: Web
Localização temporal: Java SourceofcardsController
Pre-condições:
Pós-condições: operações com os materiais de apoio disponíveis
Atores: simules, jogador administrador
Recursos: mensageira
Episódios:
- obter todos os materiais de apoio
- apagar material de apoio
- adicionar material de apoio
- atualizar material de apoio
- obter material de apoio por identificador
- obter o Maximo identificador do material de apoio

6.5.2.Cenários da Camada de Modelo

Estes cenários tratam dos objetos ou recursos com os quais o sistema opera. Lembrando que nós representamos aqueles elementos identificados dentro da

modelagem como recurso e que se encontravam dentro do léxico como símbolos tipo sujeito como candidatos a classe e conseqüentemente parte do modelo. Eles ganharam representação em cenários, pois eles apresentam comportamentos como a materialização em objeto e operações com a camada de persistência.

Titulo: Aceitar movimento
Objetivo: Descrever as operações para obter o objeto aceitar movimento.
Localização geográfica: Web
Localização temporal: Java Acceptmove
Pre-condições: Banco de dados deve estar disponível
Pós-condições: Objeto disponível
Atores: simules
Recursos: movimentos
Episódios:
1- receber objeto da camada de controle
2- conectar ao banco de dado
3- enviar objeto à base de dados
4- receber objeto da base de dados

Titulo: Carta
Objetivo: Descrever as operações para obter o objeto carta.
Localização geográfica: Web
Localização temporal: Java Card
Pre-condições: Banco de dados deve estar disponível
Pós-condições: Objeto disponível
Atores: simules
Recursos: carta (identificador, nome, referência, descrição, regra, link), material de apoio, tipo de carta.
Episódios:
1- receber objeto da camada de controle
2- conectar ao banco de dado
3- enviar objeto à base de dados
4- receber objeto da base de dados

Titulo: Tipo de Carta
Objetivo: Descrever as operações para obter o objeto tipo de carta.
Localização geográfica: Web
Localização temporal: Java CardType
Pre-condições: Banco de dados deve estar disponível
Pós-condições: Objeto disponível
Atores: simules
Recursos: tipo de carta(identificador, descrição).
Episódios:
1- receber objeto da camada de controle
2- conectar ao banco de dado
3- enviar objeto à base de dados
4- receber objeto da base de dados

Titulo: Tabuleiro Individual
Objetivo: Descrever as operações para obter o tabuleiro individual.
Localização geográfica: Web
Localização temporal: Java Individualboard
Pre-condições: Banco de dados deve estar disponível
Pós-condições: Objeto disponível
Atores: simules
Recursos: Tabuleiro Individual (identificador, configuração)
Episódios:
1- receber objeto da camada de controle
2- conectar ao banco de dado
3- enviar objeto à base de dados
4- receber objeto da base de dados

Titulo: Módulos do projeto
Objetivo: Descrever as operações para obter os módulos de um projeto.
Localização geográfica: Web
Localização temporal: Java Modulesproject
Pre-condições: Banco de dados deve estar disponível
Pós-condições: Objeto disponível
Atores: simules
Recursos: Módulos (identificador, requisitos, desenho, código, rastro, ajuda), projeto
Episódios:
1- receber objeto da camada de controle
2- conectar ao banco de dado
3- enviar objeto à base de dados
4- receber objeto da base de dados

Titulo: Módulos do projeto
Objetivo: Descrever as operações para obter os módulos de um projeto.
Localização geográfica: Web
Localização temporal: Java Modulesproject
Pre-condições: Banco de dados deve estar disponível
Pós-condições: Objeto disponível
Atores: simules
Recursos: Módulos (identificador, requisitos, desenho, código, rastro, ajuda), projeto
Episódios:
1- receber objeto da camada de controle
2- conectar ao banco de dado
3- enviar objeto à base de dados
4- receber objeto da base de dados

Titulo: Movimento
Objetivo: Descrever as operações para obter os movimentos no jogo.
Localização geográfica: Web
Localização temporal: Java Move
Pre-condições: Banco de dados deve estar disponível
Pós-condições: Objeto disponível
Atores: simules
Recursos: Movimentos (identificador, descrição, precondição, estado), ronda, jogador
Episódios:
1- receber objeto da camada de controle
2- conectar ao banco de dado
3- enviar objeto à base de dados
4- receber objeto da base de dados

Titulo: Jogador
Objetivo: Descrever as operações para obter os jogadores.
Localização geográfica: Web
Localização temporal: Java Player
Pre-condições: Banco de dados deve estar disponível
Pós-condições: Objeto disponível
Atores: simules
Recursos: Jogador (identificador, nome), dado
Episódios:
1- receber objeto da camada de controle
2- conectar ao banco de dado
3- enviar objeto à base de dados
4- receber objeto da base de dados

Titulo: Projeto
Objetivo: Descrever as operações para obter os projetos.
Localização geográfica: Web
Localização temporal: Java Project
Pre-condições: Banco de dados deve estar disponível
Pós-condições: Objeto disponível
Atores: simules
Recursos: Projeto (identificador, nome, descrição, complexidade, orçamento, status, tamanho, qualidade)
Episódios:
1- receber objeto da camada de controle
2- conectar ao banco de dado
3- enviar objeto à base de dados
4- receber objeto da base de dados

Titulo: Ronda
Objetivo: Descrever as operações para obter as rondas do jogo.
Localização geográfica: Web
Localização temporal: Java Round
Pre-condições: Banco de dados deve estar disponível
Pós-condições: Objeto disponível
Atores: simules
Recursos: Ronda (identificador, descrição, status)
Episódios:
1- receber objeto da camada de controle
2- conectar ao banco de dado
3- enviar objeto à base de dados
4- receber objeto da base de dados

Titulo: Engenheiro de Software
Objetivo: Descrever as operações para obter os engenheiros de software do jogo.
Localização geográfica: Web
Localização temporal: Java Softengineer
Pre-condições: Banco de dados deve estar disponível
Pós-condições: Objeto disponível
Atores: simules
Recursos: Engenheiro de Software (identificador, nome, descrição, salário, habilidade, maturidade), Status.
Episódios:
1- receber objeto da camada de controle
2- conectar ao banco de dado
3- enviar objeto à base de dados
4- receber objeto da base de dados

Titulo: Material de apoio **Objetivo**: Descrever as operações para obter material de apoio. **Localização geográfica**: Web **Localização temporal**: Java Sourceofcards **Pre-condições**: Banco de dados deve estar disponível **Pós-condições**: Objeto disponível **Atores**: simules **Recursos**: Material de apoio (identificador, descrição), cartas **Episódios**: 1- receber objeto da camada de controle 2- conectar ao banco de dado 3- enviar objeto à base de dados 4- receber objeto da base de dados

6.5.3.Cenários da Camada de Visão

Usualmente está relacionada com a interface do usuário, e se encarrega de transformar o modelo para que possa ser visualizado, ou seja, converte os dados para que sejam significativos para o usuário e de fácil interpretação. Além disso, se encarrega da interação com o usuário, por isso, os elementos modelados e implementados para esta camada são aqueles que apresentamos em sessões anteriores como os correspondentes às seguintes: *apresentar dinâmica do jogo, construção de artefatos, correção de artefato, gestão de material de apoio, gestão do jogo, inspeção de artefato, integração de artefatos no módulo, joga rodada de ações, joga rodada de conceitos, joga rodada de inicio, submissão de produto e tratamento de problema.*

6.6.Operacionalização das *SDsituations*

No modelo MVC existe uma separação clara entre controlador e a visão. O controlador é quem recebe o pedido. O modelo trata das operações básicas e a visão apresenta a interface. Na Figura 79 é mostrada esta arquitetura MVC refinada e usada para aplicações Web e escolhida para o SimulES-W.

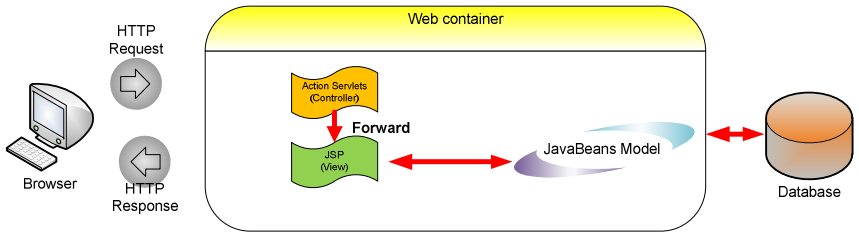


Figura 79 – MVC para aplicações Web [63].

6.6.1.*Frameworks* de Desenvolvimento Web

O termo *Framework* tornou-se popular nos últimos anos dentro do ambiente de desenvolvimento de software. É comum encontrar essa palavra em várias circunstâncias: informações sobre linguagens de programação, informações sobre novas tecnologias Web, evolução de software, qualidade de software entre outras.

- **JavaServer Faces (JSF)**

É um *framework* de interface do usuário do lado do servidor para aplicações Web baseadas na tecnologia Java e no padrão MVC (Modelo Visão Controlador) [63].

- **Visual Web JavaServer Faces**

É um *framework* de J2EE baseado em JSF. Com este *framework* é possível criar páginas Web visualmente. Visual JSF introduz algumas bibliotecas de extensão para dar suporte ao desenvolvimento de aplicações JSF [63].

- **jMaki Ajax *Framework***

É um *framework* de Ajax que fornece um modelo para *widgets* de Ajax reutilizáveis que podem ser desenhados ou estendidos de ferramentas existentes. *jMaki* facilita a passagem de parâmetros para os *widgets* e fornece os meios para uma conexão melhor dos *widgets* com recursos do servidor usando XML ou JSON. Atualmente, o *jMaki runtime* do lado do servidor é fornecido como uma biblioteca de tags JSP ou um componente JSF [63].

- **Hibernate 3.2.5**

Framework de mapeamento de objetos para o modelo relacional. Hibernate pode expressar consultas tanto em sua linguagem de consulta chamada HQL (*Hibernate Query Languaje*) como em SQL [64].

6.6.2.Ferramentas de Desenvolvimento

- **Netbeans 6.5**

Ambiente de desenvolvimento integrado de software livre propriedade da *Sun Microsystems* com estrutura modular que permite trabalhar com diferentes linguagens de desenvolvimento.

- **Servidor de aplicações GlassFish 3.0**

Aponta a camada Web para o serviço de aplicações Web e tem um desenho modular e sua característica principal é que ocupa poucos recursos. Alem disso, é um servidor Web com suporte a servlets y JSPs.

- **MySQL 5.1**

É um sistema de gestão de bases de dados relacional, *multithreading* e multiusuário. Desenvolvido pela *MySQL AB* e comprado pela *Sun MicroSystems*.

6.6.3.Desenvolvimento do SimulES-W

6.6.3.1.Operacionalização dos Cenários da Camada de Visão

Como se falou em sessões anteriores os cenários da camada de visão foram aqueles mapeados das *SDsituations*, eles apresentam aos jogadores as funcionalidades principais do jogo através das telas.

SDsituation apresentar dinâmica do jogo

Esta *SDsituations* é um caso particular pois ela surgiu em resposta à operacionalização do tabuleiro individual, ele passou de ser um recurso a ser um cenário, pois, na implementação ele precisava ter comportamento para atender a troca de mensagens entre jogadores além de fornecer as informações de interesse geral.

Na Figura 80 se tem as mensagens dos jogadores, mensagens do jogo, o projeto escolhido e seus módulos, jogadores on-line e jogadas aceitas que foram recursos identificados na modelagem.

Finalmente, como esta é a tela principal com as informações de interesse para todos os jogadores, serve de tabuleiro principal, onde são centralizadas as informações gerais do jogo.

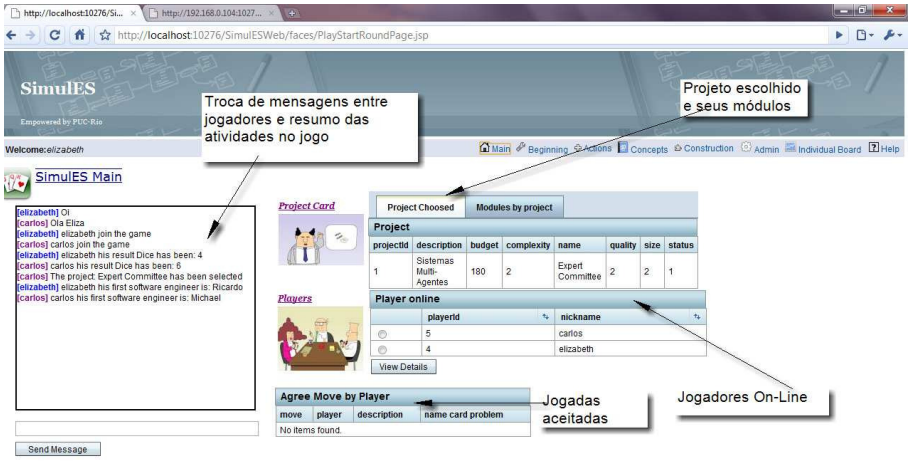


Figura 80 – Tela da *SDsituation* Apresentar Dinâmica do Jogo.

Lembrando que *SDsituations* foram descritas usando técnicas de cenários, na Figura 81 é mostrado como essa descrição chega ao nível do código.

Usaremos a *SDsitutions apresentar dinâmica do jogo* para ilustrar a operacionalização na camada de visão e sua descrição usando cenários.

```
import com.sun.rave.web.ui.appbase.AbstractPageBean;
import com.sun.rave.web.ui.component.Button;
import com.sun.rave.web.ui.component.PanelLayout;
import com.sun.rave.web.ui.component.StaticText;
import com.sun.rave.web.ui.component.TextField;
import com.sun.rave.web.ui.model.DefaultTableDataProvider;
import javax.faces.FacesException;
import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;

/**
 * ********************************************************************
Titulo: SDsituations apresentar dinâmica do jogo
Objetivo: Descrever e disponibilizar as informações gerais do jogo.
Contexto: Localização geografica: Web
Localização temporal: Java index
Precondições: Serviços web disponíveis
Poscondições: Informações gerais do jogo disponíveis Atores: jogador, SimulES
Recursos: informações do projeto, informações dos jogadores, informações dos movimentos e mensageira

public void prerender() {

    //@episodio1 jogador entra no jogo
    String username = (String) getExternalContext().getRequestParameterMap().get("username");
    if (username != null) {
        getSessionBean1().setUsername(username);
    } else if (getSessionBean1().getUsername() == null) {
        getSessionBean1().setUsername(getApplicationBean1().getNextAnonUsername());
    }
    //@episodio2 SimulES disponibiliza as informações do projeto escolhido;
    getApplicationBean1().setProjectChoose(getApplicationBean1().getProjectChoose());
    getApplicationBean1().setModulesProject(getApplicationBean1().getModulesProject());

    getApplicationBean1().setPlayerList(getApplicationBean1().getPlayerList());
    //@episodio3 SimulES disponibiliza as informações de aceitação de movimentos;
    getApplicationBean1().setAcceptmoveList(getApplicationBean1().getAcceptmoveList());
}
```

Descrições iniciais do cenário

```
    //eposidio4  SimulES disponibiliza as mensagens trocadas entre os jogadores;
    public String send_action() {

        String comment = (String) getComment().getText();
        String username = getSessionBean1().getUsername();
        getApplicationBean1().addEntry(username, comment);

        return null;
    }

//episodio5 SimulES disponibiliza as informações dos movimentos do jogo;
 public String getTranscript() {
     String[][] entries =
             getApplicationBean1().getEntries();
     if (entries == null) {
         return null;
     }
     StringBuffer sb = new StringBuffer();
     for (int i = 0; i < entries.length; i++) {
         String entryUsername = entries[i][0];
         String comment = entries[i][1];
         String color = "purple";
         String username = getSessionBean1().getUsername();
         if (username.equals(entryUsername)) {
             color = "blue";
         }
         sb.append(
                 "<div><span style=\"font-weight:bold;color:");
         sb.append(color);
         sb.append("\">[");
         sb.append(entryUsername);
         sb.append("]</span> ");
         sb.append(comment);
         sb.append("</div>");
     }
     return sb.toString();
 }
```

Episódios do cenário

Figura 81 – Parte do código da *SDsituation* Apresentar Dinâmica do Jogo.

SDsituation joga rodada de inicio

Para esta *SDsituation* a interação sistema jogador é maior, na Figura 82 vemos os principais episódios que são executados pelo jogador e processados pelo SimulES, também recursos (dado, projeto, engenheiro de software, mensagens) identificados no modelagem são apresentados aos usuários.

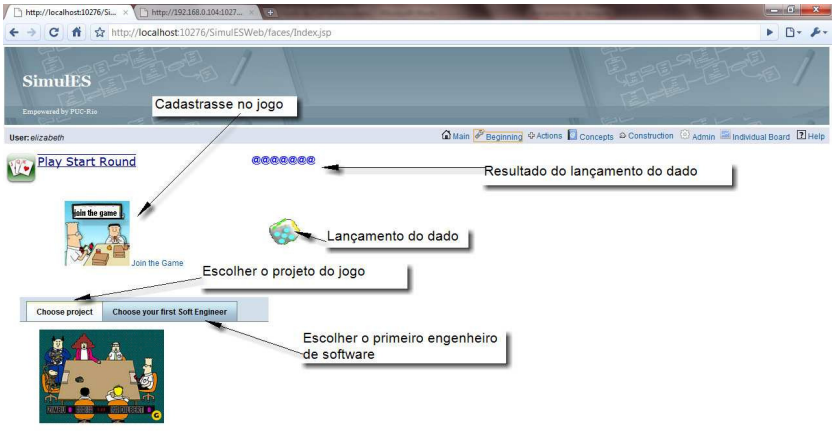


Figura 82 – Tela da *SDsituation* Joga Rodada de Inicio.

SDsituation joga rodada de ações

Em rodada de ações como é apresentado na Figura 83 tem-se o lançamento do dado como um dos episódios a serem executados, o sistema escolhe aleatoriamente um valor entre 1 e 6, depois o resultado é apresentado pelo SimulES para o jogador. Com este resultado o SimulES fornece as cartas conceito e problema e os engenheiros de software que são apresentados nas tabelas para ao jogador.

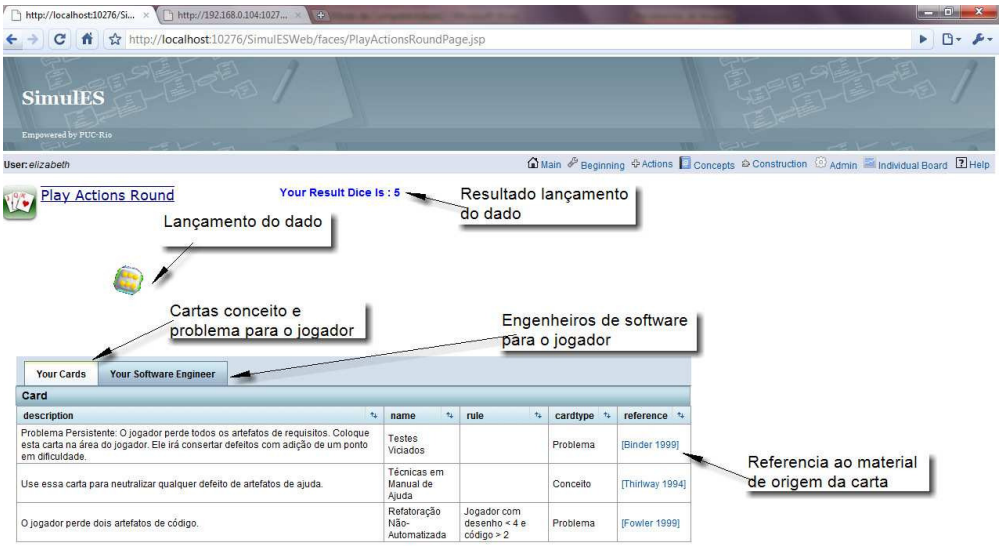


Figura 83 – Tela da *SDsituation* Joga Rodada de Ações.

SDsituation construção de artefatos

A Figura 84 ilustra o tabuleiro individual na *SDsituation* construção de artefato, nesta tela o jogador tem os engenheiros de software contratados para a construção dos diferentes artefatos, cada um deles com um elo que apresenta as informações dos engenheiros de software. A Figura representa as cartas brancas e as cartas cinza sem terem sido desviradas ainda. Também na parte inferior da tela aparecem as operações possíveis dentro do tabuleiro.

A Figura nos mostra que o jogador escolheu construir artefato, para que ação seja finalizada, ele tem que submeter a jogada usando o botão *Submit*.

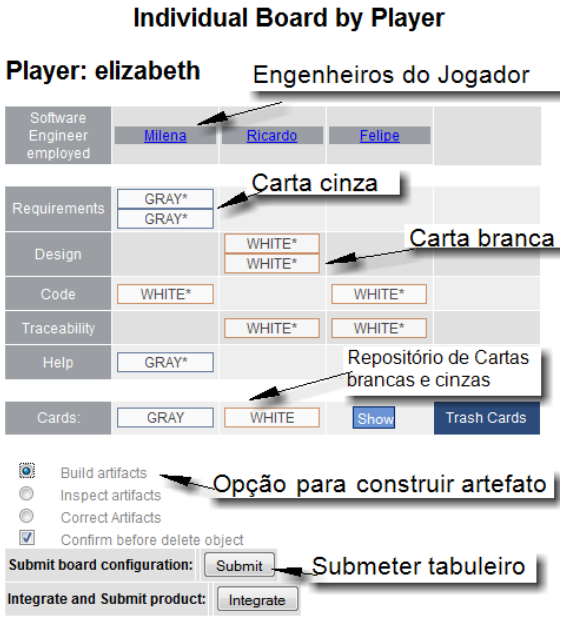


Figura 84 – Tela da *SDsituation* Construção de Artefatos.

SDsituation inspeção de artefato

Quando o jogador já construiu o artefato ele pode inspecioná-lo. Na Figura 85 ilustramos o resultado de uma inspeção quando o artefato tem erro e quando não tem. SimulES aleatoriamente escolhe a qualidade da carta branca ou carta cinza , sendo que as cartas brancas tem menos possibilidades de ter erro que as cartas cinzas.

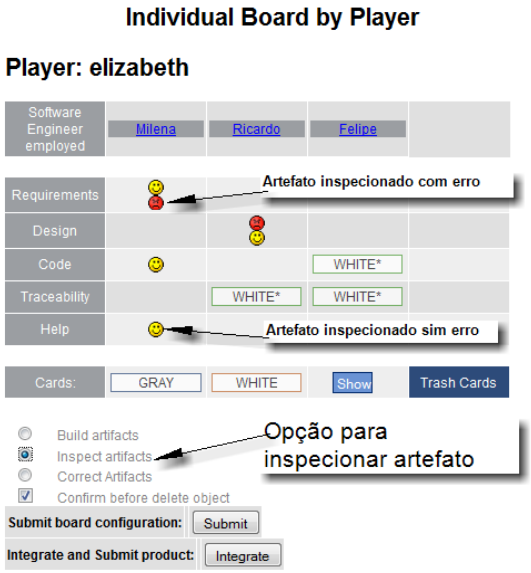


Figura 85 – Tela da *SDsituation* Inspeção de Artefato.

SDsituation correção de artefato

Quando é a vez de corrigir algum artefato defeituoso o jogador leva-o para a caixa de apagar carta (Trash Card) e escolhe do monte de cartas brancas ou cinza a carta a substituir, como é ilustrado na Figura 86.

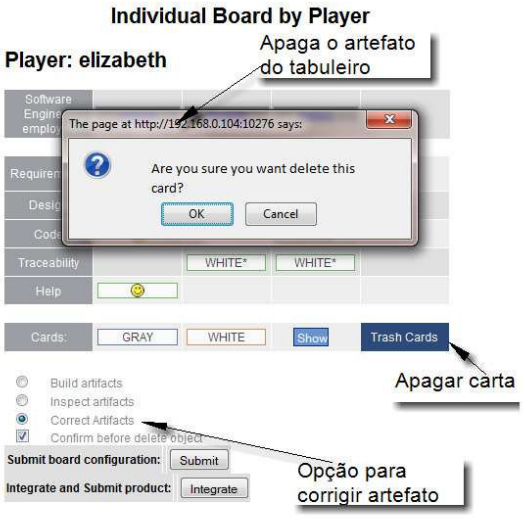


Figura 86 – Tela da *SDsituation* Correção de Artefato.

SDsituation joga rodada de conceitos

A jogada de conceito tem três ações possíveis, como representadas nas Figuras 87, 88, 89. Na primeira (Figura 87) temos as ações com os engenheiros de software. Se jogador escolhe contratar ele aparecera no tabuleiro individual ou se ele escolhe demitir o engenheiro voltara para o repositório ficando disponível para que outro jogador possa contratá-lo.

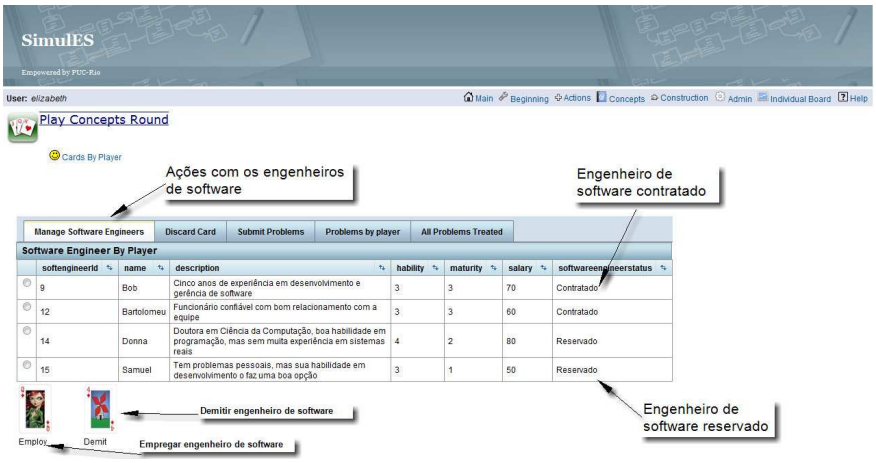


Figura 87 – Tela da *SDsituation* Joga Rodada de Conceitos ações sobre os Engenheiros de Software.

Na Figura 88 ilustramos as cartas conceito e problema que o jogador tem disponível e onde ele pode selecionar para descartar algumas delas, para que estas voltem a ficar disponíveis para que outros jogadores possam escolhê-las.

Figura 88 – Tela da *SDsituation* Joga Rodada de Conceitos ação de Descartar Cartas.

A Figura 89 ilustra como o jogador pode submeter problema para outros jogadores, ele escolhe um dos jogadores on-line para submeter à carta problema, com isso, o jogador escolhido recebe a notificação que uma carta problema foi jogada para ele.

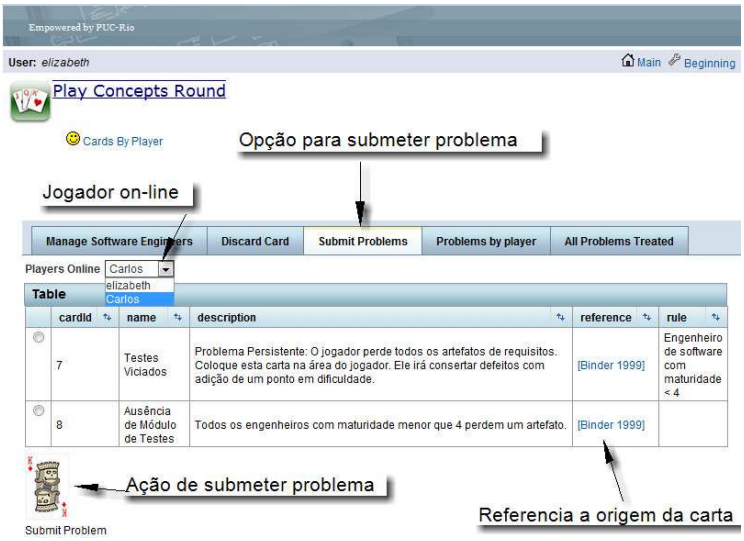


Figura 89 – Tela da *SDsituation* Joga Rodada de Conceitos ação de Submeter Problema.

SDsituation tratamento de problema

Para tratar uma carta problema existem duas opções que estão relacionadas com o conteúdo da carta problema, se tem cartas onde o jogador deve pagar uma penalidade ou se tem cartas que somente podem ser tratadas com cartas conceito.

Na Figura 90 ilustramos os problemas que um jogador tem para serem tratados, ele escolhe um destes e executa a ação tratar problema.

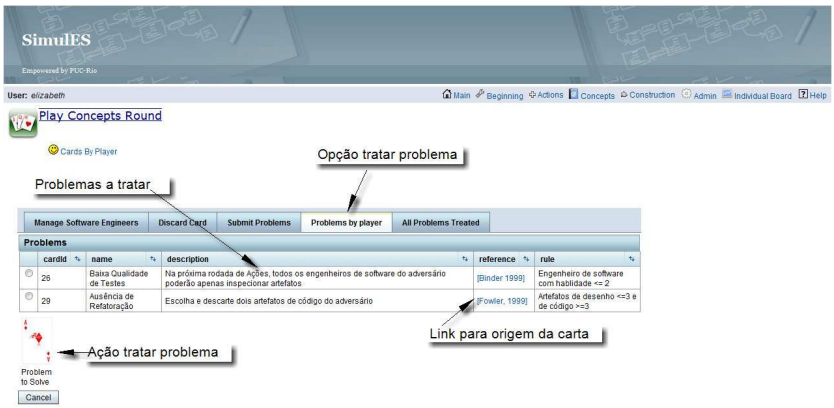


Figura 90 – Tela da *SDsituation* Tratamento de Problema, problemas por jogador.

Na Figura 91 ilustramos o caso no qual o jogador escolhe o tratamento do problema com uma carta conceito, ele submete sua operação para que seja visível para outros jogadores e estes aceitem seu tratamento, caso contrario a carta problema volta para o jogador.

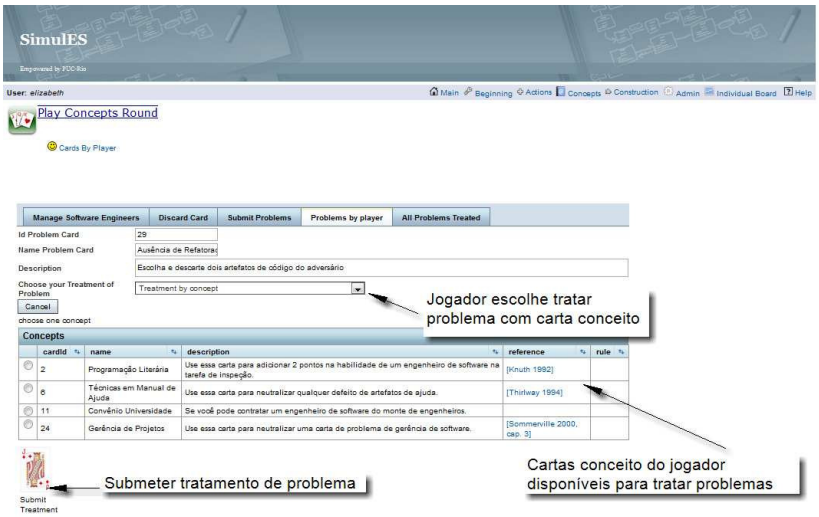


Figura 91 – Tela da *SDsituation* Tratamento de Problema, tratamento com Carta Conceito.

Têm-se os casos no qual o jogador deve pagar uma penalidade com a carta problema imposta. Aqui o jogador deve explicar como a penalidade foi executada e submeter. Igual ao caso anterior, os demais jogadores devem aceitar ou rejeitar o tratamento dado à carta.

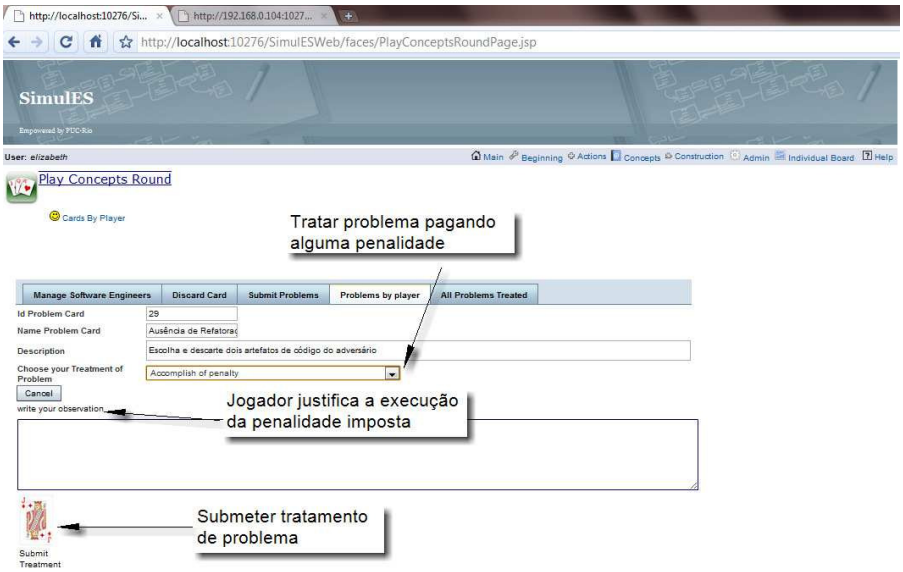


Figura 92 – Tela da *SDsituation* Tratamento de Problema, tratamento com penalidade.

Na Figura 93 ilustramos os problemas tratados e que aguardam aprovação dos jogadores. Se o tratamento é aceito o jogador fica liberado da carta, caso contrario, ela retorna para que o jogador de o tratamento adequado.

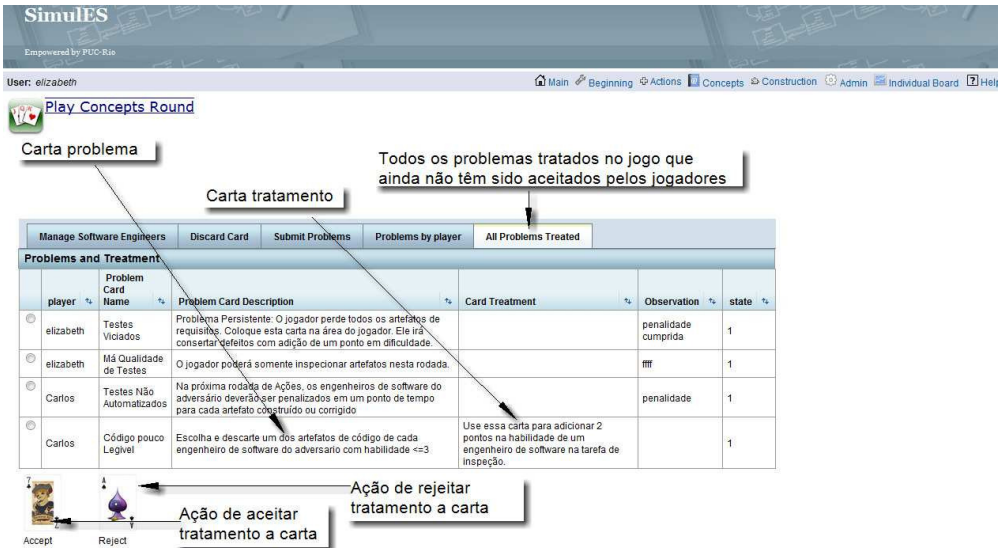


Figura 93 – Tela da *SDsituation* Tratamento de Problema, aceitação de tratamento problema pelos jogadores.

SDsituation integração de artefatos no módulo e submissão de produto

Quando o jogador chega nesta parte do jogo ele tem duas opções ilustradas na Figura 94, submeter o produto sem ter sido inspecionado (o SimulES realiza a inspeção aleatoriamente ou os jogadores inspecionaram o produto do jogador também aleatoriamente) ou submeter quando todos os artefatos tenham sido inspecionados nas diferentes rodadas do jogo.

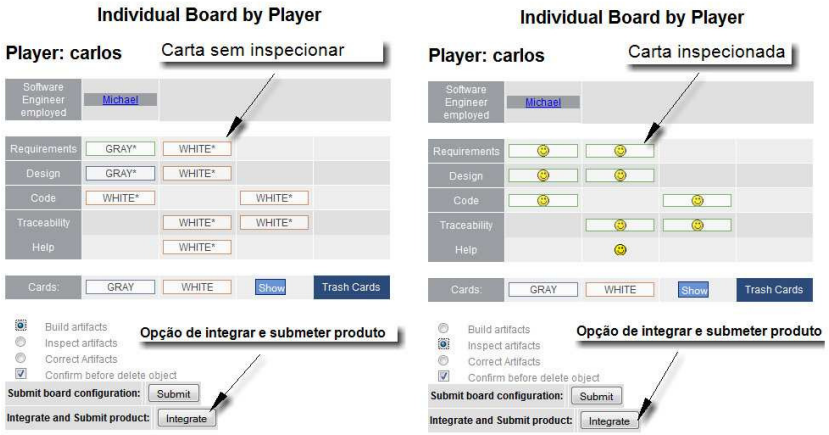


Figura 94 – Tela das *SDsituation*s Integração de Artefatos no Módulo e Submissão de Produto.

SDsituation gestão de material de apoio

As cartas conceito e cartas problemas apresentadas no jogo pertencem a um grupo ou material de apoio. Antes do jogo começar, um jogador administrador escolhe o material de apoio e o sistema disponibiliza as cartas que tenham sido configuradas no sistema para o material de apoio escolhido. Na Figura 95 vemos as operações que são possíveis realizar na tela para este fim.

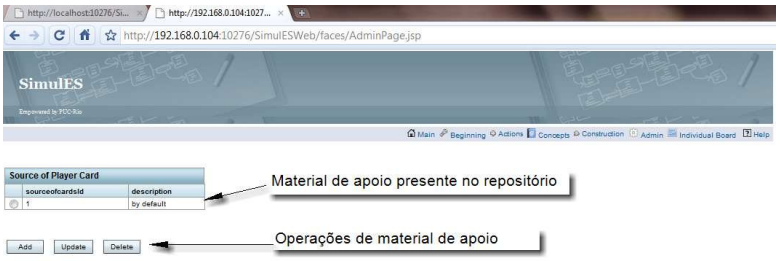


Figura 95 – Tela da *SDsituation* Gestão de Material de Apoio, criar material de apoio.

Cartas problema e cartas conceito que são apresentadas no jogo são geridas nesta tela (Figura 96), esta ilustra as diferentes operações sobre as cartas além de configurar a referência ou origem da carta. A Figura 96 nos apresenta as cartas configuradas para o material de apoio padrão.

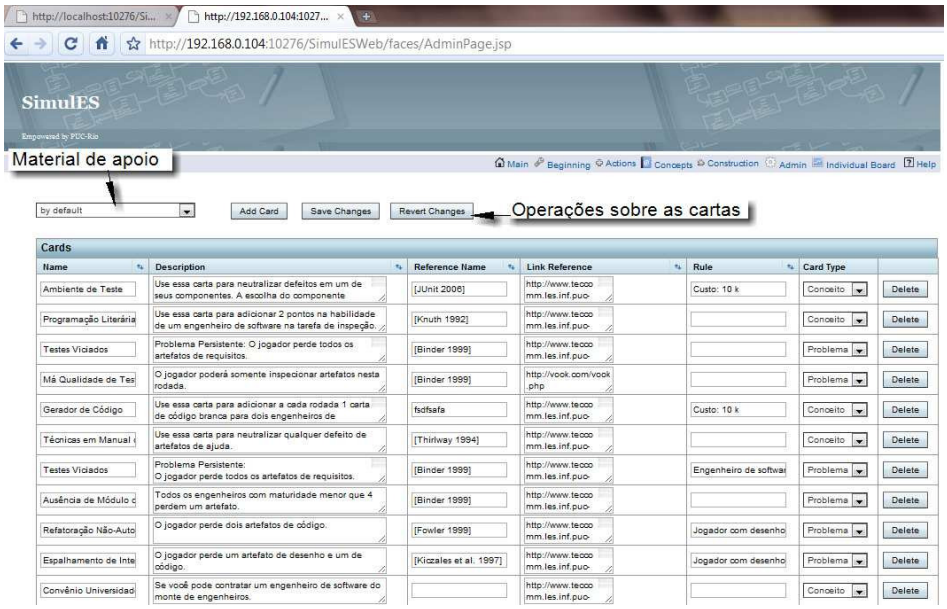


Figura 96 – Tela da *SDsituation* Gestão de Material de Apoio, operações sobre Cartas Conceito e Cartas Problema.

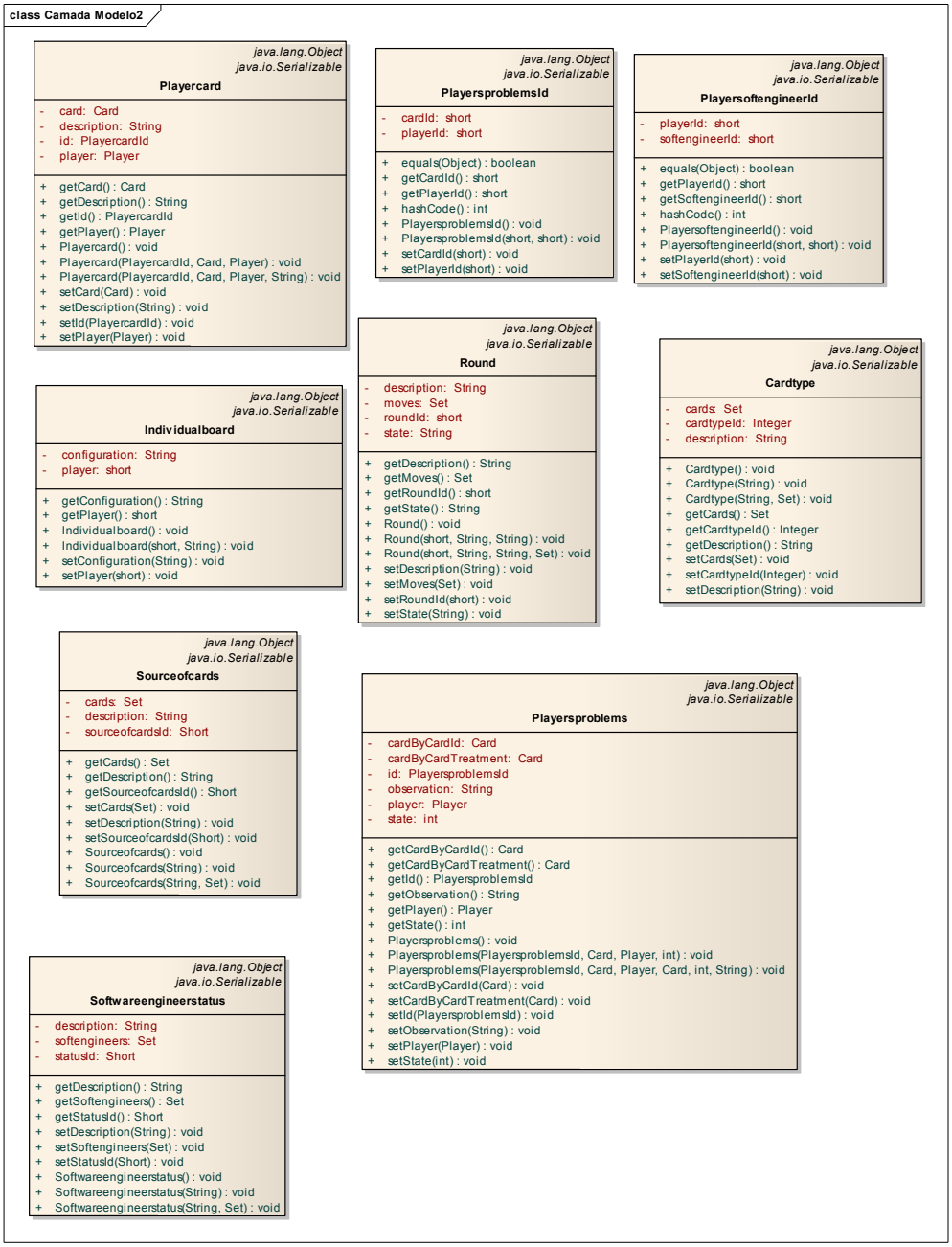
SDsituation gestão do jogo

Finalmente, a Figura 97 ilustra todas as atividades a fazer antes, durante e no final do jogo. Estas atividades devem ser realizadas pelo jogador administrador e garante o correto funcionamento dos recursos. Através das solicitações feitas pelo jogador administrador o SimulES-W deve fornecer os recursos do jogo.

Figura 97 – Tela da *SDsituation* Gestão do Jogo.

6.6.3.2. Operacionalização dos Cenários da Camada do Modelo

Os elementos identificados como recursos e modelados nos diagramas SR e diagramas SD foram representados na camada do modelo (Figura 98), além disso, foram descritos como cenários, pois na implementação apresentaram comportamentos. A Figura 98 apresenta as propriedades e métodos de cada um deles.



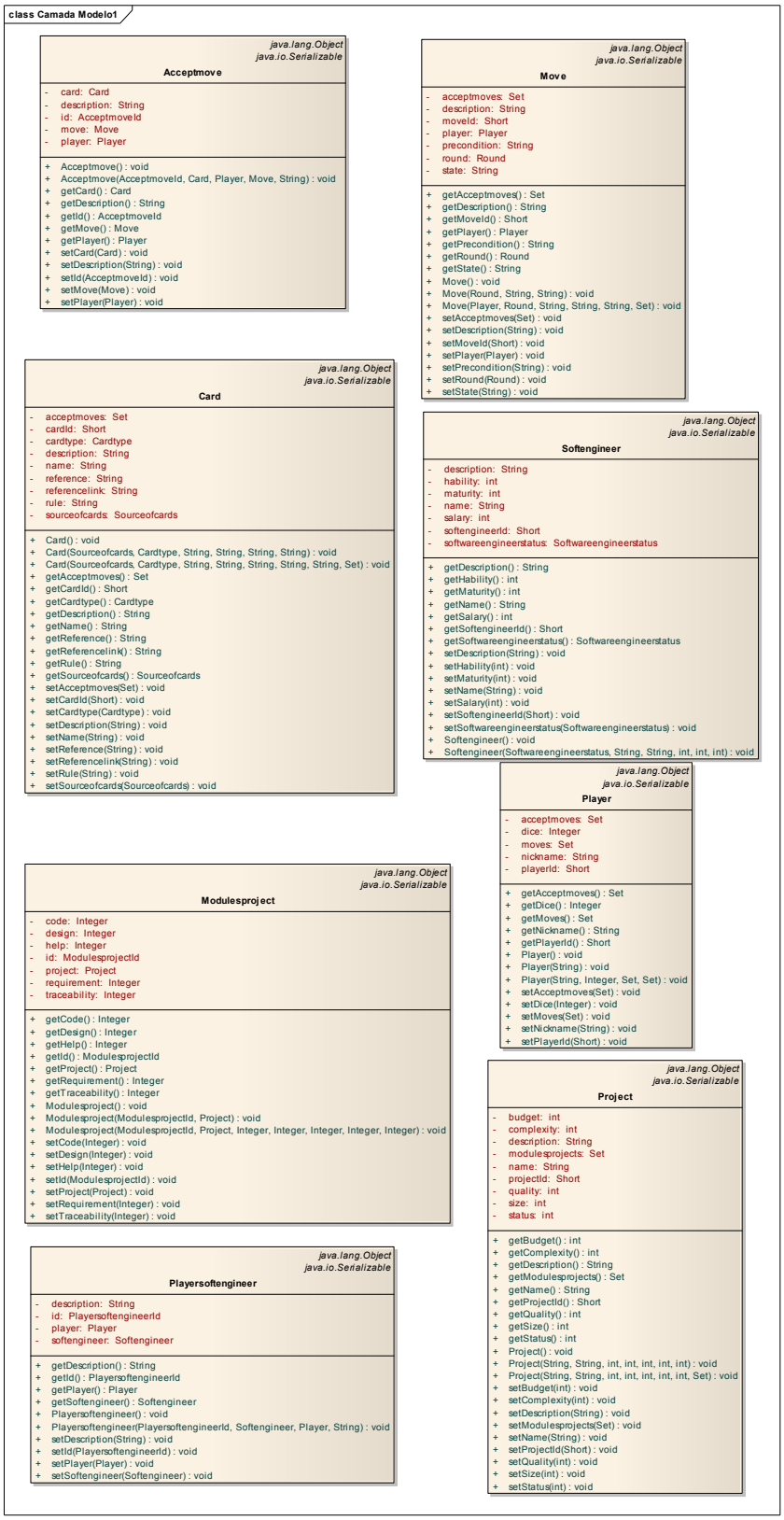


Figura 98 – Operacionalização dos cenários da camada do modelo.

Cada uma das classes na camada de modelo foi descrita usando a técnica de cenários como ilustramos na Figura 99 que representa a classe tabuleiro individual.

```
package SimulES.Model;

/**
Titulo: Tabuleiro Individual
Objetivo: Descrever as operações para obter o tabuleiro individual.
Contexto:
Localização geográfica: Web
Localização temporal: Java Individualboard
Pre-condições: Banco de dados deve estar disponível
Pos-condições: Objeto disponível
Atores: simules

 */

public class Individualboard implements java.io.Serializable {


    //Recursos: Tabuleiro Individual (jogador, configuração)
    private short player;
    private String configuration;

    public Individualboard() {
    }

    //episodio1 receber objeto da camada de controle
    public Individualboard(short player, String configuration) {
        this.player = player;
        this.configuration = configuration;
    }

    //episodio2 enviar objeto à base de dados
    public void setPlayer(short player) {
        this.player = player;
    }
     public void setConfiguration(String configuration) {
        this.configuration = configuration;
    }
     //episodio3 receber objeto da base de dados
    public String getConfiguration() {
        return this.configuration;
    }
    public short getPlayer() {
        return this.player;
    }
}
```

Figura 99 – Parte de código do cenário Tabuleiro Individual na camada modelo.

6.6.3.3.Operacionalização dos Cenários da Camada de Controle

As metas e tarefas que são comportamentos dentro do sistema realizadas por atores para um adequado fluxo do jogo estão na camada de controle.

As classes da camada de controle, Figura 100 se encarregam de receber as requisições da camada de visão, processando-as, depois a camada de visão apresenta o que a camada de controle requisitou.

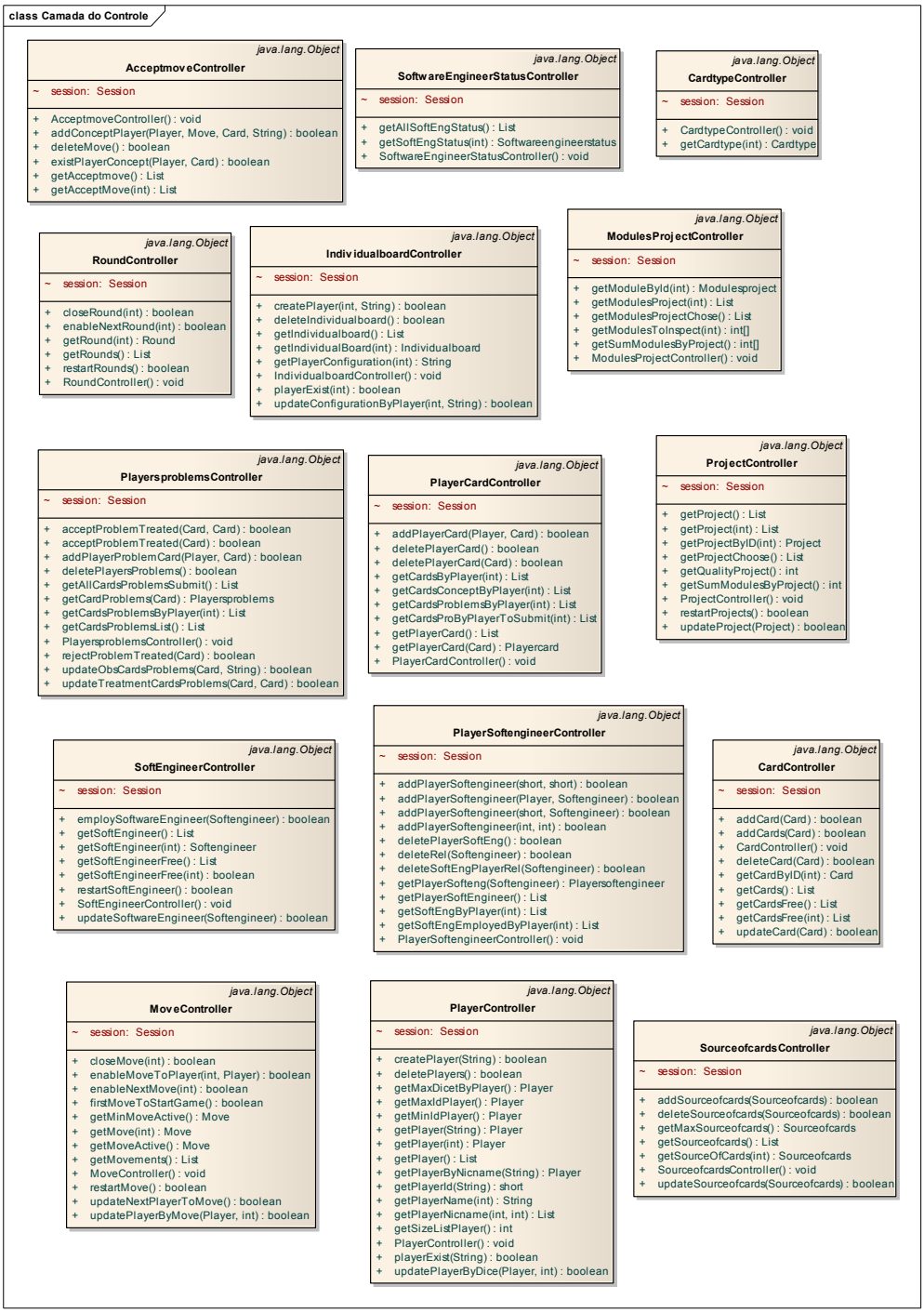


Figura 100 – Operacionalização dos cenários da camada do controle.

Os elementos da camada de controle também foram descritos usando a técnica de cenários como é mostrado na Figura 101.

```
package SimulES.Control;

import SimulES.Model.*;
import java.util.Iterator;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;

/**
Titulo: Controlador de rondas do jogo
Objetivo: Descrever as operações de controle das rondas do jogo.
Contexto:
Localização geográfica: Web
Localização temporal: Java RoundController
Pre-condições: Serviços web disponíveis, fechar a
 * entrada de jogadores já executada
Pos-condições: rondas do jogo controladas pelo SimulES
Atores: jogador, simules
Recursos: informações do projeto,
 * informações dos jogadores, informações dos movimentos e mensageira
Atores: simules, jogador
 */
public class RoundController {

    // session with Hibernate
    Session session = null;

    // constructor
    public RoundController() {
        this.session = HibernateUtil.getSessionFactory().getCurrentSession();
    }

    //@episodio obter todas as rondas do jogo
    public List getRounds() {
        List<Round> roundList = null;
        try {
            if (!this.session.isOpen()) {
                this.session =
                        HibernateUtil.getSessionFactory().getCurrentSession();
            }
            org.hibernate.Transaction tx = session.beginTransaction();
            Query q = session.createQuery("from Round");
            roundList = (List<Round>) q.list();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return roundList;
    }


    //@episodio reiniciar todas as rondas do jogo
    public boolean restartRounds() {
        boolean bTrans = false;
        try {
            if (!this.session.isOpen()) {
                this.session =
                        HibernateUtil.getSessionFactory().getCurrentSession();
            }
            org.hibernate.Transaction tx = session.beginTransaction();
            List<Round> roundList = getRounds();

            Iterator it = roundList.iterator();

            while (it.hasNext()) {
                Round round = (Round) it.next();
                round.setState("I");
            }
            bTrans = true;
            System.out.println("Restart Successfully");
            tx.commit();

        } catch (Exception e) {
            e.printStackTrace();
        }
        return bTrans;
    }

    //@episodio fechar uma ronda especifica
     public boolean closeRound(int roundId) {
        boolean bTrans = false;
        try {
            if (!this.session.isOpen()) {
                this.session =
                        HibernateUtil.getSessionFactory().getCurrentSession();
            }
            org.hibernate.Transaction tx = session.beginTransaction();
            Round round = getRound(roundId);
            session.load(Round.class, round.getRoundId());
            round.setState("E");
            tx.commit();
            System.out.println("Update Successfully");
            bTrans = true;
        //session.close();

        } catch (Exception e) {
```

```
            e.printStackTrace();

        }
        return bTrans;

    }

    //@episodio obter uma ronda especifica
    public Round getRound(int roundId) {
        Round round = null;
        try {
            if (!this.session.isOpen()) {
                this.session =
                        HibernateUtil.getSessionFactory().getCurrentSession();
            }
            org.hibernate.Transaction tx = session.beginTransaction();
            Query q = session.createQuery
                ("from Round as round where round.roundId='" + roundId + "')");
            round = (Round) q.uniqueResult();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return round;
    }


    //@episodio habilitar a seguinte ronda
    public boolean enableNextRound(int roundId) {
        boolean bTrans = false;
        try {
            if (!this.session.isOpen()) {
                this.session =
                        HibernateUtil.getSessionFactory().getCurrentSession();
            }
            org.hibernate.Transaction tx = session.beginTransaction();
            Round round = getRound(roundId);
            session.load(Round.class, round.getRoundId());
            round.setState("A");
            tx.commit();
            System.out.println("Update Successfully");
            bTrans = true;
        //session.close();

        } catch (Exception e) {
            bTrans = false;
            e.printStackTrace();

        }
        return bTrans;
    }

}
```

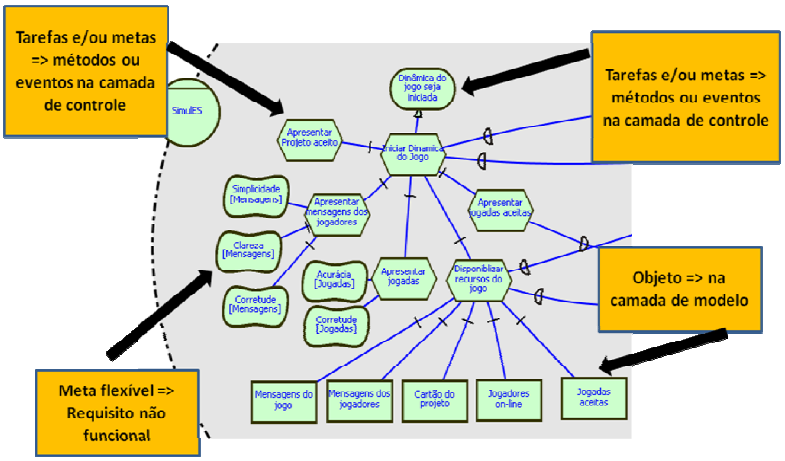
Figura 101 – Parte de código do cenário controlador de rondas da camada controle.

6.7.Rastreabilidade em SimulES-W

Para uma adequada gestão dos requisitos é necessário possuir algum mecanismo de rastreabilidade, que permita identificar a localização e trajetória dos requisitos até o produto software.

Conforme [67] a rastreabilidade de requisitos visa dar qualidade ao processo de desenvolvimento do software tanto no processo mesmo de desenvolvimento quando a evolução de sistema de software. O nosso caso, na Figura 102 apresentamos um fragmento da *SDsituation Apresentar Dinâmica do Jogo*, onde ilustramos a localização dos elementos no modelo e dentro da implementação. Por exemplo, o projeto que é escolhido para ser tratado dentro do jogo é representado como tarefa dentro do diagrama e vemos sua operacionalização na camada de controle através do método *getProjectChoose*, igualmente vemos que um dos recursos a serem fornecidos nesta *SDsituation* são os jogadores on-line e eles são

obtidos através do tipo objeto jogadores, Portanto, metas e tarefas podem ser encontradas na camada de controle, recursos e atores na camada de modelo e na camada de visão as *SDsituations* de SimulES-W.




```
  //@episodio7 retornar o projeto escolhido para ser tratado no jogo
public List getProjectChoose() {

    List<Project> projectList = null;

    try {
         if (!this.session.isOpen()) {
            this.session = HibernateUtil.getSessionFactory().getCurrentSession();
        }
        org.hibernate.Transaction tx = session.beginTransaction();
        Query q = session.createQuery("from Project as project where project.status=1");
        projectList = (List<Project>) q.list();

        int value = projectList.size();

          System.out.println("Size project: " + value);

    } catch (Exception e) {
        e.printStackTrace();
    }

    return projectList;
}
```

```
/**
Titulo: Jogador
Objetivo: Descrever o objeto Jogador.
Contexto:
Localização geográfica: Web
Localização temporal: Java Acceptmove
Pre-condições: Banco de dados deve estar disponível
Pos-condições: Objeto disponível
Atores: simules
Recursos: Player online
Episódios:
1- receber objeto da camada de controle
2- conectar ao banco de dado
3- enviar objeto à base de dados
4- receber objeto da base de dados
 */
public class Player  implements java.io.Serializable {


    private Short playerId;
    private String nickname;
    private Integer dice;
    private Set<Move> moves = new HashSet<Move>(0);
    private Set<Acceptmove> acceptmoves = new HashSet<Acceptmove>(0);

   public Player() {
   }
```

Figura 102 – Segmento da *SDsituation* Dinâmica do Jogo para ilustrar Rastreabilidade.

6.8.Avaliação dos Atributos da Transparência

O SimulES-W, a versão 4.0, surgiu como uma evolução baseada em modelos intencionais. A idéia é que SimulES-W, pela natureza de sua implementação, seja uma aplicação Web centrada nos usuários e principalmente que proporcione uma experiência positiva para os jogadores que buscam em SimulES um meio para aprendizado de engenharia de software. Esta abordagem enfatiza a importância de compreender as necessidades do usuário, as ferramentas e tecnologias utilizadas, e seu contexto social e organizacional.

Acreditamos que um desenvolvimento que esteja centrado no usuário deve seguir diretrizes baseadas na **Transparência de Software,** que permita avaliar a facilidade de uso, rendimento, satisfação e conteúdo, e que por sua vez também são atributos de qualidade do software.

Embora os atributos fossem tidos em conta na modelagem e representados como requisitos não funcionais eles precisam ser avaliados desde a perspectiva do usuário.

Na tabela 11 é apresentado um breve resumo de como os atributos da transparência foram atendidos na modelagem e na implementação do SimulES-W desde a perspectiva do desenvolvedor.

Tabela 11 – SimulES-W e os termos da Transparência.

Características NFR *Framework*	Operacionalização
Acessibilidade	
Portabilidade	SimulES tem a capacidade de rodar em qualquer ambiente por ser uma aplicação Web.
Disponibilidade	SimulES tem a capacidade de estar disponível por ser uma aplicação Web, dependendo do servidor onde este alojado e dos serviços de internet dos usuários.
Publicidade	SimulES é software livre e execução aberta.
Usabilidade	
Simplicidade	Modelagem, código e interface projetados para serem simples, usando léxicos e cenários para explicar os modelos e para explicar o código.
Uniformidade	Modelagem, interfaces e código foram projetadas seguindo padrões. O primeiro modelagem intencional e léxicos e cenários, o segundo o *framework* Visual Web JavaServer Faces e o terceiro o padrão de desenho MVC.
Intuitividade	Foi focada conforme à terminologia usada no jogo manual, telas e controles levam nomes alusivos ao processo, telas são executadas conforme a seqüência do jogo original.
Operabilidade	Implementação baseada nos modelos, com diferentes níveis de aceso, pode ser validados nas tarefas descritas na modelagem.
Adaptabilidade	Focado na evolução com uma arquitetura MVC que permite fazer mudanças de baixo

	impacto.
Desempenho	Testes de desempenho básicas foram realizadas.
Amigabilidade	Modelos descritos em léxicos e cenários o que reduz a complexidade na modelagem. Desenho simples da interface com navegabilidade em todas as telas.
Informativo	
Clareza	Desenho visando o entendimento do usuário. Modelagem e implementação que incluem léxicos e cenários e descrição nas telas conforme terminologia do jogo.
Completeza	Itens descritos na modelagem e representados na implementação.
Atualidade	A implementação do jogo segue a tendência Web.
Corretude	Testes foram realizados, é preciso realizar mais testes.
Comparabilidade	Análise comparativo de modelagens em jogos educacionais, além trabalhos práticos de implementação baseadas na mesma modelagem do SimulES.
Consistência	Testes de primeiro nível foram executadas para avaliar resultados esperados.
Integridade	Gerenciador de erros, definição de precondições e pós-condições
Acurácia	Gerenciador de erros e exceções
Entendimento	
Compositividade	Baseados nas *SDsitiations* é possível identificar e reunir as partes na implementação, camada de visão e camada de controle conforme o descrito nos léxicos e cenários.
Concisão	Gestão de mensagens da aplicação com terminologia usada no jogo manual e de forma concisa.
Dependência	A dependência das partes está representada nas SDsituations e implementada conforme a maquina de estados do jogo.
Divisibilidade	Modelagem por partes mediante SDsituations e implementado na arquitetura MVC onde a camada de visão representa as SDsituations do modelo.
Detalhamento	Detalhamento das SDsituations em modelos intencionais e descrição por cenários, refinamento de cenários conforme a implementação.
Auditabilidade	
Controlabilidade	Execução com acompanhamento da implementação.
Rastreabilidade	Aplicação de técnicas de gestão de configuração na implementação e versionamento da evolução da modelagem.
Validade	Testes executados, com a possibilidade de execução de testes especializados.
Verificabilidade	Testes executados, com a possibilidade de execução de testes especializados.
Explicação	Ajudas incorporadas na implementação e gerenciamento de mensagens de erros

Para complementar este trabalho se fez necessário a análise desde a perspectiva do usuário. Fizemos um teste do SimulES-W no LES (Laboratório de Engenharia de Software) com dois participantes, dessa experiência coletamos algumas observações:

Observações positivas:

- Jogo útil para o ensino dos conceitos da engenharia de software
- Interface limpa que facilita a jogabilidade
- Cartas problema e conceito estimulam a aprender melhor os conceitos da engenharia de software

- Cores e desenhos adequados da interface.
- Jogo em linha para ensino da engenharia de software.

Pontos de melhora:

- Melhorar as ajudas do jogo
- Confuso no uso dos artefatos
- As cartas brancas e cartas cinzas deveriam mostrar os conceitos para construção de artefatos, ou seja, as cartas brancas e cinzas deveriam ter informação e não somente o valor de ter ou não ter erro.

O objetivo da transparência de software é garantir o direito dos usuários a serem informados sobre aquilo que o software faz, porém, é necessário uma análise detalhada de cada um dos atributos de transparência desde a perspectiva do usuário. O trabalho [65] propõe uma método de avaliação de aplicações Web que poderia servir de base para projetar uma avaliação formal do SimulES-W. Trataremos disso no próximo capítulo.