

(REMAF)

Manual de Desenvolvedor

Histórico de Revisões

Comentário	Autor	Data
Versão inicial	Lucas Aciole	18/11/2016
Inclusão do capítulo sobre a customização de jogos	Letícia Domingues	17/02/2017
Refinamento e reestruturação do documento	Delano Beder	29/04/2017

Esse material está licenciado sob a Licença *Creative Commons Atribuição-NãoComercial-CompartilhaIgual 3.0 Brasil*. Para obter uma cópia dessa licença, visite <http://creativecommons.org/licenses/by-nc-sa/3.0/br>.

Abril 2017

Sumário

1	Introdução	1
1.1	Arquitetura da plataforma REMAR	2
2	Jogos customizáveis	3
2.1	Em Busca do Santo Grau	3
2.1.1	Fase de Tecnologia (opcional)	4
2.1.2	Fase Galeria (opcional)	5
2.1.3	Fase Campo Minado	6
2.1.4	Fase Blocos de Gelo	7
2.1.5	Fase TCC	8
2.2	Considerações finais	10
3	Modelos de jogos	11
3.1	Arquitetura Modelo-Visão-Controlador (MVC)	11
3.2	Grails	12
3.2.1	Projeto Grails: hierarquia de diretórios	13
3.2.2	Classes de Domínio	14
3.2.3	Controladores e visões	15
3.2.4	Convenção na nomenclatura de URLs	16
3.3	Integração à plataforma REMAR	17
3.3.1	Ambiente de Desenvolvimento	17
3.3.2	Modelo de jogo SantoGrau	18
3.4	Padronização do layout dos modelos de jogos	22
3.5	Arquivo WAR	22
3.6	Considerações finais	24

1 — Introdução

A plataforma REMAR¹ (Recursos Educacionais Multiplataforma Abertos na Rede) [1, 5] tem como objetivo facilitar e ampliar a construção e o reuso de Recursos Educacionais Abertos (REA). Destacam-se na plataforma REMAR as seguintes funcionalidades:

- Modelos de jogos: publicação de modelos de jogos customizáveis por desenvolvedores;
- Customização: criação de instâncias customizadas de jogos educacionais abertos, a partir de modelos de jogos disponíveis na plataforma;
- Multiplataforma: geração de instâncias dos jogos customizados para diferentes plataformas (web, Android, desktop Windows, Linux e Mac);
- Repositório Digital: publicação de componentes criados para a customização de jogos em Repositório Digital, sob licença Creative Commons, para que possam ser reutilizados na criação de novos jogos;
- Grupos: criação de grupos, compartilhamento de jogos com o grupo e acompanhamento do progresso dos membros do grupo nos jogos compartilhados.

Um diferencial do REMAR é a possibilidade que diferentes modelos de jogos customizáveis, desenvolvidos por diferentes equipes de desenvolvimento, sejam integrados à plataforma de tal forma que usuários (geralmente professores) consigam customizar um jogo e disponibilizar a customização realizada para demais usuários (geralmente seus alunos).

Nesse contexto, este documento tem como objetivo documentar todos os passos necessários para integrar um novo modelo de jogo no REMAR, a partir de diretrizes que favoreçam o reuso bem como a disponibilização desses recursos em diferentes plataformas (web, mobile e desktop). Ao fim desse documento, é esperado que o leitor tenha uma visão geral dos passos necessários para:

- Desenvolver uma aplicação web (modelo de jogo) responsável pelas atividades relacionadas às customizações do jogo;
- Realizar a integração do seu modelo de jogo na plataforma;

¹ <http://remar.dc.ufscar.br>

- Fazer as modificações técnicas necessárias para que, no fim de cada instância de customização, seja gerada uma versão do jogo que reflita a customização realizada.

1.1 Arquitetura da plataforma REMAR

A plataforma REMAR foi desenvolvida utilizando o framework Grails² [7, 8] seguindo uma arquitetura modular. No módulo denominado **Núcleo**, estão presentes as ferramentas que realizam o controle de acesso e permissões dos usuários e a publicação dos jogos. Uma visão geral da arquitetura da plataforma REMAR é apresentada na Figura 1.1.

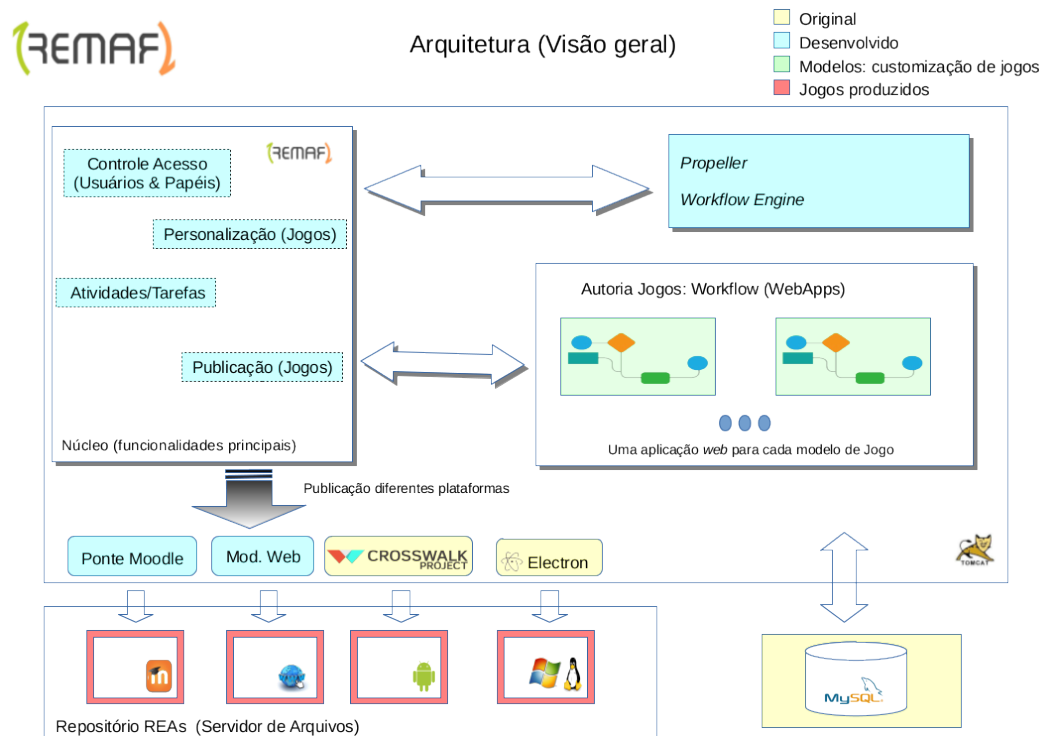


Figura 1.1: Arquitetura da Plataforma REMAR

Cada modelo de jogo representa um módulo presente na arquitetura e consiste de uma aplicação web distinta em que encontram-se o código fonte do jogo e as atividades de customização do jogo. Uma equipe de desenvolvimento que pretende produzir um novo modelo de jogo para a plataforma deverá focar no desenvolvimento, utilizando o framework Grails, da aplicação web responsável pelas atividades de customização do modelo do jogo e na sua integração com o módulo do REMAR. Ou seja, a equipe de desenvolvimento deverá se encarregar da produção de uma aplicação web seguindo as diretrizes que estão descritas neste documento.

Para realizar a integração do módulo **Núcleo** do REMAR e os demais módulos responsáveis pela customização de jogos, é utilizado o **Propeller**, um *Workflow Engine* desenvolvido pela equipe REMAR e disponibilizado para os desenvolvedores. Por fim, a plataforma REMAR utiliza algumas ferramentas (como o Crosswalk³ e o Electron⁴) que disponibilizam e realizam a publicação dos jogos para diversas plataformas (mobile, desktop, etc).

² <https://grails.org/>

³ <https://crosswalk-project.org/>

⁴ <https://electron.atom.io/>

2 — Jogos customizáveis

Atualmente a plataforma REMAR dá suporte a integração de jogos desenvolvidos nas seguintes tecnologias: **HTML5** e **Unity**⁵. Futuramente, espera-se que jogos desenvolvidos em outras tecnologias possam ser incorporados à plataforma REMAR.

No contexto da plataforma REMAR, advoga-se que os jogos (REAs) customizáveis sejam modulares de tal forma que os pontos de customização sejam facilmente delimitados. Nesse sentido, advoga-se que os pontos de customização estejam vinculados a arquivos distintos, presentes no código-fonte do jogo, de tal forma que, durante o processo de customização, esses arquivos possam ser substituídos com o objetivo de se adequarem aos diferentes conteúdos e/ou necessidades.

Entre os possíveis pontos de customização, pode-se citar:

- Arquivos XML ou JSON que, por exemplo, contém as respostas para *quizzes* presentes em um jogo.
- Arquivos de imagens que, por exemplo, podem ser utilizados como o plano de fundo (*background*) em um jogo.

2.1 Em Busca do Santo Grau

Para melhor exemplificar o processo de definição dos pontos de customização, essa seção apresenta uma visão geral das possibilidades de customização do jogo **Em Busca do Santo Grau** [2], um jogo educacional 2D de aventura para PC que permite a customização do conteúdo de quizzes e outros recursos – como inserção de links e figuras integrados em situações-problema. O jogo também possibilita a escolha de fases com os desafios mais adequadas às necessidades do professor. Este jogo oferece dois tipos de fases:

- As **opcionais**, representadas pela **Fase Tecnologia** e **Fase Galeria** (Seções 2.1.1 e 2.1.2), baseiam-se em desafios específicos que podem estar presentes ou não em uma instância customizada do jogo.

⁵<https://unity3d.com/pt>

- As **fixas** são aquelas que obrigatoriamente estarão presentes em uma instância customizada do jogo. Essas fases são baseadas na resolução de desafios associados à questões customizadas (perguntas e respostas).

A seguir serão apresentadas as descrições de todas as fases e os respectivos pontos de customização.

2.1.1 Fase de Tecnologia (opcional)

Nesta fase é possível utilizar um conteúdo multimídia (vídeo do Youtube) para auxiliar o jogador a realizar a tarefa de descobrir 3 palavras-chaves necessárias para destrancar a porta (Figura 2.1).

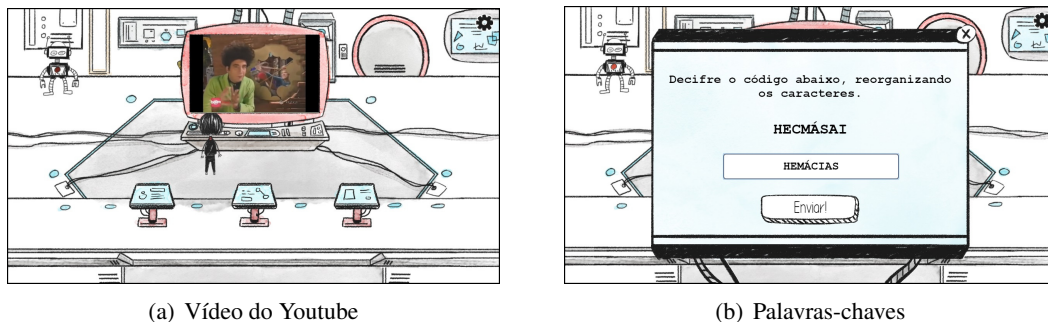


Figura 2.1: Fase de Tecnologia

Nesse contexto, a **Fase de Tecnologia** apresenta dois pontos de customização (Figura 2.2). Esses dois pontos de customização estão vinculados a arquivos de tal forma, que durante o processo de customização, esses arquivos serão gerados com o objetivo de se adequar aos diferentes conteúdos e/ou necessidades.

- O *link* para o vídeo do Youtube encontra-se descrito no arquivo **telao.html** do código-fonte do jogo⁶.
- As 3 palavras-chaves encontram-se descritas no arquivo **computadores.json** do código-fonte do jogo.

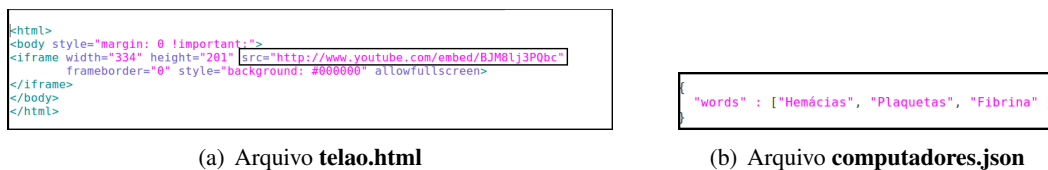


Figura 2.2: Fase de Tecnologia – pontos de customização

Por fim, a interface de customização da **Fase de Tecnologia** (Figura 2.3) consiste em uma página web em que esses dados são informados. É importante mencionar que a interface de customização da **Fase de Tecnologia**, e das demais fases desse jogo, estão sob responsabilidade do modelo de jogo **SantoGrau** (discutido na Seção 3.3.2) – aplicação web, integrada à plataforma REMAR, em que as atividades de customização estão implementadas (ver discussão na Seção 1.1).

⁶ Disponível em <https://github.com/LOA-SEAD/santo-grau>

Figura 2.3: Fase de Tecnologia – interface de customização

2.1.2 Fase Galeria (opcional)

Esta fase, apresentada como uma galeria mal assombrada, exige que o jogador organize conteúdos ilustrados por quadros nas paredes que deverão ser ordenados de acordo com uma dada sequência ou lógica proposta (Figura 2.4).

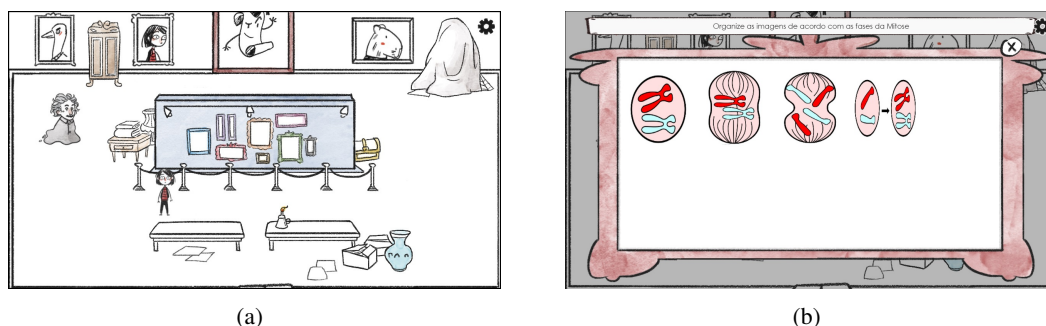


Figura 2.4: Fase Galeria

Nesse contexto, a **Fase Galeria** apresenta como pontos de customização, as imagens (no máximo 10) que serão apresentadas nos quadros presentes na galeria e uma orientação (dica) sobre como o jogador deverá organizar os quadros. Esses pontos de customização encontram-se concretizados no arquivo **quadros.json** do código-fonte do jogo (Figura 2.5) e nos arquivos de imagens (**1.png** a **10.png**). Esses arquivos serão gerados durante o processo de customização com o objetivo de se adequar aos diferentes conteúdos e/ou necessidades.

```
{
  "numero": ["6"],
  "resposta": ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10"],
  "dicaOrdenacao": ["Ordene os quadros na ordem alfabética"]
}
```

Figura 2.5: Fase Galeria – pontos de customização (arquivo **quadros.json**)

Dessa forma, a interface de customização da **Fase Galeria** (Figura 2.6) consiste em páginas web em que as imagens (no máximo 10) a serem utilizadas são informadas e uma orientação (dica) sobre como o jogador deverá organizar as imagens. É importante mencionar que essas páginas web encontram-se sob responsabilidade do modelo de jogo **SantoGrau** (Seção 3.3.2).

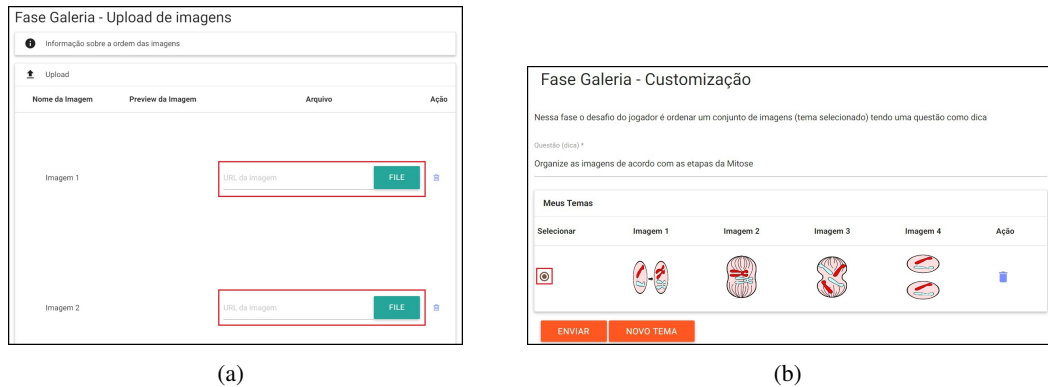


Figura 2.6: Fase Galeria – interface de customização

2.1.3 Fase Campo Minado

Etapa em que o jogador se depara com um campo minado (pontes) dividido em 4 colunas e 5 fileiras com espaços demarcados como A, B, C, D e E. O desafio desta etapa consiste em o jogador pisar no espaço correspondente à alternativa que considera a correta para cada uma das questões que aparecem na tela (Figura 2.7).

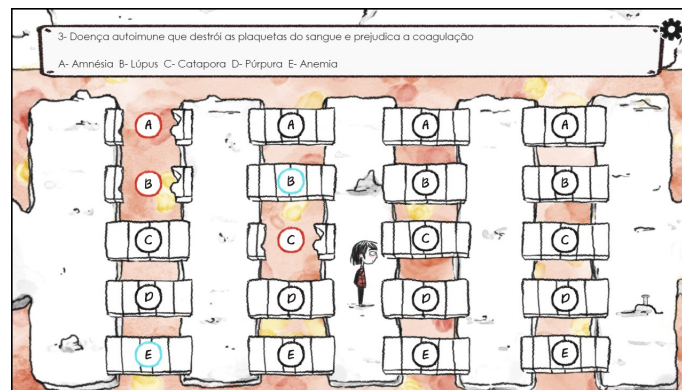


Figura 2.7: Fase Campo Minado

Nesse contexto, a **Fase Campo Minado** apresenta como pontos de customização o banco de questões (de múltipla escolha com 5 alternativas) que encontra-se concretizado no arquivo **questoescm.json** do código-fonte do jogo (Figura 2.8). Esse arquivo será gerado durante o processo de customização com o objetivo de se adequar aos diferentes conteúdos e/ou necessidades.

```

"quantidadeQuestoes": ["5"],
"1": ["Qual é a capital do Brasil?", "Salvador", "Campinas", "Tóquio", "Brasília", "Nova Iorque", "D"],
"2": ["Que feriado é comemorado no dia 21 de Abril?", "Páscoa", "Tiradentes", "Natal", "Carnaval", "Independência", "B"],
"3": ["Quem descobriu o Brasil?", "Cristóvão Colombo", "Pero Vaz de Caminha", "Dom Pedro I", "Pelé", "Pedro Álvares Cabral", "E"],
"4": ["Quanto nove existem entre 0 e 100?", "20", "19", "10", "15", "11", "A"],
"5": ["Qual valor completa a seguinte série: 9, 16, 25, 36, _", "56", "45", "49", "61", "63", "C"]

```

Figura 2.8: Fase Campo Minado – pontos de customização (arquivo **questoescm.json**)

Por fim, a interface de customização da **Fase Campo Minado** (Figura 2.9) consiste em páginas web em que questões podem ser cadastradas e posteriormente selecionadas para ser adicionadas ao banco de questões que será utilizado pelo jogo (arquivo **questoescm.json**). É importante mencionar que essas páginas web encontram-se sob responsabilidade do modelo de jogo **SantoGrau** (Seção 3.3.2).

(a)

(b)

Figura 2.9: Fase Campo Minado – interface de customização

2.1.4 Fase Blocos de Gelo

Nesta fase o jogador precisa interagir com 3 blocos que estão espalhados e deslizando por uma pista de gelo. Cada bloco (1, 2 e 3) representa uma alternativa correspondente à questão customizada e o jogador deverá pegar aquele que considerar a resposta certa. Entretanto, a dificuldade desta sala concentra-se também na captura desses blocos, os quais estarão deslizando sobre o gelo do cenário (Figura 2.10).

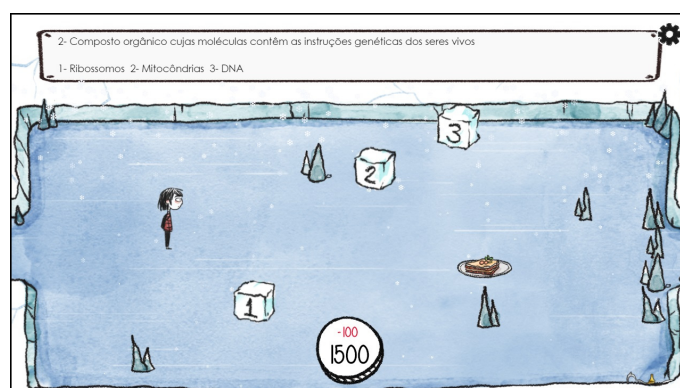


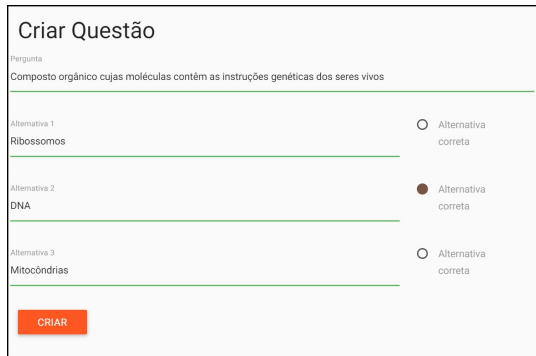
Figura 2.10: Fase Blocos de Gelo

Nesse contexto, a **Fase Blocos de Gelo** apresenta como pontos de customização o banco de questões (de múltipla escolha com 3 alternativas) que encontra-se concretizado no arquivo **questoesbn.json** do código-fonte do jogo (Figura 2.11).

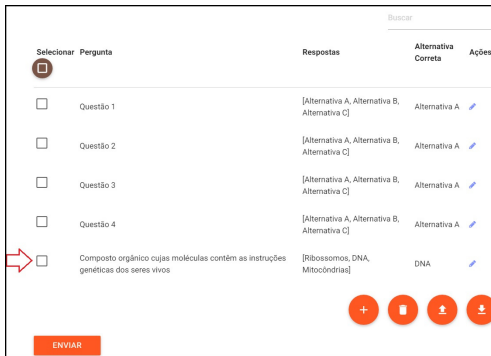
```
{
  "quantidadeQuestoes": ["4"],
  "1": ["Quem escreveu Hamlet?", "Shakespeare", "Orwell", "Dostoiévski", "A"],
  "2": ["Quais destas cores não está na bandeira brasileira?", "Azul", "Preto", "Branco", "B"],
  "3": ["Quantos meses do ano possuem apenas 30 dias?", "4", "11", "5", "A"],
  "4": ["China é um país localizado em qual continente?", "Europa", "América", "Ásia", "C"]
}
```

Figura 2.11: Fase Bloco de Gelo – pontos de customização (arquivo **questoesbn.json**)

Por fim, a interface de customização da **Fase Blocos de Gelo** (Figura 2.12) consiste em páginas web em que questões podem ser cadastradas e posteriormente selecionadas para ser adicionadas ao banco de questões que será utilizado pelo jogo (arquivo **questoesbn.json**). É importante mencionar que essas páginas web encontram-se sob responsabilidade do modelo de jogo **SantoGrau** (Seção 3.3.2).



(a)



(b)

Figura 2.12: Fase Bloco de Gelo – interface de customização

2.1.5 Fase TCC

A mecânica desta fase ocorre com o monstro de papel (o temido TCC) atirando projéteis no jogador, o qual precisa se esquivar ao mesmo tempo em que responde a 5 questões com suas alternativas demarcadas pelo cenário (A, B, C, D e E). A cada questão certa, o jogador ganha a oportunidade de atacar o monstro a partir de um pedaço de madeira que precisa ser aceso nas tochas para lançar bolas de fogo no monstro (Figura 2.13).

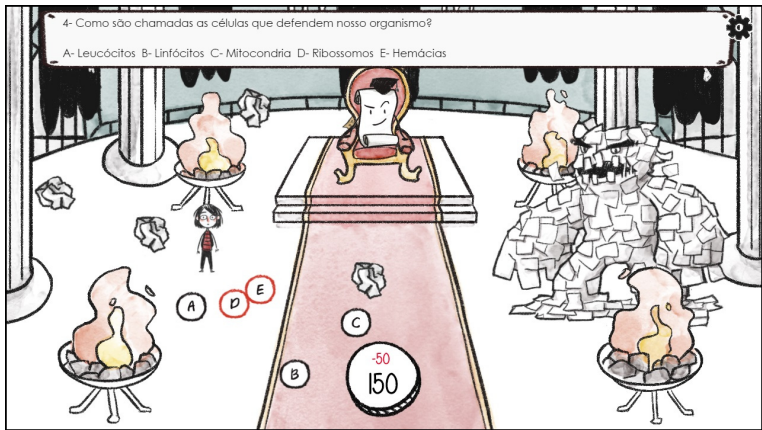


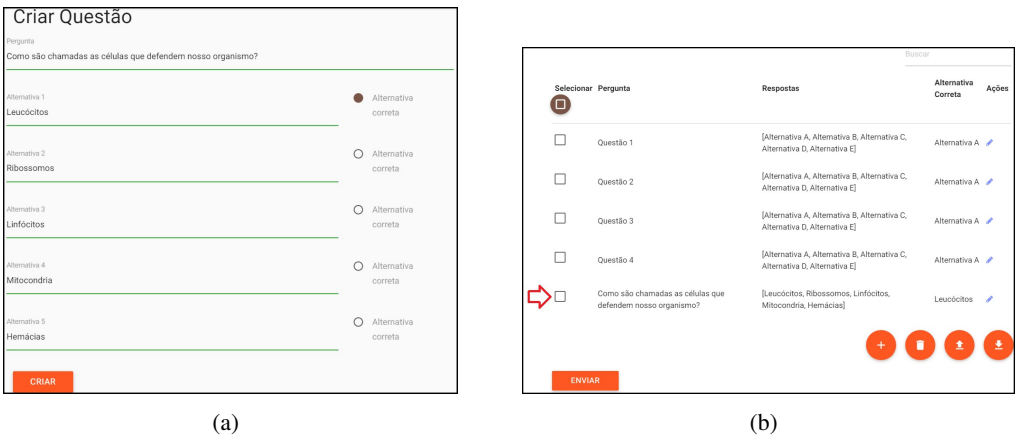
Figura 2.13: Fase TCC

Nesse contexto, a **Fase TCC** apresenta como pontos de customização o banco de questões (de múltipla escolha com 5 alternativas) que encontra-se concretizado no arquivo **questoestcc.json** do código-fonte do jogo (Figura 2.14).

```
{
  "quantidadeQuestoes": [ "6" ],
  "1": [ "Qual o nome da glândula responsável por manter a temperatura do corpo?", "Pineal", "Acinotarsal", "Sebácea", "Sudoripara", "Salivar", "D" ],
  "2": [ "Ache um número que tenha sua raiz quadrada maior do que ele mesmo.", "1/2", "1", "5", "2", "0", "A" ],
  "3": [ "Qual é o elemento que não se enquadra perfeitamente em nenhum grupo/família da tabela periódica devido a uma característica única?", "Urânio", "Hidrogênio", "Mercúrio", "Oxigênio", "Carbono", "B" ],
  "4": [ "Gregório de Matos, Padre Antônio Vieira e Bento Teixeira são escritores de qual período literário?", "Romantismo", "Modernismo", "Barroco", "Realismo", "Simbolismo", "C" ],
  "5": [ "Angola é um país localizado em qual continente?", "África", "América", "Ásia", "Europa", "Oceania", "A" ],
  "6": [ "Qual país ganhou mais medalhas de ouro nas olimpíadas do Rio de Janeiro em 2016?", "Alemanha", "Rússia", "China", "Reino Unido", "Estados Unidos", "E" ]
}
```

Figura 2.14: Fase TCC – pontos de customização (arquivo **questoestcc.json**)

Por fim, a interface de customização da **Fase TCC** (Figura 2.15) consiste em páginas web em que questões podem ser cadastradas e posteriormente selecionadas para ser adicionadas ao banco de questões que será utilizado pelo jogo (arquivo **questoestcc.json**). É importante mencionar que essas páginas web encontram-se sob responsabilidade do modelo de jogo **SantoGrau** (Seção 3.3.2).



(a)

(b)

Figura 2.15: Fase TCC – interface de customização

2.2 Considerações finais

Esse capítulo apresentou uma visão geral das possibilidades de customização do jogo **Em Busca do Santo Grau** [2]. Conforme discutido, no contexto da plataforma REMAR, advoga-se que os pontos de customização estejam vinculados a arquivos, presentes no código-fonte do jogo, de tal forma que, durante o processo de customização, esses arquivos possam ser substituídos com o objetivo de se adequarem aos diferentes conteúdos e/ou necessidades.

Nesse contexto, Tabela 2.1 apresenta os pontos de customização (e seus respectivos arquivos) definidos para as diferentes fases do jogo **Em Busca do Santo Grau**.

Fase	Arquivo	Descrição
Tecnologia	computadores.json	3 palavras-chaves necessárias para destrancar a porta presente na fase
Tecnologia	telao.html	Link para um vídeo do Youtube para auxiliar a tarefa de descobrir as 3 palavras-chave
Galeria	1.png a 10.png	Imagens (no máximo 10) que serão apresentadas nos quadros presentes na galeria
Galeria	quadros.json	Orientação sobre com o jogador deverá organizar os quadros
Campo Minado	questoescm.json	Banco de questões (de múltipla escolha com 5 alternativas)
Blocos de Gelo	questoesbn.json	Banco de questões (de múltipla escolha com 3 alternativas)
TCC	questoestcc.json	Banco de questões (de múltipla escolha com 5 alternativas)

Tabela 2.1: **Em Busca do Santo Grau**: visão geral dos pontos de customização

3 — Modelos de jogos

Este capítulo tem como objetivo documentar todos os aspectos técnicos necessários para integrar um novo modelo de jogo à plataforma REMAR.

3.1 Arquitetura Modelo-Visão-Controlador (MVC)

Conforme mencionado, a plataforma REMAR foi desenvolvida utilizando o framework de desenvolvimento Grails [7, 8]. Dessa forma, advoga-se a utilização desse framework no desenvolvimento de novos modelos de jogos. Visto que o padrão arquitetural MVC é o alicerce do framework Grails, torna-se de fundamental importância apresentar uma visão geral dos principais conceitos relacionados a esse padrão arquitetural. O modelo-visão-controlador (MVC) [4, 6] desacopla a interface com o usuário da funcionalidade e conteúdo da aplicação web. Uma representação esquemática da arquitetura MVC aparece na Figura 3.1.

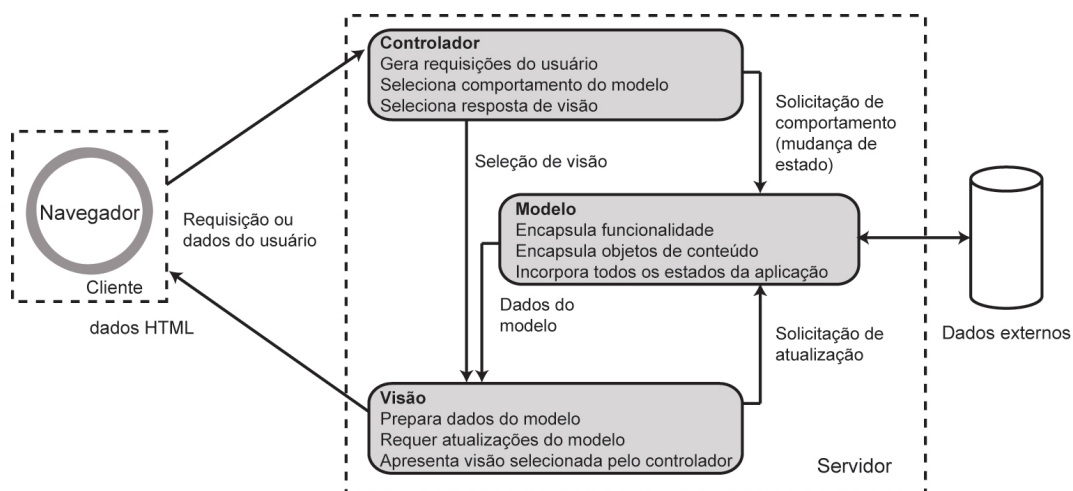


Figura 3.1: Padrão arquitetural MVC

Esse padrão arquitetural define três componentes (Modelo, Visão e Controlador) com características bem definidas:

- O **Modelo** contém todo o conteúdo específico da aplicação e lógica de processamento, incluindo todos os objetos de conteúdo, acesso a dados externos e fontes de informação, e todas as funcionalidades de processamento que são específicas da aplicação. No caso de sistemas que utilizam bases de dados, o modelo mantém o estado persistente do negócio e somente ele pode acessar as bases de dados;
- A **Visão** contém todas as funcionalidades específicas da interface que habilita a apresentação de conteúdo e lógica de processamento; e
- O **Controlador** gerencia o acesso e a manipulação do modelo e da visão, bem como coordena o fluxo de dados entre eles. Em uma aplicação web, o controlador monitora a interação com o usuário e, baseando-se nisso, recupera os dados do modelo e utiliza-os para atualizar ou construir a visão.

Conforme mostra a Figura 3.1, as solicitações ou dados do usuário são tratados pelo controlador que seleciona a visão apropriada com base na solicitação do usuário. Quando o tipo de solicitação é determinado, uma solicitação de comportamento é transmitida ao modelo, que implementa a funcionalidade ou recupera o conteúdo exigido para satisfazer a solicitação. O modelo pode acessar dados armazenados em um banco de dados corporativo, como parte de um armazenamento de dados local ou como uma coleção de arquivos independentes. Os dados devolvidos pelo modelo devem ser formatados e organizados pela visão apropriada e depois transmitidos do servidor da aplicação de volta para exibição no navegador presente na máquina do cliente [6].

3.2 Grails

Grails [7, 8] é um framework web baseado no padrão arquitetural MVC [4] que utiliza a linguagem Groovy [3], executa sobre a máquina Virtual Java (JVM) e objetiva a alta produtividade no desenvolvimento de aplicações web. Ele combina os principais frameworks (Hibernate⁷, Spring⁸, etc.) utilizados na plataforma Java e respeita o paradigma *Convention-over-configuration* (Convenção ao invés de Configuração).

Groovy. Groovy [3] é uma linguagem dinâmica, ágil para a plataforma Java inspirada em Python e Ruby que possui sua sintaxe semelhante à de aplicações desenvolvidas em Java. Apesar de poder ser usada como uma linguagem de *script*, ou seja, não gerar arquivos executáveis e não precisar ser compilada, Groovy não se limita a isso. Aplicações feitas nesta linguagem podem ser compiladas utilizando-se um compilador Java, gerando *bytecodes* Java (mesmo formato da compilação de uma aplicação escrita em Java), além disso, podem ser utilizadas em aplicações escritas puramente em Java.

A linguagem foi desenvolvida em 2004 por James Strachan. A sua sintaxe é extremamente parecida com a do Java, além disso, é possível “integrar” aplicações Java e Groovy de forma transparente. O Groovy, inclusive, simplifica a implementação por “adicionar” dinamicamente às suas classes os métodos de acesso (**get** e **set**), economizando tempo e esforço. O objetivo de Groovy é simplificar a sintaxe de Java para representar comportamentos dinâmicos como

⁷ <http://www.hibernate.org/>

⁸ <http://springsource.org/>

consultas a banco de dados, escritas e leituras de arquivos e geração de objetos em tempo de execução ao invés de compilação [3].

Convention Over Configuration (CoC). O CoC é um paradigma que visa a diminuir a quantidade de decisões que o desenvolvedor precisa tomar, tomando como “padrão” algo que é comumente usado (uma convenção). Se o padrão escolhido pelo framework for a que o desenvolvedor precisa, este não gasta tempo tendo que alterá-la. Entretanto, se ele necessita de algo diferente, fica livre para configurar da forma que desejar. No caso do Grails, ele assume diversas configurações, tais como as de banco de dados, as de localização do código-fonte, entre outras.

3.2.1 Projeto Grails: hierarquia de diretórios

Um projeto Grails consiste de uma hierarquia de diretórios que segue o paradigma *Convention Over Configuration*. Ou seja, os desenvolvedores seguem as convenções e já sabem *a priori* onde se encontram todos os elementos que compõem a aplicação em desenvolvimento. Uma visão geral do conteúdo desses diretórios é apresentada na Tabela 3.1.

Diretório	Descrição
grails-app/domain	Diretório onde se encontra o M do MVC. Ou seja, onde se encontram as classes de Domínio, ou modelos.
grails-app/controllers	Diretório onde se encontra o C do MVC. Ou seja, onde se encontram os controladores.
grails-app/views	Diretório onde se encontra o V do MVC. Ou seja, onde se encontram as visões (arquivos.gsp – Groovy Server Pages).
grails-app/taglib	Diretório onde se encontram as bibliotecas de marcas (<i>taglibs</i>) criadas pelo usuário.
grails-app/services	Diretório onde se encontram as classes utilizadas na camada de serviços (serviços web).
grails-app/i18n	Diretório onde se encontram os arquivos relacionados à internacionalização.
grails-app/conf	Diretório onde se encontram as configurações da aplicação, tais como a configuração do banco (application.yml), entre outros.
grails-app/init	Diretório onde se encontra a classe BootStrap.groovy utilizada na inicialização de dados da aplicação, entre outros.
grails-app/assets	Esse diretório possui três diretórios (<i>images</i> , <i>javascript</i> e <i>stylesheets</i>) onde se encontram os <i>assets</i> utilizados na aplicação.
src/main/groovy	Diretório onde se encontram outros códigos-fonte Java ou Groovy que não são modelos, controladores, visões ou serviços.
src/test/groovy	Diretório onde se encontram os testes unitários da aplicação.
src/integration-test/groovy	Diretório onde se encontram os testes de integração da aplicação.

Tabela 3.1: Projeto Grails: visão geral da hierarquia de diretórios

3.2.2 Classes de Domínio

Levando em consideração o padrão arquitetural MVC [4], as classes de domínio fazem parte do modelo (o M do MVC) de uma aplicação Grails. As classes de domínio são responsáveis pela representação dos dados gerenciados pela aplicação. Geralmente, cada classe de domínio está relacionada a uma tabela do banco de dados, onde seus atributos são concretizados em colunas da tabela.

A definição das classes de domínio é o primeiro passo do processo de integração de um modelo de jogo na plataforma REMAR. Ela é feita a partir da reflexão de quais dados precisam ser persistidos para fazer a customização de um jogo. Figura 3.2 apresenta a lista de classes de domínio do modelo de jogo **SantoGrau**⁹ (Seção 3.3.2).



Figura 3.2: Aplicação **SantoGrau** – classes de domínio

Código 3.1 apresenta a classe de domínio **QuestionFaseBlocoGelo** que representa as questões que são utilizadas na customização da **Fase Bloco Gelo** (Seção 2.1.4).

```
package br.ufscar.sead.loa.santograu.remar

class QuestionFaseBlocoGelo {
    String title
    String[] answers = new String[3]
    int correctAnswer
    long ownerId

    static constraints = {
    }
}
```

Código 3.1: Classe de domínio **QuestionFaseBlocoGelo**

Conforme pode ser observado, essa classe de domínio possui os seguintes atributos:

- **title** que armazena o enunciado da questão (de múltipla escolha com 3 alternativas);
- **answers** que armazena as três possíveis respostas da questão de múltipla escolha;
- **correctAnswer** que armazena a alternativa correta da questão;
- **ownerId** que armazena o *id* do usuário (autor da questão) da plataforma REMAR.

⁹ Aplicação web, integrada à plataforma REMAR, em que as atividades de customização do jogo **Em Busca do Santo Grau** estão implementadas.

Uma importante característica do Grails é que programadores não precisam se preocupar em criar os métodos *getters* e *setters*. Eles são gerados pelo Groovy. Grails, por meio do mecanismo GORM (*Grails Object-Relational Mapping*), realiza um mapeamento automático entre modelos (classes de domínio) e tabelas em um SGBD.

Dessa forma, supondo que exista uma classe de domínio **Questao**, o Groovy gera outros métodos estáticos responsáveis pelas operações *CRUD* (*Create, Read, Update e Delete*):

- **Questao.save()** armazena os dados na tabela Questao (do SGBD).
- **Questao.delete()** apaga os dados da tabela Questao.
- **Questao.list()** retorna uma lista de questões (armazenadas na tabela).
- **Questao.get(id)** retorna uma única instância da classe **Questao**.

Todos esses e outros métodos estão disponíveis para os desenvolvedores. Note que **Questao** não estende nenhuma classe pai nem implementa nenhuma interface. Graças aos recursos de metaprogramação Groovy, esses métodos simplesmente aparecem quando necessários. Apenas as classes de domínio possuem esses métodos relacionados à persistência de dados: **save()**, **delete()**, **list()** e **get()**.

3.2.3 Controladores e visões

Uma vez que as classes de domínio estão criadas, é preciso criar os controladores e as visões relacionados a essas classes de domínio. Um controlador gerencia o acesso e a manipulação do modelo (classe de domínio) e da visão, bem como coordena o fluxo de dados entre eles. Em uma aplicação Grails, o controlador monitora a interação com o usuário e, baseando-se nisso, recupera os dados do modelo (classe de domínio) e utiliza-os para atualizar ou construir a visão. No contexto dos modelos de jogos, integrados à plataforma REMAR, os controladores estão relacionados às atividades de customização previstas. Figura 3.3 apresenta a lista de controladores do modelo de jogo **SantoGrau** (Seção 3.3.2).

É importante mencionar que, no contexto do modelo de jogo **SantoGrau**, cada controlador é responsável pela implementação das atividades de customização previstas em uma das fases do jogo.

- **FaseGaleriaController**: Responsável pelas atividades de customização da **Fase Galeria**
- **FaseTCCController**: Responsável pelas atividades de customização da **Fase TCC**
- etc.

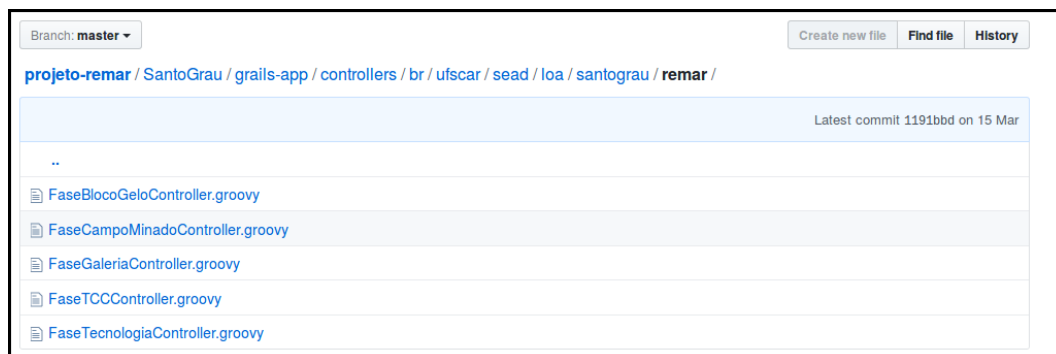


Figura 3.3: Aplicação **SantoGrau** – controladores

Em Grails, as visões são *Groovy Server Pages (GSPs)* que podem ser definidas como um arquivo HTML básico que contém algumas *tags* Grails (elementos `<g: .. />`).

Novamente os benefícios do paradigma *Convention Over Configuration* faz-se presente.

- As visões encontram-se dentro de um diretório com o nome do controlador associado (sem o sufixo **Controller**). Figura 3.4 apresenta a lista de visões associadas ao controlador **FaseBlocoGelo**.
- Toda ação (método) em um controlador é associada a uma visão baseada em seu nome (**index** → **index.gsp**). Elas geralmente são: **_form.gsp**, **create.gsp**, **edit.gsp**, **index.gsp** e **show.gsp**. Assim como seus nomes sugerem, elas oferecem as operações básicas de inserir, editar, excluir e mostrar os dados no banco de dados.



Figura 3.4: Aplicação **Santo Grau** – visões do controlador **FaseBlocoGeloController**

3.2.4 Convenção na nomenclatura de URLs

Grails usa uma convenção (Figura 3.5) para automaticamente configurar o caminho para uma ação em particular. A URL a seguir pode ser entendida da seguinte forma:

- “Execute a ação **show()** do controlador **faseTCC** (classe **FaseTCCController**) – um dos controladores da aplicação hospedada na porta 8080 do servidor **localhost**”.

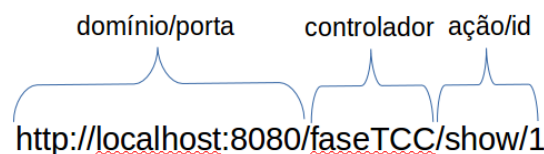


Figura 3.5: Convenção na nomenclatura de URLs.

A regra principal de roteamento define URLs que segue o padrão **/controlador/ação/id**, onde **controlador** é o controlador (da arquitetura MVC) responsável por atender a requisição especificada por aquela *URL*, **ação** é um método dentro do controlador e **id** é um parâmetro opcional passado para identificar um objeto qualquer sobre o qual a ação será efetuada. A visão associada a essa ação geralmente é invocada (o controlador pode redirecionar para outra visão).

Por exemplo, **/faseTCC/show/1** invoca a visão **faseTCC/show.gsp**

3.3 Integração à plataforma REMAR

Conforme discutido, cada modelo de jogo consiste de uma aplicação web distinta em que encontram-se o código fonte do jogo e as atividades de customização do jogo. Uma equipe de desenvolvimento que pretende produzir um novo modelo de jogo para a plataforma deverá focar no desenvolvimento, utilizando o framework Grails, da aplicação web responsável pelas atividades de customização do modelo do jogo e na sua integração com o módulo do REMAR.

Todos os modelos de jogos integrados à plataforma REMAR são representados por um processo de customização (representado pelo arquivo **process.json**¹⁰). Um processo é composto por diversas tarefas (**tasks**), que por sua vez possuem uma ou mais saídas (**outputs**). As saídas (**outputs**) representam arquivos, presentes no código-fonte do jogo, que durante o processo de customização, são substituídos com o objetivo de se adequarem aos diferentes conteúdos e/ou necessidades.

As tarefas são implementados como controladores Grails (Seção 3.2.3) e possuem a responsabilidade pela implementação das atividades de customização previstas em um jogo. Ou seja, é responsabilidade das tarefas:

- disponibilizar uma **interface** web (visões associadas ao controlador) para que as informações associadas a cada customização prevista sejam informadas;
- criar os **outputs** customizados; e
- enviar os **outputs** à plataforma REMAR para que este produza e publique uma nova instância customizada do jogo.

3.3.1 Ambiente de Desenvolvimento

Para o ambiente de desenvolvimento, é necessária a instalação das seguintes ferramentas:

Linguagens de Programação/Framework de Desenvolvimento

- Grails 2.4.5
- Java 8 - Oracle SDK

Banco de Dados

- MongoDB 3.2
- MySQL 5.5

Servidor web

- Tomcat 7.0.77

Também é indicado a instalação da IDE IntelliJ IDEA (Ultimate 2016.2), que disponibiliza um ambiente facilitando o desenvolvimento de aplicações utilizando o framework Grails. Para a utilização desse IDE é necessária uma licença, que poderá ser obtida gratuitamente caso o desenvolvedor for aluno/professor de alguma instituição de ensino superior, utilizando o e-mail da universidade para o cadastro¹¹.

¹⁰ Um exemplo desse arquivo é apresentado no Código 3.2.

¹¹ <https://www.jetbrains.com/student/>

3.3.2 Modelo de jogo SantoGrau

Essa seção apresenta uma visão geral do modelo de jogo **SantoGrau** — aplicação web, integrada à plataforma REMAR, em que as atividades de customização do jogo **Em Busca do Santo Grau** (Seção 2.1) estão implementadas.

Conforme pode-se observar, o processo de customização, representado pelo arquivo **process.json** (Código 3.2), do modelo de jogo **SantoGrau** é composto por 5 tarefas (sendo 2 opcionais). Cada tarefa (controlador listado na Figura 3.3) é responsável pela implementação das atividades de customização previstas em cada uma das fases do jogo (Seção 2.1). Por questões de brevidade, as saídas (**outputs**) de cada tarefa são omitidas no Código 3.2.

```
{
  "name": "Em Busca do Santo Grau",
  "uri": "santograu",
  "type": "html",
  "version": 1,
  "tasks": [
    {
      "name": "Fase Tecnologia",
      "uri": "faseTecnologia",
      "optional": true,
      "description": "Configurar um conteúdo multimídia com 3 palavras-chaves relacionadas ao conteúdo, que serão embaralhadas em anagramas.",
      "type": "media_list",
      "dependencies": null,
      "outputs": [
        // outputs da Fase Tecnologia
      ]
    },
    {
      "name": "Fase Galeria",
      "uri": "faseGaleria",
      "optional": true,
      "description": "Criar/editar/selecionar um tema de imagens para o nível Galeria.",
      "type": "image_collection",
      "dependencies": null,
      "outputs": [
        // outputs da Fase Galeria
      ]
    },
    {
      "name": "Fase Campo Minado",
      "uri": "faseCampoMinado",
      "description": "Criar/editar/selecionar um banco de questões para o nível Campo Minado.",
      "type": "text_database",
      "dependencies": null,
      "outputs": [
        // outputs da Fase Campo Minado
      ]
    },
    {
      "name": "Fase Bloco de Gelo",
      "uri": "faseBlocoGelo",
      "description": "Criar/editar/selecionar um banco de questões para o nível Bloco de Gelo.",
      "type": "text_database",
      "dependencies": null,
      "outputs": [
        // outputs da Fase Bloco de Gelo
      ]
    },
    {
      "name": "Fase TCC",
      "uri": "faseTCC",
      "description": "Criar/editar/selecionar um banco de questões para o nível final TCC.",
      "type": "text_database",
      "dependencies": null,
      "outputs": [
        // outputs da Fase TCC
      ]
    }
  ],
  "outputs": [
    "web"
  ],
  "vars": {
    "width": 1280,
    "height": 720
  }
}
```

Código 3.2: Arquivo **process.json** do modelo de jogo **SantoGrau**

Figura 3.6 lista as tarefas do modelo de jogo **SantoGrau**¹², conforme apresentadas na plataforma REMAR.

Tarefas	
Atenção: Tarefas marcadas com * são obrigatórias	
Nome	Status
Fase Tecnologia (Opcional)	REALIZAR
Fase Galeria (Opcional)	REALIZAR
Fase Campo Minado *	REALIZAR
Fase Bloco de Gelo *	REALIZAR
Fase TCC *	REALIZAR

Figura 3.6: Tarefas do modelo de jogo **SantoGrau**

Tarefa: Fase Tecnologia

Código 3.3 lista o trecho do arquivo **process.json** relacionado à definição da tarefa responsável pela implementação das atividades de customização da **Fase Tecnologia**. Nota-se que é uma tarefa opcional (ver discussão na Seção 2.1) que possui 3 arquivos de saída (**outputs**).

Os dois primeiros arquivos (**telao.html** e **computadores.json**) referem-se aos pontos de customização da **Fase Tecnologia** (Tabela 2.1). O último arquivo de saída (**fases.json**) é um arquivo que representa as fases que estarão presentes em uma instância customizada do **Em Busca Santo Grau**. Dessa forma, a tarefa deve atualizar esse arquivo, caso a customização da **Fase Tecnologia** seja realizada.

```
{
  "name": "Fase Tecnologia",
  "uri": "faseTecnologia",
  "optional": true,
  "description": "Configurar um conteúdo multimídia com 3 palavras-chaves relacionadas ao conteúdo, que serão embaralhadas em anagramas.",
  "type": "media_list",
  "dependencies": null,
  "outputs": [
    {
      "name": "telao.html",
      "type": "html",
      "path": ""
    },
    {
      "name": "computadores.json",
      "type": "json",
      "path": ""
    },
    {
      "name": "fases.json",
      "type": "json",
      "path": "",
      "shareable": false
    }
  ]
}
```

Código 3.3: Tarefa **Fase Tecnologia** do modelo de jogo **SantoGrau**

¹² Código-fonte disponível em <https://github.com/LOA-SEAD/projeto-remar> (diretório: SantoGrau).

Tarefa: Fase Galeria

Código 3.4 lista o trecho do arquivo **process.json** relacionado à definição da tarefa responsável pela implementação das atividades de customização da **Fase Galeria**. Nota-se que é uma tarefa opcional (ver discussão na Seção 2.1) que possui 12 arquivos de saída (**outputs**).

Os onze primeiros arquivos (**quadros.json** e **1.png a 10.png**) referem-se aos pontos de customização da **Fase Galeria** (Tabela 2.1). O último arquivo de saída (**fases.json**) é o arquivo discutido na tarefa da **Fase Tecnologia**.

```
{
  "name": "Fase Galeria",
  "uri": "faseGaleria",
  "optional": true,
  "description": "Criar/editar/selecionar um tema de imagens para o nivel Galeria.",
  "type": "image_collection",
  "dependencies": null,
  "outputs": [
    {
      "name": "quadros.json",
      "type": "json",
      "path": "",
      "shareable": false
    },
    {
      "name": "1.png",
      "type": "image",
      "path": "",
    },
    // imagens 2.png a 9.png
    {
      "name": "10.png",
      "type": "image",
      "path": "",
      "optional": true
    },
    {
      "name": "fases.json",
      "type": "json",
      "path": "",
      "shareable": false
    },
  ],
}
```

Código 3.4: Tarefa **Fase Galeria** do modelo de jogo **SantoGrau**

Fase Campo Minado

Código 3.5 lista o trecho do arquivo **process.json** relacionado à definição da tarefa responsável pela implementação das atividades de customização da **Fase Campo Minado**. Nota-se que é uma tarefa que possui apenas 1 arquivo de saída que refere-se ao único ponto de customização da **Fase Campo Minado** (Tabela 2.1).

```
{
  "name": "Fase Campo Minado",
  "uri": "faseCampoMinado",
  "description": "Criar/editar/selecionar um banco de questões para o nivel Campo Minado.",
  "type": "text_database",
  "dependencies": null,
  "outputs": [
    {
      "name": "questoescm.json",
      "type": "json",
      "path": ""
    }
  ],
}
```

Código 3.5: Tarefa **Fase Campo Minado** do modelo de jogo **SantoGrau**

Fase Bloco de Gelo

Código 3.6 lista o trecho do arquivo **process.json** relacionado à definição da tarefa responsável pela implementação das atividades de customização da **Fase Bloco de Gelo**. Nota-se que é uma tarefa que possui apenas 1 arquivo de saída que refere-se ao único ponto de customização da **Fase Bloco de Gelo** (Tabela 2.1).

```
{
  "name": "Fase Bloco de Gelo",
  "uri": "faseBlocoGelo",
  "description": "Criar/editar/selecionar um banco de questões para o nível Bloco de Gelo.",
  "type": "text_database",
  "dependencies": null,
  "outputs": [
    {
      "name": "questoesbn.json",
      "type": "json",
      "path": ""
    }
  ]
}
```

Código 3.6: Tarefa **Fase Bloco de Gelo** do modelo de jogo **SantoGrau**

Fase TCC

Código 3.7 lista o trecho do arquivo **process.json** relacionado à definição da tarefa responsável pela implementação das atividades de customização da **Fase TCC**. Nota-se que é uma tarefa que possui apenas 1 arquivo de saída que refere-se ao único ponto de customização da **Fase TCC** (Tabela 2.1).

```
{
  "name": "Fase TCC",
  "uri": "faseTCC",
  "description": "Criar/editar/selecionar um banco de questões para o nível final TCC.",
  "type": "text_database",
  "dependencies": null,
  "outputs": [
    {
      "name": "questoestcc.json",
      "type": "json",
      "path": ""
    }
  ]
}
```

Código 3.7: Tarefa **Fase TCC** do modelo de jogo **SantoGrau**

3.4 Padronização do layout dos modelos de jogos

A plataforma REMAR utiliza o Materialize¹³ como o padrão para o *design* do *layout* de suas páginas. É importante que os novos modelos de jogos, a serem incorporados à plataforma, mantenham o padrão definido pela plataforma REMAR (cores, ícones, etc).

Dessa forma, a equipe de desenvolvimento da plataforma REMAR disponibilizou¹⁴ alguns exemplos de páginas web (Figura 3.7) que podem ser consultados por equipes de desenvolvimento interessados em incorporar novos modelos de jogos à plataforma REMAR.

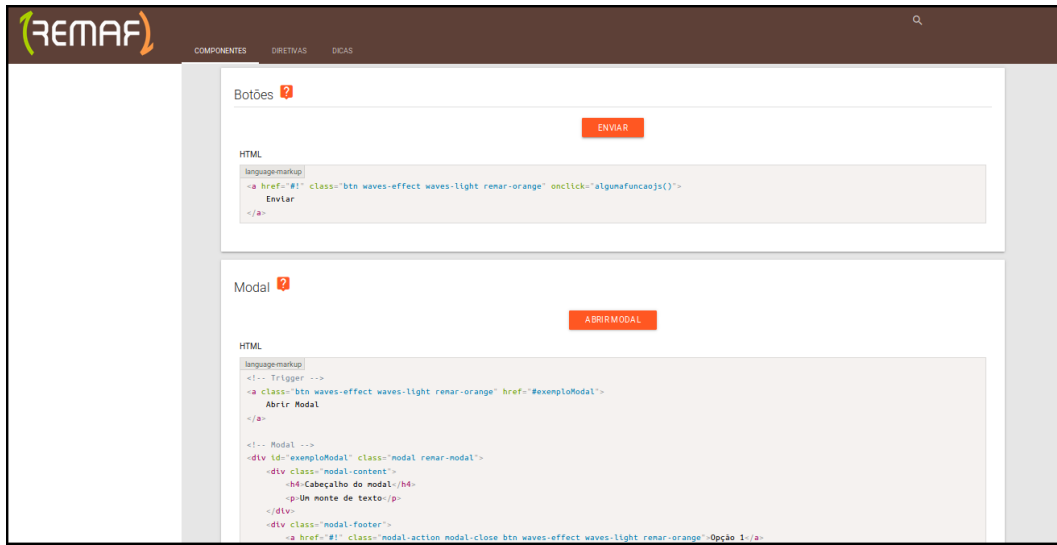


Figura 3.7: Padronização do layout dos modelos de jogos

3.5 Arquivo WAR

Todos os modelos de jogos, a serem submetidos à plataforma REMAR, devem ser compilados em um arquivo **war**¹⁵. Esse arquivo **war**¹⁶ contém uma aplicação web responsável pelas atividades relacionadas às customizações do jogo e deve possuir um diretório denominado **remar** que contém alguns itens essenciais para o modelo de jogo ser aceito na plataforma REMAR.

- o arquivo **process.json**, que foi discutido anteriormente nesse capítulo;
- o diretório **source** que contém o código executável do jogo a ser customizado;
- o diretório **images** que contém o arquivo do *banner* do jogo que será utilizado na plataforma REMAR. O formato do nome desse arquivo é **<url-do-jogo>-banner.png**.

¹³ <http://materializecss.com/>

¹⁴ Código-fonte disponível em <https://github.com/LOA-SEAD/projeto-remar> (diretório: Standardization).

¹⁵ Arquivo war (do inglês *Web application ARchive*) é um arquivo usado para distribuir uma coleção de recursos que, juntos, constituem uma aplicação web.

¹⁶ Para criar um arquivo **war** em um projeto Grails, basta executar o comando **grails war**.

Figura 3.8 apresenta a estrutura do arquivo **war** do modelo de jogo **SantoGrau** discutido na Seção 3.3.2.

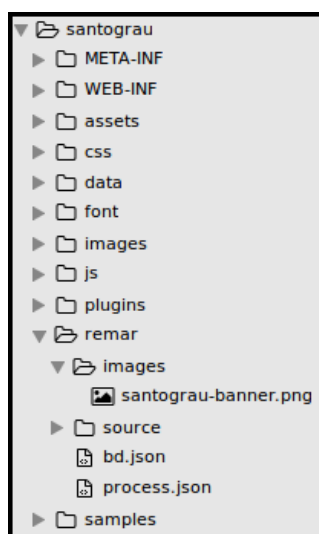


Figura 3.8: Estrutura do WAR do modelo de jogo **SantoGrau**

Para incorporar o modelo de jogo na plataforma REMAR, basta dar *upload* do arquivo **war** na área de desenvolvedor (Figura 3.9). Após o *upload*, o jogo ficará sujeito à aprovação dos administradores da plataforma. Caso aprovado, o novo modelo de jogo estará disponível para utilização pelos usuários da plataforma REMAR.

A imagem mostra a interface de upload de um arquivo WAR na plataforma REMAR. No topo, há um ícone de nuvem e o título 'Upload War'. Abaixo, há um botão laranja 'FILE' e um texto 'Selecione o arquivo do seu jogo (.war)'. Abaixo disso, há duas perguntas com ícones de informação: 'Permitir que adaptações do seu trabalho sejam compartilhadas?' e 'Permitir usos comerciais do seu trabalho?'. A primeira pergunta tem uma opção selecionada 'Sim, desde que outros compartilhem igual' com um ícone de círculo verde. A segunda pergunta tem duas opções: 'Sim' e 'Não', ambas com ícones de círculo cinza. No canto inferior direito, há um botão 'ENVIAR'.

Figura 3.9: Submeter novo modelo de jogo: Área de desenvolvedor

3.6 Considerações finais

Este capítulo teve como objetivo documentar todos os passos necessários para integrar um novo modelo de jogo no REMAR. Para melhor exemplificar os passos, foi escolhido o modelo de jogo **SantoGrau** – uma aplicação web responsável pelas atividades de relacionadas às customizações do jogo **Em Busca do Santo Grau** (Seção 2.1).

Ao final desse documento, espera-se que o leitor tenha uma visão geral do processo de integração, de um novo modelo de jogo, à plataforma REMAR:

- Identificar as possibilidades de customização do jogo (Capítulo 2);
- Desenvolver uma aplicação web, utilizando a plataforma Grails (Seção 3.2), responsável pelas atividades relacionadas às customizações do jogo; e
- Realizar a integração do seu modelo de jogo na plataforma REMAR.
 - Definir o processo de customização (arquivo **process.json**);
 - Gerar o arquivo **war** com todos arquivos necessários (**process.json**, código-fonte do jogo, etc); e
 - *Upload* do arquivo **war**.

Referências Bibliográficas

- [1] D. M. Beder, J. L. Otsuka, D. Cappelini, M. V. Fernandes, R. B. Silva, and A. R. Guido. Recursos Educacionais Multiplataformas Abertos na Rede. In *II Workshop Recursos Educacionais Abertos (WREA). Congresso Brasileiro de Informática na Educação (CBIE)*, 2015.
- [2] R. A. Bordini, J. L. Otsuka, D. M. Beder, A. E. R. de Camargo, L. Valério Neto, and M. Tsuda. Design de Em Busca do Santo Grau – Jogo Eletrônico Educacional Customizável. In *XV Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGAMES)*, 2016.
- [3] D. Köenig, A. Glover, P. King, G. Laforge, and J. Skeet. *Groovy in Action*. Manning Publications Co., Greenwich, CT, USA, 2007.
- [4] G. Krasner and S. Pope. A Cookbook for Using the Model-View Controller User Interface Paradigm in Smalltalk-80. 1(3):26–49, 1988.
- [5] J. L. Otsuka, D. M. Beder, M. V. Fernandes, L. Bocanegra, M. Mourão, and R. A. Bordini. Uma Plataforma de Apoio à Publicação e Customização de Jogos Educacionais Abertos. In *XIII Congresso Brasileiro de Ensino Superior a Distância (ESUD)*, 2016.
- [6] R. S. Pressman. *Engenharia de Software: Uma Abordagem Profissional*. Bookman, São Paulo, 7a edition, 2011.
- [7] G. Smith and P. Ledbrook. *Grails in Action*. Manning Publications Co., Greenwich, CT, USA, 2009.
- [8] H. L. Weissmann. *Falando de Grails – Altíssima produtividade no desenvolvimento web*. Casa do Código, São Paulo, 2015.