RENAN FARACO TEIXEIRA

DESENVOLVIMENTO DE UMA INTELIGÊNCIA ARTIFICIAL PARA UM JOGO DE ESTRATÉGIA BASEADO EM TURNOS PARA ANDROID

LAVRAS - MG

2014

RENAN FARACO TEIXEIRA

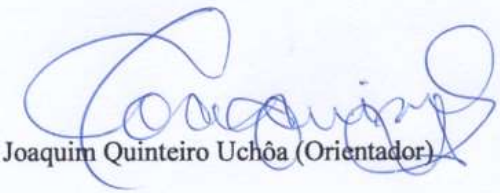
DESENVOLVIMENTO DE UMA INTELIGÊNCIA ARTIFICIAL PARA UM JOGO DE ESTRATÉGIA, BASEADO EM TURNOS PARA ANDROID

Monografia de graduação apresentada ao Colegiado do Curso de Bacharelado em Ciência da Computação, para obtenção do título de Bacharel.

APROVADA em 7 de julho de 2014.

André Pimenta Freire

Bruno de Oliveira Schneider



Joaquim Quinteiro Uchôa (Orientador)

LAVRAS-MG
2014

RESUMO

O trabalho apresenta o processo de criação de uma inteligência artificial(IA) para um jogo de estratégia baseado em turnos para dispositivos moveis com o sistema operacional Android. Foram pesquisadas técnicas de IA utilizadas em jogos eletrônicos e aplicadas três dessas técnicas, a saber: Algoritmo de Busca, Sistema Baseado em Regras e Sistema Baseado em Casos, para implementar, respectivamente, as ações de: andar, tomar decisões e entrar em combate. Após a construção do código foram feitos testes para balancear os pesos (que norteiam a escolha de uma ação) para tornar a tomada de decisão mais competitiva. Como resultado foi obtido um software capaz de escolher entre as ações e integrar as três técnicas aplicadas.

Palavras-chave: Jogos para Dispositivos Móveis. Jogo de Estratégia em Turnos. Inteligência Artificial. Sistema Baseado em Regras. Algoritmo de Busca. Sistema Baseado em Casos

LISTA DE FIGURAS

SUMÁRIO

1 Introdução

Os aparelhos com o Sistema Operacional (SO) Android[1]ocupam a maior parcela do mercado de celulares, sendo que a cada 5 celulares vendidos no mundo, 4 possuem tal Sistema. No terceiro trimestre de 2013 este sistema da Google rodava em 81,3% dos celulares inteligentes comercializados (G1, 2013).

No quesito de jogos para essa plataforma, o desenvolvimento é amplo devido à crescente procura deste Sistema Operacional e da facilidade em se criar *software*. Além disso, o desenvolvedor pode projetar para o SO sem se preocupar em qual *mobile* o *software* irá rodar (ZECHNER, 2011).

Num âmbito mais especifico de jogos, o mercado no Brasil, apesar de ser bem tímido em relação ao panorama internacional e ser imaturo e recente, apresenta crescimento constante e está em segundo lugar no mercado de jogos na América latina (LOBO, VERDI & ELIAS, 2012).

Tendo em vista a grande popularidade do Sistema Android, da facilidade em desenvolver programa para tal plataforma e o mercado crescente de jogos, foi desenvolvido um projeto de criação de um jogo de estratégia baseado em turnos para Android. Este foi modularizado em três projetos menores: interface, editor de níveis e inteligência artificial, sendo que o modulo de interface foi desenvolvido por Marlon Jonas de Oliveira Lima[2]e o módulo de editor de níveis por Jeferson Barrile Tomazella[3]. Neste projeto será criado o modulo de inteligência artificial.

No desenvolvimento de um jogo, um dos principais diferenciais é a criação de uma inteligência artificial, pois ela tem como objetivo controlar as

[1] O Android™ (http://www.android.com/) é um produto da Google, Inc (http://www.google.com/about/company/).
[2] Graduado em Ciência da Computação pela Universidade Federal de Lavras em 2013.
[3] Aluno de graduação do curso de Ciência da Computação na Universidade Federal de Lavras.

ações feitas pelo computador para que tomem ações que simulem inteligência que por sua vez tem como função tornar o jogo mais desafiador e divertido (ROUSSE, 2001).

1.1Objetivos

O objetivo geral deste trabalho foi o desenvolvimento de uma inteligência artificial(IA) para um jogo baseado em turnos para Android. Para realização deste objetivo foi estabelecido um grupo de objetivos específicos:

1. Desenvolvimento do *design* do jogo para obter as informações da mecânica relevantes para a inteligência.
2. Analise da mecânica do jogo para determinar as necessidades e técnicas para a IA.
3. Tendo em vista as diferentes necessidades de IA, estudar as técnicas usadas em jogos eletrônicos e eleger as melhores para cada parte do jogo.
4. Implementação das técnicas escolhidas nas três divisões da IA:
    a. Movimentação: aplicação do Algoritmo de Busca
    b. Batalha: aplicação do Raciocínio Baseado em Casos
    c. Decisões Gerais: aplicação do Sistema Baseado em Regras
5. Integração das três divisões da IA e testes da mesma.

2 Referencial teórico

Para a execução do projeto e seus objetivos foram utilizadas e estudadas as ferramentas e métodos a seguir:

2.1Android

A plataforma Android, de acordo com Júnior et al (2011), é um sistema operacional baseado em Java que é executado no *kernel* do Linux. O sistema é muito leve e com muitos recursos. Os aplicativos do Android são, segundo Dimarzio (2008), desenvolvidos utilizando Java e podem ser portados com bastante facilidade e também incluem aceleração 3D com motor gráfico (baseado no suporte de *hardware*), suporte de banco de dados e um navegador web integrado.

O Android é um sistema operacional *open-source*, baseado em Linux, destinado a equipamentos móveis, e foi desenvolvido inicialmente pela Google e posteriormente pela Open Handset Alliance[4]. O sistema possui uma rica e atraente interface gráfica, que apresenta uma grande diversidade de aplicações, navegador de internet, banco de dados integrado, jogos, integração com outros sistemas, como GPS, conectividades diversas, como *Bluetooth*, EDGE, 3G e *Wi-Fi* e entre outras características (SCOTA et al, 2010)

O sistema operacional Android é dividido em cinco seções principais, conforme mostrado na Figura 1: *Kernel* do Linux: que controla as funções mais básicas do dispositivo; Bibliotecas: contém bibliotecas de funções básicas como, por exemplo, som e gráfico; AndroidRuntime: máquina virtual Java modificada;

[4]http://www.openhandsetalliance.com/

Quadro de Aplicação: responsável pelo gerenciamento de funções de alto nível, como janelas; Aplicações: camada na qual os aplicativos que estão no aparelho Android se encontram (LEE, 2011).

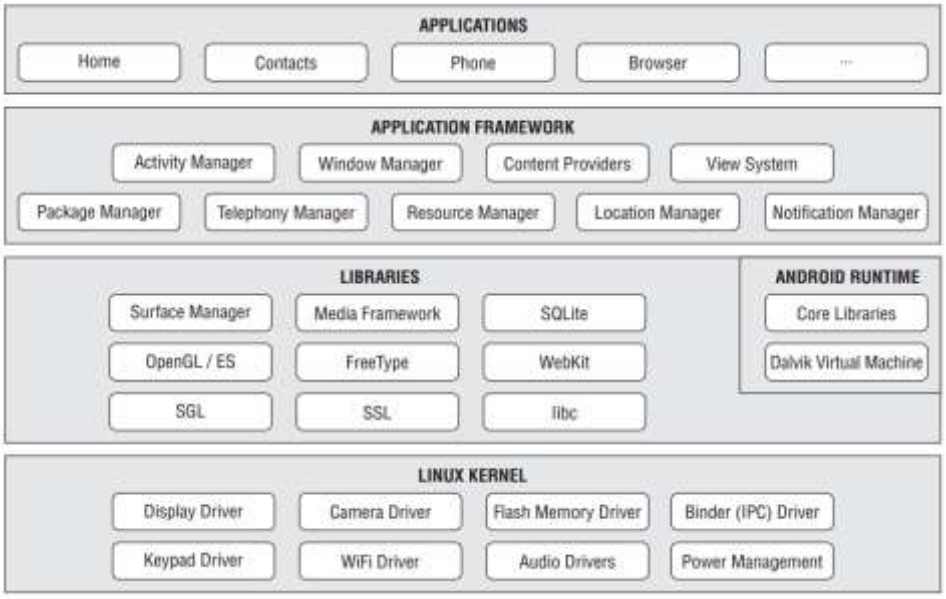


Figura 1 Arquitetura do Android

Fonte (LEE, 2011)

2.2Jogos de Estratégia Baseados em Turnos

Jogos de estratégia são jogos em que o jogador ou a inteligência artificial deve tomar decisões para administrar recursos, como personagens, recursos naturais e tecnologias, com o intuito atingir certo objetivo, que incluiu a derrota de um oponente ou aquisição de certo item (FULLERTON, 2008).

A estratégia baseada em turnos se difere das demais modalidades de estratégia pelo fato que as ações não acontecem em tempo real. Cada atitude de cada jogador, seja ele humano ou máquina, é feita em momentos diferentes e

limitado a cada jogador, chamado turno. O processo de alternância de turnos é baseado em alguma regra especifica presente no jogo e é de maneira cíclica, em outras palavras, quando todos jogarem seus turnos a vez do primeiro volta.

2.3 Inteligência Artificial Acadêmica e *Game-IA*

A Inteligência Artificial (IA) é classificada, de acordo com Luger (2004), como uma área da Ciência da Computação que se atém ao comportamento inteligente. A área de Inteligência Artificial busca colocar ou amplificar o potencial intelectual do ser humano em resolver problemas usando métodos, práticas e dispositivos computacionais

Inteligência Artificial pode ser dividida em duas áreas: a IA acadêmica visa alcançar as melhores soluções e meios de se resolver problemas complexos e tenta representar, com o máximo de similaridade, o conhecimento humano em sistemas eletrônicos; IA nos jogos, ou *Game-IA*, que é uma adaptação da IA acadêmica e está relacionada com jogos eletrônicos e tem como objetivo levar desafio e diversão ao usuários (KISHIMOTO, 2004).

Pozzer (2006) explica que uma boa *Game-IA* não visa obter o melhor resultado possível, já que ao se focar nesse melhor resultado, os jogadores humanos iriam se frustrar por não serem capazes de poderem enfrentá-la. Pozzer(2006) também afirma que se devem incluir erros na *Game-IA*, para que esta não seja necessariamente esperta, e sim apresente competitividade entre os jogadores humanos, criando um equilíbrio entre erros plausíveis e comportamento inteligente.

2.4 Jogos Eletrônicos e a *Game-IA*

No princípio do desenvolvimento de jogos eletrônicos, a IA era mais conhecida como "programação de jogabilidade", devido a não existência de atitudes inteligentes demonstrada pelos personagens manipulados pelo computador. As primeiras técnicas usadas em jogos eletrônicos, nas décadas de 1970 e 1980 eram padrões de movimento ou movimentos repetitivos e/ou aleatórios, como por exemplo, *Qwak*, *Space Invaders* e *Pac-Man*(SCHWAB, 2004).

Em 1989, métodos de *Artificial Life* começam a ser usados por jogos *Sims*[5]. No início da década de 1990, os jogos de estratégia (como, por exemplo, *Civilization*) foram os primeiros a usar técnicas para controlar grupos de personagens e elaborar táticas, pois para serem viáveis necessitavam de uma boa IA (TOZOUR, 2002).

Schwab (2004) aponta que, em meados da década de 1990, ocorre o lançamento do primeiro jogo comercial a usar Redes Neurais Artificiais, o *BattleCruiser*: 3000AD.

Em 2000, a empresa Maxis[6], utiliza, no jogo *The Sims*[7], maquinas de estado *fuzzy*, além dos métodos usados por seus predecessores (TOZOUR, 2002).

A partir de 2001, os jogos passaram a apresentar mais de uma técnica de Inteligência Artificial, como em *Black & White* que utilizava várias aplicações de redes neurais(SCHWAB, 2004).

2.5Técnicas e Algoritmos usados em *Game-IA*

[5] Simuladores de atividades corriqueiras, como fazendas e relações entre pessoas.
[6] Empresa responsável pela criação dos jogos Sims (http://www.maxis.com/)
[7]http://www.thesims.com/

As técnicas principais usadas para *Game-IA*de acordo com Dalmau (2004), citadas por Kishimoto(2004), são: maquinas de estado, sistema baseado em regras, algoritmo de busca e algoritmo genético. De acordo com Paula (2007), o sistema baseado em caso, ou raciocínio baseado em caso, também possui grande eficácia em jogos eletrônicos.

Neste projeto foram utilizados: Algoritmo de Busca: para a movimentação pelo cenário; Sistema Baseado em Casos: para as escolhas durante o combate e; Sistema Baseado em Regras: para controlar as escolhas do jogador artificial quanto à construção, compra e recrutamento de unidades.

2.5.1 Algoritmo de Busca

O problema de busca em jogos eletrônicos, segundo Bourg & Seeman(2004) e Rodrigues, Marchi& Dias(2013), é a movimentação pelo cenário: deslocar-se entre locais e desviar de obstáculos. Caso essa busca seja deficitária a diversão do jogo pode ser prejudicada, e a IA pode apresentar comportamentos que não demonstrem inteligência.

Dalmau (2004) cita que para a resolução do problema de busca o Algoritmo de Busca A*[8] é muito utilizado e conhecido. Karlsson (2006) explica que tal algoritmo encontra o caminho mínimo entre dois pontos no mapa, se um caminho existir. O mapa é dividido em nós, ou células, que são os representantes do posicionamento dos elementos. A busca é feita nesse grafo, e o caminho será uma resultante na forma de uma lista de pontos.

A equação base usada no cálculo do menor caminho é:

$$f(n) = g(n) + h(n)$$

[8] A*, geralmente pronunciado como a-estrela.

Sendo que *g(n)* é o custo associado em movimentar do ponto inicial até o ponto *n*; *h(n)* é a distância de entre *n* e o ponto final e; *f(n)* é a função que retorna o valor do caminho. O algoritmo é rodado várias vezes até se chegar ao ponto final ou não se encontrar caminho. Dos vários caminhos testados *f(n)* retorna o menor (BORGES, BARREIRA& SOUZA, 2009).

2.5.2 Sistema Baseado em Regras

Em um Sistema Baseado em Regras(SBR), o conhecimento é definido através de um conjunto de parâmetros, ou variáveis, e um grupo de regras trabalha sobre esses parâmetros. Ao serem processadas essas regras, uma "tomada de decisão" é resultada (KARLSSON,2006).

As variáveis, de acordo com Borges, Barreira, Souza (2009), são características importantes do jogo e sua estrutura dentro do SBR é: "condição →ação".

Exemplo de regras:

- Fome & osso por perto →comer;
- Fome & não osso por perto →procurar;
- Não fome &sono →dormir;
- Não fome & não sono →andar e latir.

2.5.3 Sistema Baseado em Casos

Sistema Baseado em Casos (ou Raciocínio Baseados em Casos, ou SBC) é um método para resolver problemas que reutiliza problemas similares anteriormente resolvidos para buscar a solução de um novo problema (RIESBECK&SCHANK, 1989, citado por PAULA, 2007).

O SBC utiliza-se de uma memória, a qual é uma base com conhecimentos indexados e usados para armazenar problemas e soluções. Uma experiência (resolução de um problema) é armazenada sobre a forma de casos em uma biblioteca, que pode ser alimentada por novos casos ao passo que novos problemas são solucionados. E por sua vez, casos são situações vividas pelo sistema e permitem uma representação simplificada de uma situação real (PAULA, 2007).

2.6América Tribal

América tribal, segundo o *design* de Lima (2013) apresentado no Anexo A, é um jogo de estilo estratégia baseado em turnos desenvolvido para a plataforma Android, onde o jogador pode escolher entre três tribos baseadas nas tribos americanas. Essas tribos são divididas no jogo em três tipos: Norte, Centro e Sul, sendo baseadas respectivamente, em tribos como Cherokee, Apache, entre outras (do norte americano), como Incas e Maias (da América central e do sul) e como Guarani e a Pataxó (do território brasileiro).

2.6.1 Mecânica do Jogo

O cenário do jogo é dividido em células quadriculadas tendo regiões vazias(passagens), recursos (comida, madeira e pedras) e bloqueios (montanhas, florestas e água).

O jogador pode escolher entre três tribos (acima descritas), três dificuldades e três mapas distintos. O jogo começa com uma aldeia e um explorador e seguindo a base de turno (cada jogador tem sua vez), e nesses turnos o jogador pode tomar ações e gerenciar os recursos. As ações são: andar, coletar recurso, construir prédios, comprar soldados, recrutar soldados, lutar com

o explorador inimigo e lutar com o a aldeia inimiga; este último, caso o jogador atacante vença a disputa também ganha o jogo.

Para o combate o jogador tem cinco opções de unidade: básica (atacam corpo a corpo), de longo alcance (atacam com armas de longo alcance), animal (animais típicos de cada cultura), mística (sacerdotes de cada cultura) e mítico (divindades de cada cultura). Na aldeia, o jogador tem cinco opções, enquanto no explorador apenas três opções. O sistema de combate também funciona baseado em turnos: uma ordem pré-determinada de ataque das unidades é usada para saber de qual unidade será a vez, o jogador do turno atual escolhe qual será seu alvo, aquele que perder todas as unidades primeiro perde a disputa.

2.7libGDX

O *libGDX*[9] é um *framework* de desenvolvimento de jogos em Java, desenvolvido pela empresa Badlogic Games[10]. Este *framework* pode exportar os jogos criados para diversas plataformas, como Windows, Linux, iOS, Android e HTML5, utilizando um mesmo código base.

O libGDX possui bibliotecas que auxiliam o gerenciamento de código de baixo nível: gráfico, como gerenciamento de câmeras, carregamento, descarregamento e apresentação de imagens e integração com o OpenGL; som, como o gerenciamento de efeitos sonoros e músicas; entrada de periféricos, como o *mouse*, o teclado e o *touch*; e gerenciamento de arquivos texto ou de dados.

Todo o código da aplicação feito em libGDX é separado do código de exportação para cada plataforma, sendo este último criado automaticamente, e o primeiro é um código único, independente de plataforma.

[9] http://libgdx.badlogicgames.com/
[10] http://www.badlogicgames.com/wordpress/

3 Metodologia

Este projeto teve início em conjunto com o início do projeto do jogo América Tribal. Inicialmente, para o desenvolvimento do projeto, foi feito o levantamento dos requisitos do *software* que consistem em elaborar um jogo de estratégia em turnos para Android. Seguindo-se a isto foi elaborado o *game design*[11] (GD), contendo a descrição completa das informações relevantes da mecânica e dos componentes do jogo, como personagens e cenários.

Com o GD, foi feita a análise e elaboração de como seriam aplicadas as técnicas de IA para simular as mecânicas do jogo de forma a propor um desafio ao usuário. Após tal análise foi determinado que a IA seria dividida em três áreas do jogo: Movimentação, Batalha e Decisões Gerais.

O próximo passo foi a busca e estudo das várias técnicas usadas em *Game-IA* para determinar a melhor para cada uma das três áreas do jogo. As técnicas escolhidas foram: Sistemas Baseados em Regras, Algoritmo de Busca e Sistema Baseado em Regras.

Para as Decisões Gerais foi escolhido a técnica Sistemas Baseados em Regras. Esta área consiste em decidir qual edifício construir, qual recurso buscar, e qual soldado comprar e recrutar. Para isso o Sistema Baseado em Regras foi útil tendo em mente que as decisões são ativadas por um conjunto de regras, a exemplo:

- "recurso crítico→ determinar qual recurso=> buscar recurso",
- "recurso não critico => escolher qual prédio construir => construir".

Para a área de Movimentação foi usado a técnica Algoritmo de Busca, mais especificadamente o Algoritmo A*. Tal algoritmo foi eleito visando ser a

[11] O documento de *game design* do jogo America Tribal pode ser encontrado em LIMA,(2013)

mais comumente usado e com baixo custo de processamento, para jogos. Os dados que são relevantes para essa área são os possíveis destinos: recursos, base, inimigo e base inimiga;ou obstáculos. O destino a ser decidido é escolhido pelas Decisões Gerais.

Para a Batalha, foi usada a técnica Sistema Baseado em Casos. Nesta técnica os casos a serem guardados são uma sequência de números referentes as classes envolvidas no combate: “classe do atacante” “classe do defensor” “classes dos inimigos restantes” que inicialmente é escolhido calculando a efetividade do ataque levando em conta os atributos do atacante e do defensor. Para julgar se um caso específico verifica-se a porcentagem de dano causado, sendo superior a um valor de comparação o caso é armazenado, sendo inferior descartado e se o caso já esteja armazenado não será necessário um novo cálculo. Contudo, o valor da porcentagem do novo ataque é usando novamente para se fazer uma média com a porcentagem anterior.

Por fim, para foi usado as funções do libGDX para implementar a parte gráfica do código. Esta parte simplesmente mostra, de forma visual, o cenário, os objetos, e os personagens dos dois jogadores, sendo possível observar os caminhos que a IA escolhe, assim como as ações de coleta de recursos. Um exemplo de cenário é apresentado na Figura 2.

Figura 2 Exemplo de cenário

3.1Estrutura do código

O código possui sete classes principais, apresentadas na Figura 3, que são: Guerra, CarregarMapa, Mapa, Lista, Jogador, Regras, IA e Vila. Para mostrar o mapa (com recursos, bloqueio e casas) e auxiliar os comandos vindos de periféricos para passar para o próximo turno, foi utilizado o *framework*

libGDX, implementando as interfaces ApplicationListener[12] e InputProcessor[13] na classe IA.

A classe Regras é responsável por usar das informações contidas em Mapa, Jogador, Guerra e Vila a qual serve para determinar e executar as decisões tomadas, as quais são: construir, comprar, recrutar e andar. Guerra é responsável por gerenciar os métodos envolvidos no combate.

Na classe Mapa, está contida a localização dos recursos (bem como seus tipos e quantidades), dos obstáculos, dos personagens e bases, tendo todas essas informações obtidas de um arquivo texto usando a classe CarregarMapa.

A classe Lista contém o algoritmo de busca, responsável por fazer o personagem buscar recursos, retornar à base ou ir à batalha que é invocada pela classe Regras.

A classe Jogador contém todas as informações de cada jogador como posição, prédios possuídos, destino, número de soldados com o herói e a classe Vila.

Por fim a classe Vila possui todas as informações da base, como número de recursos, quanto de bônus cada recurso possui, número de soldados na base e possui funções auxiliares para a execução das decisões como comprar, construir e andar.

[12] Interface responsável por gerenciar o ciclo de vida da aplicação, ou seja, quando ela será desenhada na tela, quando ela será iniciada, e quando será finalizada.
[13] Interface responsável por gerenciar a entrada de teclado, *mouse* e *touch*.

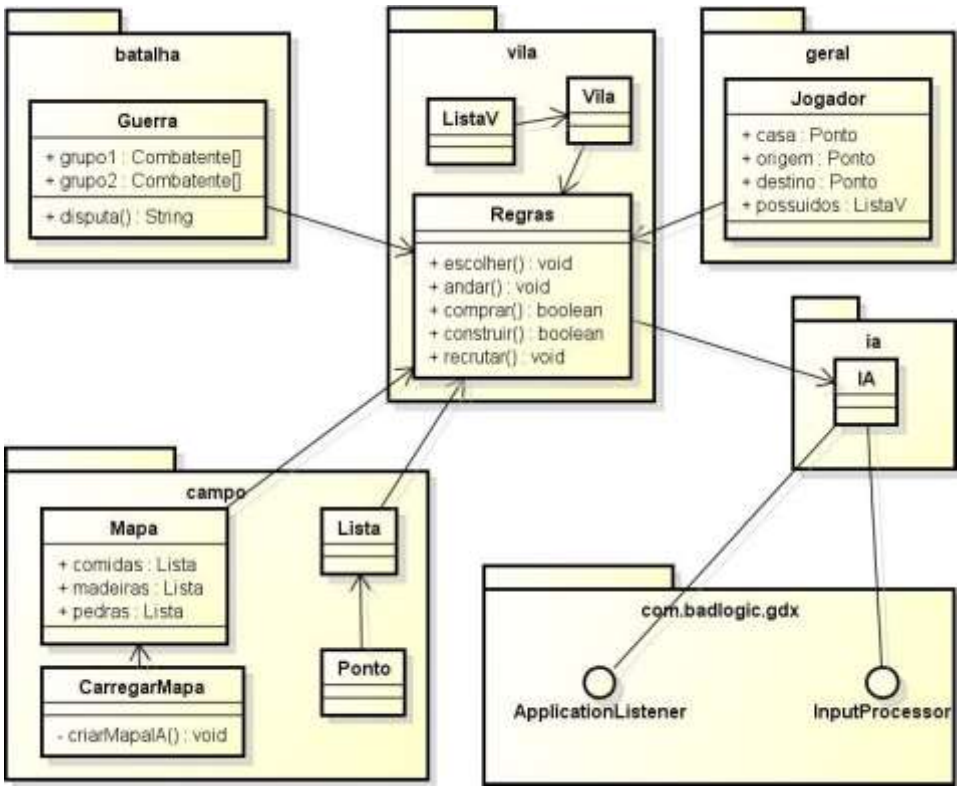


Figura 3 Diagrama de classe

3.1.1 Descrição das classes

3.1.1.1 br.ufla.americatribal.batalha

3.1.1.1.1 Personagem.java

A classe Personagem, apresentada no Apêndice D, é responsável por armazenar todos os atributos básicos dos soldados bem como fazer a inicialização dos valores para cada atributo.

Esta desenvolvida para resolver o problema de armazenar os dados de todas as umas das diferentes unidades.

3.1.1.1.2 Combatente.java

A classe Combatente, mostrada no Apendice D, tem como função armazenar os dados da classe personagem juntamente com as variáveis intrínsecas da batalha, como a quantidade de soldados e a qual time pertence.

Também é responsável por executar o ataque ao inimigo e fazer a escolha por meio do calculo de qual será o alvo.

3.1.1.1.3 ListaB.java

A classe ListaB , mostrada no Apêndice D, é utilizada para manipular a classe Combatente em forma de lista para facilitar estruturar a ordem de ataque.

3.1.1.1.4 Guerra.java

A classe Guerra, aprensentada no Apêndice D, armazena os dados dos dois grupos durante a bataha, bem como o banco de casos e a lista de combatentes em ordem de ataque.

Possui métodos resposaveis pela escolha feita utilizando o sistema de casos, estabelecerem a ordem de ataque, estruturar a mecânica do combante (fazer a lista ser rotacinada a cada ataque), determinar a qual será o atacante e gerar os casos a serem armazenados.

3.1.1.2 br.ufla.americatribal.campo

3.1.1.2.1 Ponto.java

A classe Ponto, apresentada no Apêndice B, armazena todos os dados importantes relacionados a cada elemento da matriz mapa, como o custo daquele ponto, qual seu antecessor e o tipo e quantidade de recurso nele contido.

As funções dessa classe são para auxiliar na manipulação de Ponto nas demais classes como obter os pontos adjacentes e se existem todos esses pontos e determinar os custos associados a cada ponto.

3.1.1.2.2 Lista.java

A classe Lista, demonstrada no Apêndice B, manipula a classe Ponto no formato de lista para implementar o Algoritmo de Busca A*.

Os métodos dessa classe são usados para dar suporte ao A*.

3.1.1.2.3 CarregarMapa.java

A classe CarregarMapa, mostrada no Apêndice B, é resposável por armazenar os dados obtidos do mapa, apresentados nos Anexos B e C, e adapta-los para as demais classes e funções do projeto.

3.1.1.2.4 Mapa.java

A classe Mapa, explicitada no Apêndice B, contém dados das posições dos objetos e recursos no mapa, bem como uma codificação para determinar como os recursos serão tratados e manipulados pelas funções.

Possui métodos pra determinar se um ponto é passável, se está dentro dos limites, determinar qual, dentre os de interesse do jogador, é o ponto mais próximo e listar todas as localizações dos recursos.

3.1.1.3 br.ufla.americatribal.geral

3.1.1.3.1 Jogador.java

A classe Jogador, mostrada no Apêndice C, armazena todos os dados relevantes ao jogador como: localiações de casa, posição atual, destino, identificador do jogador (um ou dois), nível tecnilógico atual, prédios possuídos e possíveis prédios a se construir e os valores de soldados que acompanham o herói.

Possui métodos para alterar a necessidade de cada recurso, mostrar os dados relevantes no terminal e inicializar as posições da casa e de inicio.

3.1.1.4 br.ufla.americatribal.ia

3.1.1.4.1 IA.java

A classe IA, demostrada no Apêndice E, armazena os dados do libGDX para o carregamento do mapa, como câmera e desenhos de texturas, bem como é responsável por estabelecer qual jogador será primeiro e estruturar a alternância de turnos.

3.1.1.4.2 Config.java

A classe Config, mostrada no Apêndice E, configura a resolução para o modulo gráfico do libGDX.

3.1.1.5 br.ufla.americatribal.vila

3.1.1.5.1 Predio.java

A classe Predio, apresentada no Apêndice A, armazena todos os atributos das construções como: atratividade, classe(militar, base ou recurso), nível, peso, lista de prédios necessários e tipos de bônus.

A classe possui funções de inicialização de todos os prédios com seus respectivos valores.

3.1.1.5.2 ListaV.java

A classe ListaV, mostrada no Apêndice A, manipula a classe prédio em formato de lista, com funções como gerar uma sub lista com prédios de um certo nível, inserir todos os prédios e seus requisitos em lista e encontrar um prédio específico.

3.1.1.5.3 Vila.java

A classe Vila, mostrada no Apêndice A, contém os dados referentes à base do jogador como soldados disponíveis para comprar, soldados na base, quantas unidades aumentará ao final de sete turnos e quantidade de recursos.

Possui funções para auxiliar a comprar de soldados, a construção de prédios, à escolha de qual recurso será buscado, bem como liberar a construção de um prédio(verificar se os recursos são suficientes).

3.1.1.5.4 Regras.java

A classe Regras, apresentada no Apêndice A, contém variáveis auxiliares utilizadas no sistema de regras como exemplo o número de passos, se é possível comprar e se é possível comprar.

Nesta classe são implementados os métodos que determinam e executam as escolhas. A saber, as utilidades de cada método principal:

- Ajustar Pesos: atualiza o peso dos soldados junto do herói após a batalha
- Andar: desloca o personagem pelo cenário
- Atratividade: atribuiu os valores de importanticia a cada prédio
- Coletar: recolhe recursos do cenário
- Comprar Soldados: retira os soldados do banco de soldados disponíveis à compra e os coloca na base
- Escolher: método principal que encadeia todas as regras
- Proximidade: verifica a distancia entre soldados
- Qual Construir: determina qual será o prédio a ser constuido
- Recrutar Soldados: retira os da base e coloca no grupo do herói
- Teste e Construção: verifica se a tecnologia associada ao recurso em falta pode ser construída permitindo que outro recurso seja buscado.

3.2Sistema de Regras

O Sistema de Regras, como mostrado no Apêndice A, é composto por um conjunto de variáveis que são:

- Numero de prédios disponíveis, no nivel atual, para construção
- Existência de algum recurso crítico
- Destino do herói
- Nivel de recurso crítico
- Número de soldados no herói
- Posicionamento do herói
- Predios já construídos
- Ações disponíveis
- Número de soldados na base
- Quantidade de recursos
- Limite de gasto de recursos
- Pesos de predios possuidos
- Pesos de soldados na base
- Pesos de soldados junto ao herói
- Nivel de dificuldade
- Estado atual do jogador

Sobre cada uma dessas variáveis é utilizadas um grupo de regras para determinar qual ação será escolhida. As regras são:

Aumentar o nivel de prédios a serrem escolhidos: se o número de prédios disponíveis no nivel atual de construções for igual a zero, o nivel é aumentando em um.

Nivel de falta de recurso: se algum recurso esta abaixo de cinco, o tipo do recurso é armazenado para consultas futuras e um valor de recurso critíico vai para dois. Se o menor falar de algum recurso for mais que cinco, porém menos

que trinta o valor de recurso critico é um. Se todos os recursos estiverem acima de trinta o recurso critíco vai à zero.

Pegar recursos: se o recurso não estiver em nivel satisfatório e o herói não estiver indo atacar o inimigo, será executada a instrução de pegar recurso

Atacar o heroi inimigo: se o recurso não estiver no vermelho, se estou heróis estiverem próximos e se os soldados no herói forem em numero suficiente, este irá andar em direção ao inimigo e ao encontrar-se no mesmo espaço que ele, atacará.

Atacar a base inimiga: se o recurso não estiver no vermelho, se estou heróis estiverem próximos e se os soldados no herói forem em numero suficiente, este irá andar em direção a base e ao encontrar-se no mesmo espaço que ela, atacará, caso seja vitorioso, vencerá o jogo.

Possibilidade de comprar soldados: caso algum prédio militar(capaz de produzir unidade) esteja na lista de prédios possuidos, será possível escolher a ação comprar soldado

Possibilidade de recrutar soldados: caso seja possível comprar soldados, caso o número de soldados na base de qualquer classe é maior que zero, será possivel recrutar soldados

Escolher uma ação entre construir, comprar e recrutar: se todos os recursos estiverem acima dos limites para comprar e construir, é feita a escolha da ação que será construir se os pesos dos predios possuidos for o menor, comprar se os pesos de soldados da base for o menor e caso o peso de soldados junto ao herói for o menor, recrutar

Atratividade: para determinar qual prédio será construído, é atribuído a cada prédio um valor de atratividade que depende do nível de dificuldade, da quantidade de tipos de bônus e de quanto recurso será preciso

Construir: é escolhido o prédio com maior atratividade e se este possúi os requisistos e o jogador detem os recursos necessários, tal prédio é construído.

Comprar: a princípio determina-se qual classe de soldados está mais em falta na base, depois se determina quantos soldados serão necessários, em seguida quantos dessa quantia serão possíveis devido à quantidade de recursos e por fim feita a compra dessa quantidade efetiva. Caso a base esteja vazia, será comprada a classe de mais baixo nível se estiver na dificuldade fácil, a de nível mais elevado se estiver no difícil e será sorteada a classe se estiver no médio

Recrutar: caso o soldado esteja na base, determina qual classe está mais em falta (ou escolhe uma classe de acordo com a dificuldade caso não haja soldados), determina a quantidade de soldados a serem transferidos para o herói e o recrutamento é feito.

3.3Testes

Os testes foram feitos em cada uma das três áreas separadamente e com a sinconização das mesmas.

O algoritmo de busca, na fase de testes, foi alimentado com quatro exemplos de mapa. O primeiro exemplo foi um mapa cinco por cinco sem obstáculos ou recursos, para testar a funcionalidade básica do código. O segundo exemplo foi um mapa com uma coluna de obstáculos e apenas um quadrado de passagem para o lado oposto ao início para testar a busca por caminhos de forma básica. Os terceiro e quatro exemplo, como mostrado no Anexo B e Anexo C, são para testar o algoritmo em seu objetivo real com obstáculos e recuros.

O algoritmo de casos foi alimentado para testes com diversas combinações de soldados e quantidades. Um exemplo de saída é apresentado na sessão de resultados.

Por fim, o sistema de regras foi testado tanto isolado dos outros módulos quanto em sincronia. Informações como dados dos soldados e prédios foram

obtidos do *game design* apresentado no Anexo A. Para os testes de escolha de ação e determinar o nível do recurso foram obtidos pesos de forma impírica.

Os valores dos pesos para soldados são:

- Classe 1, Unidade Básica: 4
- Classe 2, Unidade de Longa Distância: 5
- Classe 3, Unidade Animal: 6
- Classe 4, Unidade Mística: 8
- Classe 5, Unidade Mítica: 12

Os pesos para cada prédio são:

- Nivel 1, Aldeia 1: construida desde o início;
- Nivel 2, Arêna 1: 8;
- Nivel 2, Comida 1: 5;
- Nível 3, Aldeia 2: 12;
- Nível 3, Comida 2: 10;
- Nível 3, Pedra 1: 5;
- Nível 3, Madeira 1: 5;
- Nível 3, Estábulo 1: 18;
- Nível 3, Arena 2: 13;
- Nível 3, Defesa 1: 12;
- Nivel 4, Aldeia 3: 32;
- Nivel 4, Comida 3: 15;
- Nivel 4, Pedra 2: 10;
- Nivel 4, Madeira 2:
- Nivel 4, Templo: 23;
- Nivel 4, Totem 1: 10;
- Nivel 4, Defesa 2: 15;
- Nivel 5, Aldeia 4: 40;

- Nivel 5, Pedra 3: 10;
- Nivel 5, Madeira 3: 15;
- Nivel 5, Estabulo 2: 18;
- Nivel 5, Totem 2: 9;
- Nivel 6, Totem 3: 27;
- Nivel 7: Defesa 3: 14;

3.4 Materiais

Para o desenvolvimento e testes das técnicas de Inteligência Artificial foi usando um computador *desktop* com 4GB de memória RAM, processador Pentium® Dual-Core de 2.5 GHz, com 300GB de espaço de armazenamento, utilizando um Sistema Operacional de 64-bit.

Foi utilizado como linguagem de desenvolvimento o JAVA, usando-se do IDE Eclipse[14] com integralização ADT Bundle[15], para manipulação de *software* para Android. Em conjunto com estes foi usado o *framework*libGDX para visualização do mapa e dos movimentos.

[14]http://www.eclipse.org/
[15]http://developer.android.com/sdk/index.html

4 Resultados e discussão

4.1Resultados

As Figuras 4 e 5 representam as escolhas e resultados de duas das três técnicas aplicados nesse projeto: Algoritmo de Busca e Sistema Baseado em Regras. Nas primeiras linhas são apresentadas as informações relevantes dos jogadores como número de recursos, soldados na base e no herói e prédios já construídos,

Após isso, é apresentado qual recurso está em mais falta. Em seguida é mostrado a origem e destino do herói. Nas linhas onze de cada imagem é mostrado o resultado da técnica Algoritmo de Busca. Na próxima linha aparece qual recurso coletado, a quantidade e o tipo (finito ou infinito).

A ação na linha seguinte é determinada por um sistema de regras que é guiada pela comparação dos pesos referentes a ação de construir, comprar ou recrutar. Caso a ação escolhida não seja concretizada, é retornada uma mensagem e caso outra ação esteja disponível, esta é escolhida.

Para o fim do turno analisado, as informações mostradas no início são repetidas para quesito de comparação de progresso.

```
01- Vez do jogadaor: dois
02- Comida: 4 Madeira: 4 Pedra: 4
03- Sem soldados na base
04- Soldados no heroi
05- 3 de classe: 1
06- 1 de classe: 2
07- Predios de nivel 1: all.
08- Recurso Critico: madeira
09- Andou de 2 2 para 4 7
10- --Caminho--
11- p(2,2) p(2,3) p(2,4) p(2,5) p(2,6) p(3,6) p(4,6) p(4,7)
12- Coletou 1 madeira, fonte infinita,
13- AÇÃO CONSTRUIR FOI ESCOLHIDA
14- Não construiu, falta recurso
15- Comida: 6 Madeira: 7 Pedra: 6
16- Sem soldados na base
17- Soldados no heroi
18- 3 de classe: 1
19- 1 de classe: 2
20- Predios de nivel 1: all
21- fim da vez do jogador dois
```

Figura 4 Turno 1, Jogador 2

```
01- Vez do jogador: um
02- Comida: 4 Madeira: 4 Pedra: 4
03- Sem soldados na base
04- Soldados no heroi
05- 3 de classe: 1
06- 1 de classe: 2
07- Predios de nivel 1: all.
08- Recurso Critico: pedra
09- Andou de 27 27 para 25 22
10- --Caminho--
11- p(27,27) p(27,26) p(27,25) p(27,24)
    p(26,24) p(26,23) p(25,23) p(25,22)
12- Comida: 6 Madeira: 6 Pedra: 6
13- Sem soldados na base
14- Soldados no heroi
15- 3 de classe: 1
16- 1 de classe: 2
17- Predios de nivel 1: all.
18- fim da vez do jogador um
```

Figura 5 Turno 1, Jogador 1

Nas Figuras 6 e 7 é apresentado o terceiro turno de cada jogador. Todas as informações são semelhantes a das figuras 4 e 5, exceto a partir da escolha.

Quando a ação é concretizada, aparece a informação do que foi feito (na figura mostra o código de qual prédio foi construído). É feita uma segunda tentativa pois a ação ainda não alcançou o limite da recursos a serem gastos na construção (limite que é a média dos recursos de todos os prédios do nível atual).

```
01- Vez do jogadaor: dois
02- Comida: 4 Madeira: 10 Pedra: 8
03- Sem soldados na base
04- Soldados no heroi
05- 3 de classe: 1
06- 1 de classe: 2
07- Predios de nivel 1: al1.
08- Predios de nivel 2: co1.
09- Recurso Critico:  comida
10- Andou de 2 6 para 5 3
11- --Caminho--
12- p(2,6) p(2,5) p(2,4) p(2,3)
    p(3,3)  p(4,3)
13- coletou 2 comida, fonte
14- finita,
15- AÇÃO CONSTRUIR FOI ESCOLHIDA
16- Construiu ar1------------
17- AÇÃO CONSTRUIR FOI ESCOLHIDA
18- Não construiu, falta recurso
19- Comida: 4 Madeira: 8 Pedra: 5
20- Sem soldados na base
21- Soldados no heroi
22- 3 de classe: 1
23- 1 de classe: 2
24- Predios de nivel 1: al1.
25- Predios de nivel 2: co1.
```

Figura 6 Turno 3, Jogador 2

```
01- Vez do jogadaor: dois
02- Comida: 4 Madeira: 10 Pedra: 8
03- Sem soldados na base
04- Soldados no heroi
05- 3 de classe: 1
06- 1 de classe: 2
07- Predios de nivel 1: al1.
08- Predios de nivel 2: co1.
09- Recurso Critico:  comida
10- Andou de 2 6 para 5 3
11- --Caminho--
12- p(2,6) p(2,5) p(2,4) p(2,3)
    p(3,3)  p(4,3)
13- coletou 2 comida, fonte
14- finita,
15- AÇÃO CONSTRUIR FOI ESCOLHIDA
16- Construiu ar1------------
17- AÇÃO CONSTRUIR FOI ESCOLHIDA
18- Não construiu, falta recurso
19- Comida: 4 Madeira: 8 Pedra: 5
20- Sem soldados na base
21- Soldados no heroi
22- 3 de classe: 1
23- 1 de classe: 2
24- Predios de nivel 1: al1.
25- Predios de nivel 2: co1.
```

Figura 7 Turno 3, Jogador 1

A Figura 8 apresenta inicialmente as informações do jogador, o recurso crítico e o caminho percorrido. A partir da linha dezenove, são testadas as ações escolhidas, quando uma ação se concretiza, as informações referentes a ação (neste caso a classe e a quantidade de soldados).

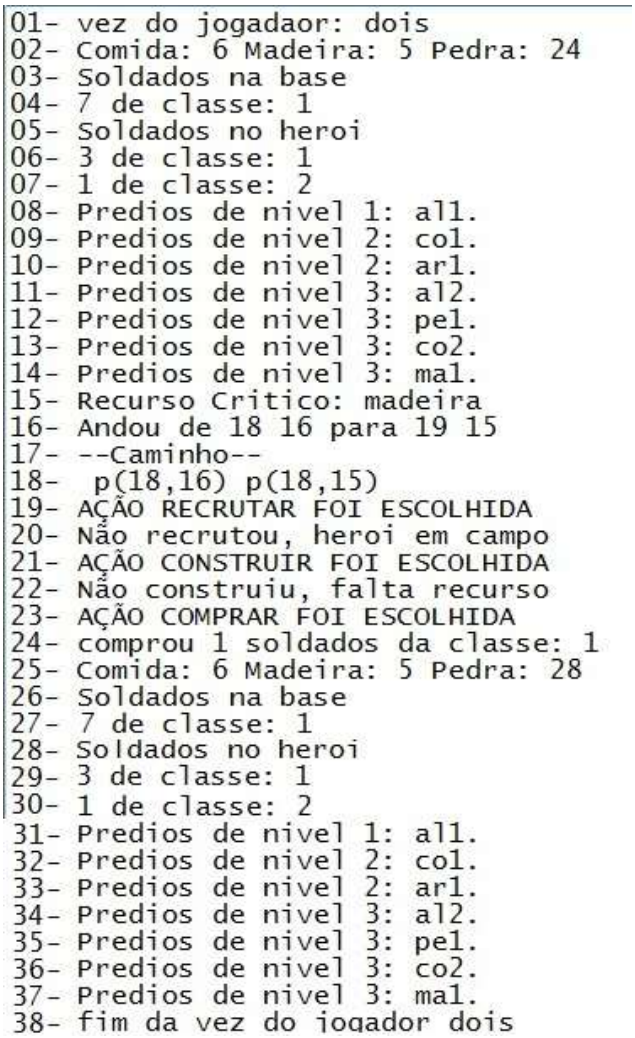


Figura 8 Turno 15, Jogador 2

A Figura 9 mostra as informações, localização, deslocamento e ações do jogador. Na linha dezoito é iniciada a batalha que acontece caso ambos os jogadores estejam no mesmo quadrado e é mostrada em detalhe na Figura 10.

```
01- Vez do jogador: um
02- Comida: 6 Madeira: 10 Pedra: 17
03- Soldados na base
04- 6 de classe: 1
05- Soldados no heroi
06- 3 de classe: 1
07- 1 de classe: 2
08- Predios de nivel 1: al1.
09- Predios de nivel 2: co1.
10- Predios de nivel 2: ar1.
11- Predios de nivel 3: al2.
12- Predios de nivel 3: pe1.
13- Predios de nivel 3: co2.
14- Predios de nivel 3: ma1.
15- andou de 20 11 para 19 15
16- --Caminho--
17- p(20,11) p(19,11) p(19,12)
    p(19,13)  p(19,14)
18- **Batalha**
19- AÇÃO RECRUTAR FOI ESCOLHIDA
20- Não recurtou, heroi em campo
21- AÇÃO COMPRAR FOI ESCOLHIDA
22- comprou 1 soldados da classe: 1
23- AÇÃO CONSTRUIR FOI ESCOLHIDA
24- Não construiu, falta recurso
25- Comida: 6 Madeira: 11 Pedra: 20
26- Soldados na base
27- 7 de classe: 1
28- Soldados no heroi
29- 3 de classe: 1
30- 1 de classe: 2
31- Predios de nivel 1: al1.
32- Predios de nivel 2: co1.
33- Predios de nivel 2: ar1.
34- Predios de nivel 3: al2.
35- Predios de nivel 3: pe1.
36- Predios de nivel 3: co2.
37- Predios de nivel 3: ma1.
38- fim da vez do jogador um
```

Figura 9 Turno 15, Jogador 1

Quando o método de disputar é chamado, são ordenados os soldados de cada time seguindo um valor aleatório mais um valor básico de cada tipo de unidade. A vez de um soldado consiste em escolher um alvo. Para isso, é feito um cálculo baseado na unidade atacante, a unidade defensora, e a chance de

acerto, e este cálculo é armazenado em um banco de dados. Quando o banco de dados possuir todas as configurações de unidades da batalha atual, a escolha é feita utilizando-se o Sistema Baseado em Casos no banco. Feita a escolha é calculado se o soldado acertou o alvo, e quanto de dano o defensor recebeu. Os valores de vida de cada grupo de soldados de classes iguais é mostrado por ordem de classe abaixo do time.

A Figura 10 apresenta o final da batalha cuja configuração inicial foi: 3 soldados de classe 1 (ou Unidade Básica) e 1 soldado de classe 2 (ou Unidade de Longo alcance.

No início do da batalha não existem casos armazenados, por se tratar da primeira batalha, portando a escolha do alvo é feita por um cálculo de alvo sofrerá o dano mais efetivo. Feito isso são armazenados os dados no banco de casos como mostra o exemplo:

Soldado da classe 2 do time A atacou soldado da classe 1 do time B:

- Chave gerada: 212
- Valor: porcentagem de dano causado

Soldado da classe 2 do time B atacou soldado da classe 2 do time A:

- Chave gerada: 221
- Valor: porcentagem de dano causado

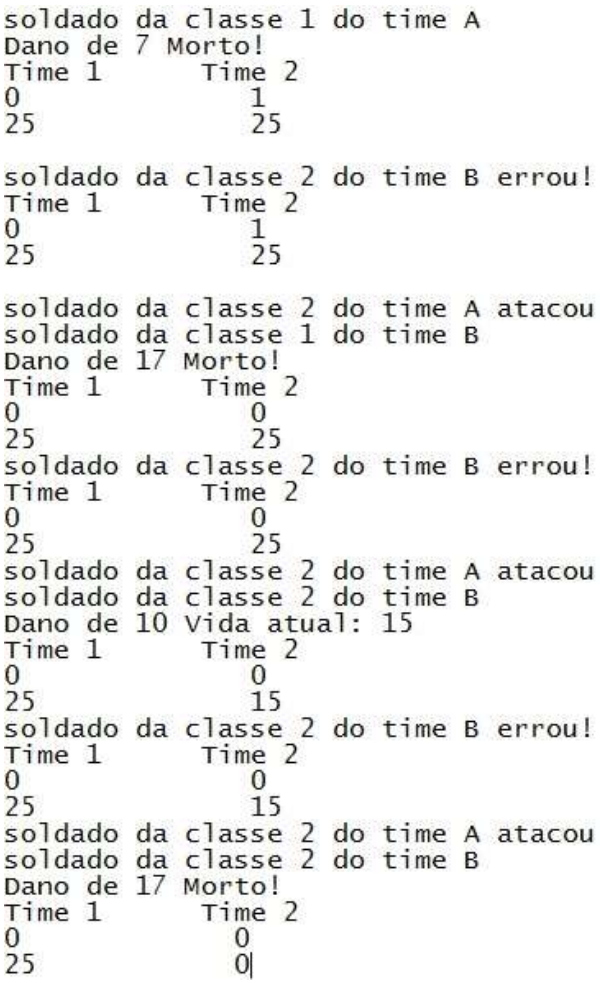


Figura 10 Batalha, rodadas finais

Os valores dos pesos usados para cada soldado e prédio foram estabelecidos de forma empírica, buscando o maior equilíbrio entre a relevância dos soldados e a relevância dos prédios para melhorar a decisão entre as ações.

4.2Discussão

A primeira dificuldade encontrada foi na busca por técnicas já aplicadas e/ou adaptadas para um jogo eletrônico. A elaboração de uma*Game-IA* não possui, em muitos casos, uma metodologia formal, de acordo com Kishimoto (2004). Outra dificuldade em se achar uma técnica focada e adaptada para um jogo é a grande variedade de jogos e necessidades de tais jogos resultando numa interpretação ampla do que é IA (KISHIMOTO, 2004).

Durante o desenvolvimento, a dificuldade encontrada foi no planejamento da classe responsável pelo combate devido ao grande número de métodos auxiliares para que a ação principal fosse executada.

Ainda durante a implementação, houve grande dificuldade para e elaboração das funções responsáveis pela construção e compra de soldados por razão do planejamento das funções de suporte.

Durante a fase de testes, estabelecer um conjunto de pesos para prédios e unidades, visando o equilíbrio e melhores escolhas, demando grande tempo e tentativas para chegar aos pesos ideias.

Após vários testes com diferentes pesos, para prédios e soldados, para obter valores que gerassem ações que refletiam competitividade e ações inteligentes, foi possível obter uma solução que demonstrasse escolhas adequadas por parte dos jogadores e equilíbrios entre tais escolhas.

Não houve sincronização com o modulo gráfico desenvolvido por Marlon(2013) devido a diferenças de estrutura e módulos de acoplagem e nem há interação com o usuário devido à falta de um modulo gráfico.

Ocorreu a sincronização com o modulo de criação de mapa utilizando a classe CarregarMapa.java apresentada no Apendice B.

5 Conclusão

Nesse projeto foi desenvolvido um módulo de inteligência artificial para um jogo de estratégia baseado em turnos para Android.Nesse módulo foram implementados três técnicas para que a IA fosse efetiva no estilo de jogo proposto: Algoritmo de Busca, utilizando o A*, para deslocamento no cenário; Sistema Baseado em Regras, para a escolha das ações a serem tomadas pelo jogador artificial e; Sistema Baseado em Casos, para auxiliar e guiar as escolhas durante a Batalha

Conforme observação e análise das imagens dos turnos selecionados, das repostas obtidas das três técnicas aplicadas, conclui-se que as técnicas possuem eficácia satisfatória e resposta em tempo praticável, seguindo as técnicas apresentadas por Kishimoto (2004) e Paula (2007).

Trabalhos futuros podem ser focados em melhorar os algoritmos usados para a execução das escolhas, como a inclusão de mais regras, maior adaptação ao ambiente Android e possibilitar uma interação com o usuário.

6 Referências Bibliográficas

BORGES, D. M., BARREIRA, R. G., SOUZA, J. G.**Comportamento de personagens em jogos de computador**. Palmas: Centro Universitário Luterano de Palmas, 2009. p. 113-120

BOURG, D. M.; SEEMAN, G. **AI for GameDevelopers.** Sebastopol: O'Reilly. 2004.

DALMAU, D. S.C. **Core Techniques and Algorithms in Game Programming**. Indianapolis: New Riders. 2004.

DIMARZIO, J F. **Android: A programmer's Guide**. New York. McGrawHill E-Book, 16, 2008.

FULLERTON, T. **GAME DESIGN WORKSHOP, a Playcentric Approach toCreatingInnovative Games.** 2. Ed. USA: ElsevierInc, Morgan Kaufmannpublications, 2008. 491 p.

G1. **No mundo, 4 a cada 5 smartphones vendidos rodam o sitemaAndroid:** G1, Globo.com, 2013. Disponível em:<Http://http://g1.globo.com/tecnologia/noticia/2013/10/no-mundo-4-cada-5-smartphones-vendidos-rodam-o-sistema-android.html>. Acesso em 30 de Maio 2013

JÚNIOR, M. A. P.; CASTRO, P. O.**Um estudo de caso da plataforma Android com Interfaces Adaptativas**, São Carlos, 2011.

KARLSSON, B. F. F. **Um Middleware de Inteligência Artificial para Jogos Digitais. Dissertação de Mestrado**, 2005. Rio de Janeiro. 126 p.

KISHIMOTO, A. **Inteligência Artificial em Jogos Eletrônicos.** 2004, p 11

LEE, W. M. **Beginning Android™ Application Development**. Indianapolis: Wiley Publishing, Inc, 2011. 450 p.

LIMA, M. J. O. **Pesquisa e Desenvolvimento da Interface de um Jogo para a Plataforma Android**, 2013. 51 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Universidade Federal de Lavras, Lavras, 2013.

LUGER, G. F. **Inteligência Artificial: Estruturas e Estratégias para a Solução de Problemas Complexos**. 4ª ed. Porto Alegre: Bookman, 2004. 774 p.

PAULA, B. C. **Sistema Inteligente Para Jogos De Estratégia Baseados Em Turnos: Uma Abordagem Utilizando Planejamento Baseado Em Casos**. Curitiba, 2007.

POZZER, C. T. **Game IA**. Santa Maria 2006

RODRIGUES, J. A. B; MARCHI, K, R, C; DIAS, J. W. Paradigmas da Inteligência Artificial em Jogos Eletrônicos. **Universidade Paranaense (Unipar)**. Paranavaí, PR, Brasil, 2013. Disponível em: <http://web.unipar.br/~seinpar/2013/artigos/Joao%20Antonio%20Bezerra%20Rodrigues.pdf >. Acesso em: 20 de Abril de 2014.

RIESBECK, C. K.; SCHANK, R. C. **Inside Case-BasedReasoning**. Lawrence Erlbaum, 1989.

ROUSSE, R. **Game Design:** Theory & Practice. 2nd. Ed. Wordware Publishing, Inc.; 2001. 584p

SCOTA, D. F.; ANDRADE, G. E.; XAVIER, R. C.**Configuração de Rede sem Fio e Segurança no Sistema Operacional Android**, Curitiba, 2013.

SCHWAB, B. **IA Game Engine Programming.** Hingham: Charles River Media. 2004.

TOZOUR, P. **The Evolution of Game AI from AI GameProgramming Wisdom**. Hingham: Charles River Media. 2002.

ZECHNER, M. **Beginning Android Games**. 2ª. ed. [S.l.]: Apress, 2011. 688 p.

7 Apêndice A

Implementação do Sistema de Regras

Este é a implementação do sistema de regras e foi dividida em quatro classes: Regras.java, Predio.java, ListaV.java e Vila.java. Essas classes são referentes ao pacote br.ufla.americatribal.vila

1. Regras.Java

Nesta classe estão contidas todas as regras do sistema de regras, bem como a funções principais para a realização das ações

```
package br.ufla.americatribal.vila;

import br.ufla.americatribal.geral.Jogador;
import br.ufla.americatribal.batalha.Combatente;
import br.ufla.americatribal.batalha.Guerra;
import br.ufla.americatribal.campo.Lista;
import br.ufla.americatribal.campo.Mapa;
import br.ufla.americatribal.campo.Ponto;

public class Regras {

	public enum Dificuldade {
		FACIL, MEDIO, DIFICIL
	}

	public Dificuldade nivel;

	public Guerra g = new Guerra();
	public Lista caminho = new Lista();

	public int passos;

	public boolean FIMDEJOGO = false;
	public String VENCEDOR;
```

```
	enum PrediosMilitares {
		ar1, ar2, es1, tem, to3

	}

	PrediosMilitares militares;
	public boolean comprarSoldados = false;
	public boolean recrutarSoldados = false;

	public boolean modificarPeso = true;

	public boolean investindo;

	public Regras() {
	}

	public void Mostrar(Object o) {
		System.out.println(o);
	}

	public void Escolher(Jogador j, Jogador j2, Mapa m) {
		g = new Guerra(j.soldados, j2.soldados);
		passos = 7;
		g.combatentesVila(j.v, j);
		System.out.println("nivel tec: " + j.nTec);
		j.possiveis = j.restantes.separaPorNivel(j.nTec);
		if (j.possiveis.lista.size() != 0) {
			this.recursoLimite(j);
		}
		if (j.restantes.lista.size() != 0) {
			if (j.possiveis.lista.size() == 0) {
				j.nTec++;
				j.possiveis =
j.restantes.separaPorNivel(j.nTec);
				this.recursoLimite(j);
			}
		}
		System.out.println("nivel tec: " + j.nTec);
		j.opcoes = RecursoCritico(j, m);
		if (j.precisaRecurso != 0 && !investindo) {
			pegarRecurso(j, j2, m);
			// return;
		}
		// VERIFICA SE O RECURSO NÃO ESTA NO VERMELHO (2)
		if (j.precisaRecurso < 2) {
```

```
                    // AJUSTA OS PESOS CONFORME A SITUAÇÃO
                    if (Dificuldade.FACIL == nivel) {
                          if (this.proximidade(j.origem,
j2.origem, m) < 2) {
                                Mostrar("perto");
                                if (j.PesoSoldadoHeroi >= 17) {
                                      j.destino = new
Ponto(j2.origem);
                                      this.Andar(j, m);
                                      investindo = true;
                                      Mostrar("investindo");
                                      j.limitador += 4;
                                      if
(j.origem.igual(j2.origem)) {

      this.g.disputa(j.soldados, j2.soldados);

      this.ajustarPesos(j);

      this.ajustarPesos(j2);
                                            investindo = false;
                                      }
                                      if (j.PesoSoldadoHeroi >
17) {
                                            j.limitador += 4;
                                      }
                                }
                          }
                          if (this.proximidade(j.origem, j.casa,
m) < 1) {
                                if (j.PesoSoldadoHeroi >= 32) {

                                }
                          }
                    }

                    // VERIFICA SE SERÁ POSSIVEL COMPRAR SOLDADOS
                    for (PrediosMilitares pm :
PrediosMilitares.values()) {
                          if
(j.possuidos.encontraId(pm.toString())) {
                                comprarSoldados = true;
                                break;
                          }
                    }
```

```
                  if (comprarSoldados) {
                        // CASO SEJA POSSIVEL COMPRA SOLDADOS,
VERIFICA SE PODE RECRUTAR
                        for (int i = 0; i < 5; i++) {
                              if (j.v.base[i] > 0) {
                                    recrutarSoldados = true;
                                    break;
                              }
                        }
                  }
                  boolean limiteConstruir = true, limiteComprar
= true;
                  j.limitador = 0;
                  while ((limiteConstruir || limiteComprar) &&
(j.limitador < 7)) {
                        Mostrar("pode construir " +
limiteConstruir);
                        Mostrar("pode comprar " +
limiteComprar);
                        Mostrar("limitador " + (j.limitador));
                        Mostrar("comprar Soldado " +
this.comprarSoldados);
                        Mostrar("recrutar Soldado " +
this.recrutarSoldados);
                        if (j.recursoConstrucao <= 0) {
                              limiteConstruir = false;
                        }
                        if ((j.v.recursos[0] <
j.recursoMilitar)
                                    || (j.v.recursos[1] <
j.recursoMilitar)
                                    || (j.v.recursos[2] <
j.recursoMilitar)) {
                              limiteComprar = false;
                        }
                        Mostrar("escolher acao");
                        this.escolherAcao(j, j2, m,
comprarSoldados, recrutarSoldados);
                  }
                  Mostrar("Fim da escolha da ação");

            }
            modificarPeso = true;
      }
```

```
	public void escolherAcao(Jogador j, Jogador j2, Mapa m,
			boolean comprarSoldados, boolean
recrutarSoldados) {
		if (comprarSoldados) {
			if (recrutarSoldados) {
				switch (j.limitador) {
				case 0:
					if (j.PesoPredio <
j.PesoSoldadoBase) {
						if (j.PesoPredio <
j.PesoSoldadoHeroi) {
							Mostrar("AÇÃO
CONSTRUIR FOI ESCOLHIDA");
							if
(!this.acaoConstruir(j, j2, m))
								break;
						} else {
							Mostrar("AÇÃO
RECRUTAR FOI ESCOLHIDA");
							if
(!this.acaoRecrutar(j, m))
								break;
						}
					} else {
						if (j.PesoSoldadoBase <
j.PesoSoldadoHeroi) {
							Mostrar("AÇÃO
COMPRAR FOI ESCOLHIDA");
							if
(!this.acaoComprar(j))
								break;
						} else {
							Mostrar("AÇÃO
RECRUTAR FOI ESCOLHIDA");
							if
(!this.acaoRecrutar(j, m))
								break;
						}
					}
				case 1:
					if (j.PesoSoldadoBase <
j.PesoSoldadoHeroi) {
						Mostrar("AÇÃO COMPRAR FOI
ESCOLHIDA");
						if (!this.acaoComprar(j))
```

```
                                        break;
                                } else {
                                        Mostrar("AÇÃO RECRUTAR FOI
ESCOLHIDA");
                                        if (!this.acaoRecrutar(j,
m))
                                                break;
                                }
                        case 2:
                                if (j.PesoPredio <
j.PesoSoldadoHeroi) {
                                        Mostrar("AÇÃO CONSTRUIR
FOI ESCOLHIDA");
                                        if (!this.acaoConstruir(j,
j2, m))
                                                break;
                                } else {
                                        Mostrar("AÇÃO RECRUTAR FOI
ESCOLHIDA");
                                        if (!this.acaoRecrutar(j,
m))
                                                break;
                                }
                        case 3:
                                Mostrar("AÇÃO RECRUTAR FOI
ESCOLHIDA");
                                if (!this.acaoRecrutar(j, m))
                                        break;
                        case 4:
                                if (j.PesoPredio <
j.PesoSoldadoBase) {
                                        Mostrar("AÇÃO CONSTRUIR
FOI ESCOLHIDA");
                                        if (!this.acaoConstruir(j,
j2, m))
                                                break;
                                } else {
                                        Mostrar("AÇÃO COMPRAR FOI
ESCOLHIDA");
                                        if (!this.acaoComprar(j))
                                                break;
                                }
                        case 5:
                                Mostrar("AÇÃO COMPRAR FOI
ESCOLHIDA");
```

```
                                    if (!this.acaoComprar(j))
                                          break;
                              case 6:
                                    Mostrar("AÇÃO CONSTRUIR FOI
ESCOLHIDA");
                                    if (!this.acaoConstruir(j, j2,
m))
                                          break;
                              }
                        } else {
                              if (j.limitador == 1) {
                                    Mostrar("AÇÃO COMPRAR FOI
ESCOLHIDA");
                                    this.acaoComprar(j);
                              } else if (j.limitador == 2) {
                                    Mostrar("AÇÃO CONSTRUIR FOI
ESCOLHIDA");
                                    this.acaoConstruir(j, j2, m);
                                    ;
                              } else if (j.limitador == 0) {
                                    if (j.PesoPredio <
j.PesoSoldadoBase) {
                                          Mostrar("AÇÃO CONSTRUIR
FOI ESCOLHIDA");
                                          this.acaoConstruir(j, j2,
m);
                                    } else {
                                          Mostrar("AÇÃO COMPRAR FOI
ESCOLHIDA");
                                          this.acaoComprar(j);
                                    }
                              } else {
                                    j.limitador = 7;
                              }
                        }
                  } else {
                        Mostrar("AÇÃO CONSTRUIR FOI ESCOLHIDA");
                        if (!this.acaoConstruir(j, j2, m)) {
                              j.limitador = 7;
                        }
                  }
            }

            // verifica se algum recurso esta em grande falta
            public Lista RecursoCritico(Jogador j, Mapa m) {
```

```
                int aux = 0;
                for (int i = 0; i < 3; i++) {
                        if (j.v.recursos[i] < 30) {
                                if (j.v.recursos[i] > 5)
                                        aux = 1;
                                else
                                        aux = 2;
                        }
                }
                if (aux > 0) {
                        int critico = j.v.escolheRecurso();
                        switch (critico) {
                        case 0:
                                if (!(TesteEConstrucao("co", j,
j.possiveis))) {
                                        System.out.print("Recurso
Critico: ");
                                        j.rc = 2;
                                        j.precisaRecurso = aux;
                                        System.out.print(" comida \n");
                                        return m.comidas;
                                }
                                break;
                        case 1:
                                if (!(TesteEConstrucao("ma", j,
j.possiveis))) {
                                        System.out.print("Recurso
Critico: ");
                                        j.rc = 3;
                                        j.precisaRecurso = aux;
                                        System.out.print("madeira \n");
                                        return m.madeiras;
                                }
                                break;
                        case 2:
                                if (!(TesteEConstrucao("pe", j,
j.possiveis))) {
                                        System.out.print("Recurso
Critico: ");
                                        j.rc = 4;
                                        j.precisaRecurso = aux;
                                        System.out.print("pedra \n");
                                        return m.pedras;
                                }
                                break;
```

```
                }
                return this.RecursoCritico(j, m);
            }
            j.precisaRecurso = aux;
            return null;
      }

      // verifica se uma construção pode ser feita e caso
positivo construi esta
      // usado para co,de,ma,pe
      public boolean TesteEConstrucao(String id, Jogador j,
ListaV possiveis) {
            if (j.possuidos.encontraId(id + "1")) {
                  if (j.possuidos.encontraId(id + "2")) {
                        if (!(j.possuidos.encontraId(id +
"3"))) {
                              if
(j.v.liberarConstrucao(possiveis.encontraPredio(id + 3),
                                          j)) {
                                    j.v.construir(id + "3",
j.possuidos, j.restantes);
                                    j.PesoPredio +=
possiveis.encontraPredio(id + 3).peso;
                                    j.alterarCarencia(id);
                                    j.precisaRecurso = 0;
                                    System.out

      .println("Construção de emergincia \n Recurso "

      + id);
                                    return true;
                              }
                        }
                  } else {
                        if (possiveis.encontraPredio(id + 2) !=
null
                                    && j.v.liberarConstrucao(

      possiveis.encontraPredio(id + 2), j)) {
                              j.v.construir(id + "2",
j.possuidos, j.restantes);
                              j.alterarCarencia(id);
                              j.precisaRecurso = 0;
                              System.out.println("Construção
de emergincia \n Recurso "
```

```
                                                + id);
                                    return true;
                              }

                        }
                  } else {
                        if (possiveis.encontraPredio(id + 1) != null
                                    &&
j.v.liberarConstrucao(possiveis.encontraPredio(id + 1),
                                                j)) {
                              j.v.construir(id + "1", j.possuidos,
j.restantes);
                              j.alterarCarencia(id);
                              j.precisaRecurso = 0;
                              System.out.println("Construção de
emergincia \n Recurso " + id);
                              return true;
                        }
                  }
                  return false;
            }

            public void pegarRecurso(Jogador j, Jogador j2, Mapa m) {
                  Andar(j, m);
                  if ((this.proximidade(j.origem, j2.origem, m) < 3)
&& modificarPeso) {
                        j.PesoSoldadoBase *= 0.75;
                        modificarPeso = false;
                  }
                  Ponto aux = new Ponto();
                  if (m.comidas.contem(j.origem)) {
                        j.precisaRecurso = 0;
                        aux = m.comidas.pegar(j.origem);

                  } else if (m.madeiras.contem(j.origem)) {
                        j.precisaRecurso = 0;
                        aux = m.madeiras.pegar(j.origem);

                  } else if (m.pedras.contem(j.origem)) {
                        j.precisaRecurso = 0;
                        aux = m.pedras.pegar(j.origem);
                  }
                  j.origem.tipo = aux.tipo;
                  j.origem.qtde = aux.qtde;
                  coletar(j, m);
```

```
	}

	public void coletar(Jogador j, Mapa m) {
		int recurso = m.mapa[j.origem.x][j.origem.y];
		if (j.origem.tipo != null) {
			if (j.origem.tipo.equals("UNICO")) {
				if (recurso == Mapa.COMIDA || recurso
== Mapa.COMIDA_COL) {
					m.comidas.remover(j.origem.x,
j.origem.y);
					j.v.recursos[0] +=
j.origem.qtde;
					if (j.v.carece[0] > 8) {
						j.v.carece[0] -= 8;
					} else {
						j.v.carece[0] = 0;
					}
					System.out.println("coletou
comida, fonte finita, \nqtde"
							+ j.origem.qtde +
"------------");
				} else if (recurso == Mapa.MADEIRA
						|| recurso ==
Mapa.MADEIRA_COL) {
					m.madeiras.remover(j.origem.x,
j.origem.y);
					j.v.recursos[1] +=
j.origem.qtde;
					if (j.v.carece[1] > 8) {
						j.v.carece[1] -= 8;
					} else {
						j.v.carece[1] = 0;
					}
					System.out.println("coletou
madeira, fonte finita, \nqtde"
							+ j.origem.qtde +
"------------");
				} else if (recurso == Mapa.PEDRA ||
recurso == Mapa.PEDRA_COL) {
					m.pedras.remover(j.origem.x,
j.origem.y);
					j.v.recursos[2] +=
j.origem.qtde;
					if (j.v.carece[2] > 8) {
						j.v.carece[2] -= 8;
```

```
                                    } else {
                                          j.v.carece[2] = 0;
                                    }
                                    System.out.println("coletou
madeira, fonte finita, \nqtde"
                                                + j.origem.qtde +
"------------");
                              }
                        } else {
                              if (!(j.origem.tipo.equals(j.id))) {
                                    if (recurso == Mapa.COMIDA) {
                                          int aux =
m.comidas.posicao(j.origem.x, j.origem.y);

      m.comidas.lista.get(aux).tipo = j.id;
                                          j.v.recursoBonus[0] +=
j.origem.qtde;
                                          if (j.v.carece[0] > 8) {
                                                j.v.carece[0] -= 8;
                                          } else {
                                                j.v.carece[0] = 0;
                                          }

      System.out.println("coletou comida, fonte oo, \nqtde"
                                                      +
j.origem.qtde + "------------");
                                    } else if (recurso ==
Mapa.MADEIRA) {
                                          int aux =
m.madeiras.posicao(j.origem.x, j.origem.y);

      m.madeiras.lista.get(aux).tipo = j.id;
                                          j.v.recursoBonus[1] +=
j.origem.qtde;
                                          if (j.v.carece[1] > 8) {
                                                j.v.carece[1] -= 8;
                                          } else {
                                                j.v.carece[1] = 0;
                                          }

      System.out.println("coletou madeira, fonte oo, \nqtde "
                                                      +
j.origem.qtde + "------------");
                                    } else {
```

```
                                        int aux =
m.pedras.posicao(j.origem.x, j.origem.y);

      m.pedras.lista.get(aux).tipo = j.id;
                                        j.v.recursoBonus[2] +=
j.origem.qtde;
                                        if (j.v.carece[2] > 8) {
                                              j.v.carece[2] -= 8;
                                        } else {
                                              j.v.carece[2] = 0;
                                        }

      System.out.println("coletou pedra, fonte oo, \nqtde"
                                                    +
j.origem.qtde + "------------");
                                  }
                            }
                      }
                }

      }

      public boolean acaoComprar(Jogador j) {
            if (!this.ComprarSoldado(j)) {
                  j.limitador += 2;
                  return false;
            }
            return true;
      }

      // escolhe o tipo de unidade que esta em maior falta e
compra o valor para
      // suprir
      public boolean ComprarSoldado(Jogador j) {
            Mostrar("vai tentar comprar");
            int contador = 0, qtde = 0, posicao = 0;
            boolean escolhido = true;
            for (PrediosMilitares pm :
PrediosMilitares.values()) {
                  if (j.possuidos.encontraId(pm.toString())) {
                        contador++;
                  }
            }
            for (int i = 0; i < contador; i++) {
                  if (j.v.base[0] == 0) {
```

```
                        qtde = 5 - i;
                        escolhido = true;
                        if (nivel != Dificuldade.DIFICIL) {
                              break;
                        }
                  }
            }
            if (!escolhido) {
                  int[] nPelotao = { 0, 0, 0, 0, 0 };
                  float[] porcentagem = { 0, 0, 0, 0, 0 };
                  for (int i = 0; i < 5; i++) {
                        int inc = 0;
                        int aux = (5 - i) * inc;
                        if (j.v.base[i] == 0) {
                              continue;
                        }
                        while (j.v.base[i] > aux) {
                              inc++;
                              aux = (5 - i) * inc;
                        }
                        nPelotao[i] = inc;
                        porcentagem[i] = (aux - j.v.base[i]) /
aux;
                        if (j.v.base[i] == aux) {
                              nPelotao[i]++;
                              aux = (5 - i) * inc;
                              porcentagem[i] = (aux -
j.v.base[i]) / aux;
                              porcentagem[i] *= .5;
                        }
                  }
                  if (((j.v.recursos[0] > j.recursoMilitar)
                              && (j.v.recursos[1] >
j.recursoMilitar) && j.v.recursos[2] > j.recursoMilitar)) {
                        float porcentagemEscolhida =
porcentagem[0];
                        posicao = 0;
                        for (int i = 1; i < 5; i++) {
                              if (porcentagemEscolhida <
porcentagem[i]) {
                                    porcentagemEscolhida =
porcentagem[i];
                                    porcentagem[i] = 0;
                                    posicao = i;
                              }
```

```
                        }
                        qtde = nPelotao[posicao] * (5 -
posicao) - j.v.base[posicao];
                  }
            }
            if (j.v.comprar((posicao + 1), qtde, j)) {
                  switch (posicao + 1) {
                  case 1:
                        j.PesoSoldadoBase += 1 * qtde;
                        break;
                  case 2:
                        j.PesoSoldadoBase += 2 * qtde;
                        break;
                  case 3:
                        j.PesoSoldadoBase += 3 * qtde;
                        break;
                  case 4:
                        j.PesoSoldadoBase += 4 * qtde;
                        break;
                  case 5:
                        j.PesoSoldadoBase += 5 * qtde;
                        break;
                  }
                  return true;
            }

            return false;
      }

      public boolean acaoRecrutar(Jogador j, Mapa m) {
            if (j.origem.igual(j.casa)) {
                  this.RecrutarSoldados(j);
                  return true;
            } else {
                  j.destino = new Ponto(j.casa);
                  this.Andar(j, m);
                  if (j.origem.igual(j.casa)) {
                        this.RecrutarSoldados(j);
                        return true;
                  } else {
                        j.limitador += 4;
                        return false;
                  }
            }
      }
```

```
	// compra a unidade que esta mais em falta
	public void RecrutarSoldados(Jogador j) {
		int[] relatorio = { -1, -1, -1 };
		int classe = 0, qtde = 0;
		for (int i = 0; i < 3; i++) {
			if (j.soldados != null) {
				if (j.soldados[i] != null) {
					if (j.soldados[i].qtde != 0) {
						relatorio[i] =
j.soldados[i].p.classe;
					}
				}
			}
		}
		int pos = -1;
		for (int i = 2; i >= 0; i--) {
			if (relatorio[i] == -1) {
				pos = i;
			}
		}
		if (pos > -1) {
			classe = this.auxRecrutar(j, relatorio);
			qtde = j.v.fatorDeSoldadosHeroi * (6 -
classe);
			if (qtde >= j.v.base[classe - 1]) {
				j.soldados[pos] = new
Combatente(classe, j.v.base[classe - 1]);
				j.v.base[classe - 1] = 0;
			} else {
				j.soldados[pos] = new
Combatente(classe, qtde);
				j.v.base[classe - 1] -= qtde;
			}
		} else {
			classe = j.soldados[0].p.classe;
			float porcentagem = (float)
j.soldados[0].qtde
					/ (j.v.fatorDeSoldadosHeroi * (6
- classe));
			int posicao = 0;
			for (int t = 1; t < 3; t++) {
				int classeAux = j.soldados[t].p.classe;
				float aux = (float) j.soldados[t].qtde
```

```
                                        /
(j.v.fatorDeSoldadosHeroi * (6 - classeAux));
                        if (((porcentagem < aux) ||
(porcentagem == 1.0))) {
                                classe = classeAux;

                                porcentagem = aux;
                                posicao = t;
                        }
                }
                qtde = j.v.fatorDeSoldadosVila * (6 - classe)
                                - j.soldados[posicao].qtde;
                if (qtde >= j.v.base[classe - 1]) {
                        j.soldados[posicao].qtde +=
j.v.base[classe - 1];
                        j.v.base[classe - 1] = 0;
                } else {
                        j.soldados[posicao].qtde += qtde;
                        j.v.base[classe - 1] -= qtde;
                }
                switch (classe) {
                case 1:
                        j.PesoSoldadoHeroi += 6;
                        j.PesoSoldadoBase -= 6;
                        break;
                case 2:
                        j.PesoSoldadoHeroi += 8;
                        j.PesoSoldadoBase -= 8;
                        break;
                case 3:
                        j.PesoSoldadoHeroi += 10;
                        j.PesoSoldadoBase -= 10;
                        break;
                case 4:
                        j.PesoSoldadoHeroi += 12;
                        j.PesoSoldadoBase -= 12;
                        break;
                case 5:
                        j.PesoSoldadoHeroi += 18;
                        j.PesoSoldadoBase -= 18;
                        break;
                }
                pos = posicao;
        }
```

```
            Mostrar("Recrutou " + qtde + " da classe " + classe
+ " na posicao "
                        + pos);

      }

      public int auxRecrutar(Jogador j, int[] exc) {
            int ctrl = 0;
            if (Dificuldade.FACIL == nivel || nivel ==
Dificuldade.MEDIO) {
                  // recurtar a classe mais fraca
                  while ((j.v.base[ctrl] == 0) || (ctrl + 1 ==
exc[0])
                              || (ctrl + 1 == exc[1] || ctrl +
1 == exc[2])) {
                        ctrl++;

                  }
            }

            return ctrl + 1;
      }

      public void Andar(Jogador j, Mapa m) {

            if (passos != 0) {
                  if (j.opcoes.lista.size() == 1) {
                        j.destino = j.opcoes.lista.get(0);
                  } else {
                        j.destino = m.destinoOp(j);
                  }
                  System.out.print("andou de " + j.origem.x + "
" + j.origem.y);
                  caminho = Lista.aEstrela(m, j);
                  Ponto aux = new Ponto();

                  // só pra rodar mais, pq não anda se entrar
aqui
                  if (caminho == null || caminho.lista == null)
{
                        System.out.println(" para " +
j.origem.x + " " + j.origem.y);
                        System.out.println("\n--\nCAMINHO
NULO!\n--");
                        return;
```

```
                }

                if (passos >= caminho.lista.size() - 1) {
                        aux =
caminho.lista.get(caminho.lista.size() - 1);
                        aux.pai = null;
                        j.origem = aux;
                        caminho.imprimir(passos);
                        passos -= caminho.lista.size();
                } else {
                        aux = caminho.pegar(passos);
                        if (aux != null) {
                                aux.pai = null;
                                j.origem = aux;
                                caminho.imprimir(passos);
                                passos = 0;
                        } else {
                                System.out
                                                .println("\n--
\nCaminho não possui passos suficientes\n--");
                        }
                }
                System.out.println(" para " + j.origem.x + "
" + j.origem.y);
        }
    }

    public boolean acaoConstruir(Jogador j, Jogador j2, Mapa
m) {
        this.Atratividade(j, j2, false, m);
        if (!this.QualConstruir(j.possiveis, j.nTec, j)) {
                j.limitador += 1;
                return false;
        }
        j.possiveis = j.restantes.separaPorNivel(j.nTec);
        if (j.possiveis.lista.size() != 0) {
                this.recursoLimite(j);
        }

        return true;
    }

    // Funcoes para decidir qual predio sera construido
    public boolean QualConstruir(ListaV possiveisNivel, int
nivelTec, Jogador j) {// testado!
```

```
            int maior = 0, peso = 0;
            String id = "vazio";
            for (Predio p : possiveisNivel.lista) {
                  if (j.v.liberarConstrucao(p, j)) {
                        if (p.atratividade > maior) {
                              maior = p.atratividade;
                              peso = p.peso;
                              id = p.id;
                        }
                  }
            }
            if (!id.equals("vazio")) {
                  j.v.construir(id, j.possuidos, j.restantes);
                  j.recursoConstrucao -=
possiveisNivel.encontraPredio(id).maior;
                  j.PesoPredio += peso;
                  return true;
            }
            return false;
      }

      // verifica se e quantos soldados falto para o minimo
      public boolean PrecisaSoldadosVila(Jogador j) {// testado
            boolean aux = false;
            int i = 0;
            for (PrediosMilitares pm :
PrediosMilitares.values()) {
                  if (j.possuidos.encontraId(pm.toString())
                              && (j.v.fatorDeSoldadosVila * (5
- i)) > 0) {
                        if (j.v.base[i] <
j.v.fatorDeSoldadosVila * (5 - i)) {
                              aux = true;
                        }
                  }
                  i++;
            }
            if (aux) {
                  System.out.println("precisa de soldados");
                  return true;
            }
            return false;

      }
```

```
	// verifica se e quantos soldados precisam para o minimo
no heroi
	public boolean PrecisaSoldadosHeroi(Jogador j) {
		if (j.soldados != null) {
			int id, falta;
			for (int i = 0; i < 3; i++) {
				if (j.soldados[i] != null) {
					id = j.soldados[i].p.classe;
					falta = j.soldados[i].qtde -
j.v.fatorDeSoldadosHeroi
							* (5 - id + 1);
					if (falta < 0) {
						System.out.println("HEROI
PRECISA DE SOLDADOS");
						return true;
					}
				} else {
					for (i = 0; i < 5; i++) {
						if ((j.v.base[0] > 0)
								&&
(j.v.fatorDeSoldadosHeroi * (5 - i + 1) >= 1)) {

	System.out.println("HEROI PRECISA DE SOLDADOS");
							return true;
						}
					}
				}

			}

		}
		return false;
	}

	public boolean podeAtacar(Jogador j) {
		if (j.soldados != null)
			for (int i = 0; i < 3; i++) {
				if (j.soldados[i] != null) {
					if (j.soldados[i].qtde <= 0)
						return false;
				} else {
					return false;
				}

			}
```

```
            return true;

      }

      public void Atratividade(Jogador j, Jogador j2, boolean
inicio, Mapa m) {
            for (Predio p : j.possiveis.lista) {
                  p.atratividade = p.atratividadeBase;
                  if (p.classe.equals("base")) {
                        // AJUSTE DE PERSONALIDADE
                        if ((this.nivel == Dificuldade.FACIL)
&& (inicio))
                              p.atratividade += 3;
                        // AJUSTE INTEGRAL DE PROXIMIDADE DO
INIMIGO
                        if (this.proximidade(j.casa, j2.origem,
m) == 3)
                              // FEITO EM PREDIOS DE DEFESA
                              if (p.classe.equals("de1") ||
p.classe.equals("de2")
                                          ||
p.classe.equals("to1") || p.classe.equals("de3")
                                          ||
p.classe.equals("to2") || p.classe.equals("to3"))
                                    p.atratividade += 4;
                  }
                  if (p.classe.equals("recursos")) {
                        // AJUSTE DE PERSONALIDADE
                        if ((this.nivel == Dificuldade.MEDIO)
&& (inicio))
                              p.atratividade += 2;
                        // AJUSTE PARCIAL QUANDO RECURSO EM
NIVEL AMARELO
                        if (j.precisaRecurso == 1)
                              p.atratividade += 2;
                  }
                  if (p.classe.equals("militar")) {
                        // AJUSTE DE PERSONALIDADE
                        if ((this.nivel == Dificuldade.DIFICIL)
&& (inicio))
                              p.atratividade += 2;
                        // AJUSTE INTEGRAL DE PROXIMIDADE DO
INIMIGO
                        if (this.proximidade(j.casa, j2.origem,
m) == 2)
```

```
                                    p.atratividade += 4;

                        }

                }

        }

        // AJUSTAR OS PESOS PARA SITUAÇÕES ATIPICAS

        public int proximidade(Ponto p1, Ponto p2, Mapa m) {
                Jogador fake = new Jogador();
                fake.origem = new Ponto(p1);
                fake.destino = new Ponto(p2);
                int distancia = Lista.aEstrela(m, fake).lista.size()
- 1;
                return Math.abs(distancia / 7);

        }

        public void recursoLimite(Jogador j) {
                int soma = 0, tamanho = j.possiveis.lista.size();
                for (Predio p : j.possiveis.lista) {
                        soma += p.maior;
                }
                j.recursoConstrucao = soma / tamanho;
        }

        public void ajustarPesos(Jogador j) {
                j.PesoSoldadoHeroi = 0;
                for (int i = 0; i < 3; i++) {
                        if (j.soldados[i] != null) {
                                j.PesoSoldadoHeroi +=
j.soldados[i].qtde * j.soldados[i].p.peso;
                        }
                }
        }

}
```

2. Vila.Java

Nesta classe estão contidas as informações da vila (atributos e recursos) bem como as funções auxiliares de comprar soldados e construir prédios.

```
	package br.ufla.americatribal.vila;

import java.util.Random;

import br.ufla.americatribal.batalha.Personagem;
import br.ufla.americatribal.geral.Jogador;

public class Vila {
	public int[] recursos = { 4, 4, 4 };
	public int[] carece = { 0, 0, 0 };
	int turno = 0;
	int[] bonusUnidade = { 0, 0, 0, 0, 0 };
	int[] heroi = { 0, 0, 0, 0, 0 };
	int[] bancoDeUnidade = new int[5];
	public int[] base = { 0, 0, 0, 0, 0 };
	float[] defesas = new float[2];
	public float[] recursoBonus = { 2, 2, 2 };
	int fatorDeSoldadosVila = 0;
	int fatorDeSoldadosHeroi;
	public int[] faltaSoldadosVila = { 0, 0, 0, 0, 0 };

	public Vila() {
	}

	// faz as alterações nos valores da vila quando um predio
é construido
	public void construir(String id, ListaV possuidos, ListaV
restantes) {
		System.out.println("Construiu " + id + "------------
");
		Predio p = restantes.encontraPredio(id);

		for (int i = 0; i < 3; i++) {
			this.recursos[i] -= p.custo[i];
			if (p.recursoB[i] >= 1) {
				this.recursoBonus[i] +=
p.recursoB[i];// valor absoluto
			} else {
				this.recursoBonus[i] +=
this.recursoBonus[i] * p.recursoB[i];// valor
```

```
	// percentual
                }
            }
            switch (p.tipoU[0]) {
            case "basico":
                System.out.println("basico antes" +
this.bancoDeUnidade[0]);
                this.bancoDeUnidade[0] += p.unidadeB[0];
                this.bonusUnidade[0] += p.unidadeB[0];
                System.out.println("basico depois" +
this.bancoDeUnidade[0]);
                break;
            case "animal":
                System.out.println("animal antes" +
this.bancoDeUnidade[2]);
                this.bancoDeUnidade[2] += p.unidadeB[0];
                this.bonusUnidade[2] += p.unidadeB[0];
                System.out.println("animal depois" +
this.bancoDeUnidade[2]);
                break;
            case "mistico":
                System.out.println("mistico antes" +
this.bancoDeUnidade[3]);
                this.bancoDeUnidade[3] += p.unidadeB[0];
                this.bonusUnidade[3] += p.unidadeB[0];
                System.out.println("mistico depois" +
this.bancoDeUnidade[3]);
                break;
            case "mitico":
                this.bancoDeUnidade[4] += p.unidadeB[0];
                this.bonusUnidade[4] += p.unidadeB[0];
                break;
            }
            if (p.tipoU[1].equals("longe")) {
                System.out.println("longe antes" +
this.bancoDeUnidade[1]);
                this.bancoDeUnidade[1] += p.unidadeB[1];
                this.bonusUnidade[1] += p.unidadeB[1];
                this.bonusUnidade[0] += p.unidadeB[0];
                System.out.println("longe depois" +
this.bancoDeUnidade[1]);
            }
            switch (p.tipoA) {
```

```
                case "df":
                        this.defesas[0] += p.atributoB;
                        break;
                case "dm":
                        this.defesas[1] += p.atributoB;
                        break;

                }

                restantes.lista.remove(p);
                possuidos.lista.add(p);
        }

        // faz as alterações nos valores da vila quando uma
quantidade de unidades é
        // escolhida
        public boolean comprar(int classe, int qtde, Jogador j)
{// compra uma

                                        // quantidade
                // "qtde"
                // de soldados tipo "classe"

                Personagem rec = new Personagem(classe + 1);
                int aux = this.quantidadeDeSoldadosPossiveis(classe,
qtde, rec);
                if (aux == 0)
                        return false;
                System.out.println("comprou " + aux + " soldados da
classe: "
                                + (classe) + "------------");
                this.bancoDeUnidade[classe] -= aux;
                this.base[classe] += aux;
                if (rec.custo[0] > 0)
                        this.recursos[0] -= rec.custo[0] * aux;
                if (rec.custo[1] > 0)
                        this.recursos[1] -= rec.custo[1] * aux;
                if (rec.custo[2] > 0)
                        this.recursos[2] -= rec.custo[2] * aux;
                return true;
        }

        public int quantidadeDeSoldadosPossiveis(int classe, int
qtde,
                        Personagem rec) {
```

```
			for (int i = 0; i < 3; i++) {
				boolean suficiente = false;
				if (rec.custo[i] > 0) {
					while (!suficiente) {
						if (rec.custo[i] * qtde > 
recursos[i]) {
							qtde--;
							carece[i]++;
							carece[i]++;
							if (qtde == 0) {
								carece[i] = 4;
								return 0;
							}
						} else {
							suficiente = true;
						}

					}

				}
			}
			if (this.bancoDeUnidade[classe - 1] < qtde) {
				return this.bancoDeUnidade[classe - 1];
			}
			return qtde;

		}

		// faz as alterações nos valores da vila quando o fim do 
turno chega
		public int fimDeTurno(int turno) {// utilizar
			this.recursos[0] += this.recursoBonus[0];
			this.recursos[1] += this.recursoBonus[1];
			this.recursos[2] += this.recursoBonus[2];
			if (turno % 7 == 0) {
				for (int j = 0; j < 5; j++) {
					this.bancoDeUnidade[j] += 
this.bonusUnidade[j];
				}

			}
			if ((turno > 20) && (turno < 40)) {
				this.fatorDeSoldadosVila = 1;
				if (turno > 30) {
					this.fatorDeSoldadosHeroi = 1;
```

```
                }
            } else if ((turno >= 40) && (turno < 60)) {
                this.fatorDeSoldadosVila = 2;
                if (turno > 50) {
                    this.fatorDeSoldadosHeroi = 2;
                }
            }

            System.out.println("vila" + fatorDeSoldadosVila);
            System.out.println("heroi" + fatorDeSoldadosHeroi);
            return this.turno++;
        }

        public boolean liberarConstrucao(Predio p, Jogador j) {
            if (p == null) {
                System.out.println("p null");
                return false;
            }
            if (p.id == null) {
                System.out.println("id null");
                return false;
            }
            if (p.prereq == null) {
                System.out.println("prereq null");
                return false;
            }
            if (p.prereq.lista == null) {
                System.out.println("listaq null");
                return false;
            }
            if (p.prereq.dentroDe(j.possuidos)) {
                if (p.maior<= j.recursoConstrucao) {
                    for (int i = 0; i < 3; i++) {
                        if (this.recursos[i] <
p.custo[i]) {
                            this.carece[i] += 4;
                            return false;
                        }
                    }
                }else{
                    return false;
                }

            } else {
                return false;
```

```
		}
		return true;
	}

	public int escolheRecurso() {
		Random rec = new Random();
		if (this.carece[0] == this.carece[1]) {
			if (this.carece[1] == this.carece[2]) {
				return
auxEscolheRecurso(this.recursos);
			} else if (this.carece[1] > this.carece[2]) {
				if (rec.nextBoolean())
					return 0;
				else
					return 1;
			} else
				return 2;

		} else if (this.carece[0] > this.carece[1]) {
			if (this.carece[0] == this.carece[2]) {
				if (rec.nextBoolean())
					return 0;
				else
					return 2;
			} else if (this.carece[0] > this.carece[2])
				return 0;
			else
				return 2;
		} else {
			if (this.carece[1] == this.carece[2]) {
				if (rec.nextBoolean())
					return 1;
				else
					return 2;
			} else if (this.carece[1] > this.carece[2])
				return 1;
			else
				return 2;
		}

	}

	public int auxEscolheRecurso(int[] vet) {
		Random rec = new Random();
		if (vet[0] == vet[1]) {
```

```
                        if (vet[1] == vet[2]) {
                                return rec.nextInt(3);
                        } else if (vet[1] > vet[2]) {
                                return 2;
                        } else {
                                return rec.nextInt(2);
                        }
                } else if (vet[0] < vet[1]) {
                        if (vet[0] == vet[2]) {
                                if (rec.nextBoolean()) {
                                        return 0;
                                } else {
                                        return 2;
                                }
                        } else if (vet[0] > vet[2]) {
                                return 2;
                        } else {
                                return 0;
                        }
                } else {
                        if (vet[1] == vet[2]) {
                                if (rec.nextBoolean()) {
                                        return 1;
                                } else {
                                        return 2;
                                }
                        } else if (vet[1] > vet[2]) {
                                return 2;
                        } else {
                                return 1;
                        }
                }
        }
}
```

3. Predio.Java

Nesta classe estão contidas as informações de todos os prédios do jogo. Algumas das informações são: custo, bônus e pré-requisitos. Alguns prédios são: aldeia nível 1, comida nível 1 e madeira nível 1.

```java
package br.ufla.americatribal.vila;

public class Predio {

	int[] custo = { 0, 0, 0 };
	float[] recursoB = { 0.f, 0.f, 0.f };
	int[] unidadeB = { 0, 0 };
	String[] tipoU = { "null", "null" };
	float atributoB = 0.f;
	String tipoA =new String( "null");
	public String id = new String("null");
	public ListaV prereq = new ListaV();
	public int nivel = 1;
	public int peso, atratividade, tipo ;
	public int maior =0;

	String classe;
	public int atratividadeBase;

	public Predio() {
	}


	public Predio(Predio p) {
		this.custo = p.custo;
		this.recursoB = p.recursoB;
		this.unidadeB = p.unidadeB;
		this.tipoU = p.tipoU;
		this.atributoB = p.atributoB;
		this.tipoA = p.tipoA;
		this.id = p.id;
		this.prereq = p.prereq;
		this.nivel = p.nivel;
		this.peso = p.peso;
		this.atratividade = p.atratividade;
		this.tipo = p.tipo;
		this.classe = p.classe;
	}


	// construtor
	public Predio(String id) {
		switch (id) {
```

```
case "al1":
      al1();
      break;
case "al2":
      al2();
      break;
case "al3":
      al3();
      break;
case "al4":
      al4();
      break;
case "to1":
      to1();
      break;
case "to2":
      to2();
      break;
case "to3":
      to3();
      break;
case "de1":
      de1();
      break;
case "de2":
      de2();
      break;
case "de3":
      de3();
      break;
case "ar1":
      ar1();
      break;
case "ar2":
      ar2();
      break;
case "tem":
      tem();
      break;
case "es1":
      es1();
      break;
case "es2":
      es2();
      break;
```

```
		case "co1":
			co1();
			break;
		case "co2":
			co2();
			break;
		case "co3":
			co3();
			break;
		case "ma1":
			ma1();
			break;
		case "ma2":
			ma2();
			break;
		case "ma3":
			ma3();
			break;
		case "pe1":
			pe1();
			break;
		case "pe2":
			pe2();
			break;
		case "pe3":
			pe3();
			break;
		default:
			id = new String("erro");
			break;
		}
}


// a partir daqui, valores de cada predio
public void al1() {
		id = new String("al1") ;
		classe = new String("base");
}

public void al2() {
		custo[1] = 10;
		custo[2] = 10;
		recursoB[0] = 1;
		recursoB[1] = 2;
```

```
        recursoB[2] = 1;
        id = new String("al2");
        nivel = 3;
        classe = new String("base");
        peso = 12;
        atratividadeBase = 3;
        maior = 10;

}

public void al3() {
        custo[1] = 30;
        custo[2] = 30;
        recursoB[0] = 1;
        recursoB[1] = 1;
        recursoB[2] = 1;
        id = new String("al3");
        nivel = 4;
        classe = new String("base");
        peso = 32;
        atratividadeBase = 3;
        maior = 30;

}

public void al4() {
        custo[1] = 60;
        custo[2] = 60;
        recursoB[0] = 1;
        recursoB[1] = 1;
        recursoB[2] = 1;
        id = new String("al4");
        nivel = 5;
        classe = new String("base");
        peso = 40;
        atratividadeBase = 3;
        maior = 60;
}

public void to1() {
        custo[1] = 10;
        atributoB = 0.5f;
        tipoA = new String("dm");
        id = new String("to1");
        nivel = 4;
```

```
		classe = new String("base");
		peso= 10;
		atratividadeBase = 5;
		maior =10;

	}

	public void to2() {
		custo[1] = 10;
		custo[2] = 5;
		atributoB = 0.15f;
		tipoA =new String( "dm");
		id = new String("to2");
		nivel = 5;
		classe = new String("base");
		peso= 9;
		atratividadeBase = 3;
		maior =10;
	}

	public void to3() {
		custo[0] = 40;
		custo[1] = 10;
		atributoB = 0.25f;
		unidadeB[0] = 5;
		tipoU[0] =new String( "mitico");
		tipoA =new String( "dm");
		id = new String("to3");
		nivel = 6;
		classe = new String("militar");
		peso= 27;
		atratividadeBase = 3;
		maior =40;
	}

	public void de1() {
		custo[1] = 10;
		custo[2] = 10;
		atributoB = 0.1f;
		tipoA =new String( "df");
		id = new String("de1");
		nivel = 3;
		classe = new String("base");
		peso= 12;
		atratividadeBase = 3;
```

```
        maior =10;
}

public void de2() {
        custo[2] = 15;
        atributoB = 0.2f;
        tipoA = new String("df");
        id = new String("de2");
        nivel = 4;
        classe = new String("base");
        peso= 15;
        atratividadeBase = 5;
        maior =15;
}

public void de3() {
        custo[0] = 15;
        custo[1] = 10;
        atributoB = 0.25f;
        tipoA =new String( "df");
        id = new String("de3");
        nivel = 7;
        classe = new String("base");
        peso= 14;
        atratividadeBase = 3;
        maior =15;
}

public void ar1() {
        custo[0] = 5;
        custo[1] = 5;
        custo[2] = 5;
        unidadeB[0] = 5;
        tipoU[0] =new String( "basico");
        id = new String("ar1");
        nivel = 2;
        classe = new String("militar");
        peso= 8;
        atratividadeBase = 1;
        maior =5;
}

public void ar2() {
        custo[0] = 15;
        custo[1] = 10;
```

```
        custo[2] = 5;
        unidadeB[0] = 5;
        tipoU[0] =new String( "basico");
        unidadeB[1] = 3;
        tipoU[1] =new String( "longe");
        id = new String("ar2");
        nivel = 3;
        classe = new String("militar");
        peso= 13;
        atratividade = 1;
        maior =15;
}

public void tem() {
        custo[0] = 20;
        custo[1] = 20;
        custo[2] = 20;
        unidadeB[0] = 5;
        tipoU[0] =new String( "mistico");
        atributoB = 0.2f;
        tipoA =new String( "df");
        id = new String("tem");
        nivel = 4;
        classe = new String("militar");
        peso= 23;
        atratividadeBase = 1;
        maior =20;
}

public void es1() {
        custo[0] = 15;
        custo[1] = 15;
        custo[2] = 15;
        unidadeB[0] = 2;
        tipoU[0] =new String( "animal");
        id = new String("es1");
        nivel = 3;
        classe = new String("militar");
        peso= 18;
        atratividadeBase = 1;
        maior =15;
}

public void es2() {
        custo[0] = 20;
```

```
        custo[1] = 15;
        custo[2] = 10;
        unidadeB[0] = 4;
        tipoU[0] =new String( "animal");
        id = new String("es2");
        nivel = 5;
        classe = new String("militar");
        peso= 18;
        atratividadeBase = 1;
        maior =20;
}

public void co1() {
        custo[0] = 5;
        recursoB[0] = 0.1f;
        id = new String("co1");
        nivel = 2;
        classe = new String("recursos");
        peso= 5;
        atratividadeBase = 5;
        maior =5;
}

public void co2() {
        custo[0] = 10;
        recursoB[0] = 0.2f;
        tipoA = new String( "null");
        id = new String("co2");
        nivel = 3;
        classe = new String("recursos");
        peso= 10;
        atratividadeBase = 5;
        maior =10;
}

public void co3() {
        custo[0] = 15;
        recursoB[0] = 0.3f;
        id = new String("co3");
        nivel = 4;
        classe = new String("recursos");
        peso= 15;
        atratividadeBase = 5;
        maior =15;
}
```

```
public void ma1() {
	custo[1] = 5;
	recursoB[1] = 0.1f;
	id = new String("ma1");
	nivel = 3;
	classe = new String("recursos");
	peso= 5;
	atratividadeBase = 5;
	maior =5;
}

public void ma2() {
	custo[1] = 10;
	recursoB[1] = 0.2f;
	id = new String("ma2");
	nivel = 4;
	classe = new String("recursos");
	peso= 10;
	atratividadeBase = 5;
	maior =10;
}

public void ma3() {
	custo[1] = 15;
	recursoB[1] = 0.3f;
	id = new String("ma3");
	nivel = 5;
	classe = new String("recursos");
	peso= 15;
	atratividadeBase = 5;
	maior =15;
}

public void pe1() {
	custo[2] = 5;
	recursoB[2] = 0.1f;
	id = new String("pe1");
	nivel = 3;
	classe = new String("recursos");
	peso= 5;
	atratividadeBase = 5;
	maior =5;
}
```

```
	public void pe2() {
		custo[1] = 10;
		recursoB[2] = 0.2f;
		id = new String("pe2");
		nivel = 4;
		classe = new String("recursos");
		peso= 10;
		atratividadeBase = 5;
		maior =10;
	}

	public void pe3() {
		custo[1] = 15;
		recursoB[2] = 0.3f;
		id = new String("pe3");
		nivel = 5;
		classe = new String("recursos");
		peso= 15;
		atratividadeBase = 5;
		maior =15;
	}
}
```

4. ListaV.Java

Esta classe é responsável pela manipulação da classe Predio.java em forma de lista para facilitar a verificação de quais prédios estão disponíveis e quais prédios são requisitos de um prédio em questão .

```
package br.ufla.americatribal.vila;


import java.util.ArrayList;

public class ListaV {
	public ArrayList<Predio> lista = null;
```

```
	public ListaV() {
		lista = new ArrayList<Predio>();
	}



	// confere se uma lista esta dentro da outra
	public boolean dentroDe(ListaV l) {
		int tam = this.lista.size();
		for (Predio p : this.lista) {
			for (Predio p2 : l.lista) {
				if (p.equals(p2)) {
					tam--;
				}
			}
		}
		if (tam == 0) {
			return true;
		}
		return false;
	}

	public boolean encontraId(String id) {
		for (Predio p : this.lista) {
			if (p.id.equals(id)) {
				return true;
			}
		}
		;
		return false;
	}

	public Predio encontraPredio(String id) {
		for (Predio p : this.lista) {
			if (p.id.equals(id)) {
				return p;
			}
		}
		return null;
	}


	public ListaV separaPorNivel(int nivel) {// precisa fazer
um if para
```

```
		// incrementar o nivel quando

		// vazio
		ListaV aux = new ListaV();
		System.out.println(nivel);
		for (Predio p : this.lista) {
			if (p.nivel == nivel) {
				System.out.println(p.id);
				aux.lista.add(p);
			}
		}
		return aux;
}

// inicializa os predios com valores e pre-requisitos
public void inicializaRestasnte(ListaV possuidos) {
		Predio al2 = new Predio("al2");
		Predio co1 = new Predio("co1");
		Predio ar1 = new Predio("ar1");
		Predio co2 = new Predio("co2");
		Predio pe1 = new Predio("pe1");
		Predio ma1 = new Predio("ma1");
		Predio al3 = new Predio("al3");
		Predio es1 = new Predio("es1");
		Predio ar2 = new Predio("ar2");
		Predio de1 = new Predio("de1");
		Predio co3 = new Predio("co3");
		Predio pe2 = new Predio("pe2");
		Predio ma2 = new Predio("ma2");
		Predio al4 = new Predio("al4");
		Predio tem = new Predio("tem");
		Predio to1 = new Predio("to1");
		Predio de2 = new Predio("de2");
		Predio pe3 = new Predio("pe3");
		Predio ma3 = new Predio("ma3");
		Predio es2 = new Predio("es2");
		Predio to2 = new Predio("to2");
		Predio to3 = new Predio("to3");
		Predio de3 = new Predio("de3");
		//1
		co1.prereq.lista.add(possuidos.lista.get(0));
		this.lista.add(co1);
		ar1.prereq.lista.add(possuidos.lista.get(0));
		this.lista.add(ar1);
```

```
//3
al2.prereq.lista.add(possuidos.lista.get(0));
this.lista.add(al2);
co2.prereq.lista.add(possuidos.lista.get(0));
co2.prereq.lista.add(co1);
this.lista.add(co2);
pe1.prereq.lista.add(al2);
this.lista.add(pe1);
ma1.prereq.lista.add(al2);
this.lista.add(ma1);
es1.prereq.lista.add(al2);
this.lista.add(es1);
ar2.prereq.lista.add(ar1);
ar2.prereq.lista.add(al2);
this.lista.add(ar2);
de1.prereq.lista.add(ar1);
this.lista.add(de1);
//10
al3.prereq.lista.add(al2);
this.lista.add(al3);
co3.prereq.lista.add(co2);
co3.prereq.lista.add(al3);
this.lista.add(co3);
pe2.prereq.lista.add(pe1);
pe2.prereq.lista.add(al3);
this.lista.add(pe2);
ma2.prereq.lista.add(ma1);
ma2.prereq.lista.add(al3);
this.lista.add(ma2);
tem.prereq.lista.add(al3);
this.lista.add(tem);
to1.prereq.lista.add(ar2);
this.lista.add(to1);
de2.prereq.lista.add(de1);
de2.prereq.lista.add(ma1);
this.lista.add(de2);
//17
al4.prereq.lista.add(al3);
this.lista.add(al4);
pe3.prereq.lista.add(pe2);
pe3.prereq.lista.add(al4);
this.lista.add(pe3);
ma3.prereq.lista.add(ma2);
ma3.prereq.lista.add(al4);
this.lista.add(ma3);
```

```
        es2.prereq.lista.add(es1);
        es2.prereq.lista.add(al4);
        this.lista.add(es2);
        to2.prereq.lista.add(to1);
        to2.prereq.lista.add(al4);
        to2.prereq.lista.add(tem);
        this.lista.add(to2);
        //22
        to3.prereq.lista.add(al4);
        to3.prereq.lista.add(to2);
        this.lista.add(to3);
        //23
        de3.prereq.lista.add(de2);
        de3.prereq.lista.add(to3);
        this.lista.add(de3);
        //24
    }


}
```

8 Apêndice B

Implementação do algoritmo de movimentação e do mapa

Esta é a implementação do algoritmo de movimentação pelo mapa bem como as informações e funções auxiliares para a construção do mesmo. Este apêndice corresponde ao pacote br.ufla.americatribal.campo e compreende as classes carregarMapa.java, Lista.java, Mapa.java e Ponto.java .

1. CarregarMapa.java

Esta classe é responsável por pegar os dados do arquivo texto contendo as informações do mapa e carregando ela nas variáveis responsáveis por sua manipulação

```
package br.ufla.americatribal.campo;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class CarregarMapa {
	public int mapa[][];
	public int chao[][];
	private int objeto[][];
	public int tam_x = 0;
	public int tam_y = 0;

	private void criarMapaIA(){
		mapa = new int[tam_x][tam_y];

		for (int y = 0; y < tam_y; y++){
			for (int x = 0; x < tam_x; x++){
			//for (int y = 0; y < tam_y; y++){
				mapa[x][tam_y-y-1] = 0;
```

```
                        // aldeia jogador
                        if (objeto[x][tam_y-y-1] == 8){
                              mapa[x][tam_y-y-1] =
Mapa.JOG_ALDEIA;
                        // explorador jogador
                        }else if (objeto[x][tam_y-y-1] == 10){
                              mapa[x][tam_y-y-1] =
Mapa.JOG_EXPLOR;

                        // aldeia computador
                        }else if (objeto[x][tam_y-y-1] == 9){
                              mapa[x][tam_y-y-1] =
Mapa.COM_ALDEIA;
                        // explorador computador
                        }else if (objeto[x][tam_y-y-1] == 11){
                              mapa[x][tam_y-y-1] =
Mapa.COM_EXPLOR;

                        // recursos infinitos
                        }else if (objeto[x][tam_y-y-1] > 0 &&
                                    objeto[x][tam_y-y-1] < 4)
                        {
                              mapa[x][tam_y-y-1] =
objeto[x][tam_y-y-1] + 1;
                        // recursos finitos (não são alterados)
                        }else if (objeto[x][tam_y-y-1] > 4 &&
                                    objeto[x][tam_y-y-1] < 8)
                        {
                              mapa[x][tam_y-y-1] =
objeto[x][tam_y-y-1];

                        // bloqueio pelo chao
                        }else if ((objeto[x][tam_y-y-1] == 0 &&
                                    chao[x][tam_y-y-1] > 2) ||
                                    objeto[x][tam_y-y-1] == 4)
                        {
                              mapa[x][tam_y-y-1] =
Mapa.obstaculo;
                        }

                  }
            }
      }

      private int pegarInteiro(String s){
```

```
		if (s.isEmpty())
			return 0;

		String e = "";
		for (int i = 0; i < s.length(); i++){
			if (s.charAt(i) != ' '){
				e += s.charAt(i);
			}
		}

		return Integer.valueOf(e);
	}

	public boolean carregar(String nomeMapa){
		// ".atm" = "América Tribal Mapa"
		File arquivo = new File(nomeMapa+".atm");
		try {
			if (arquivo.exists()) {
				//faz a leitura do arquivo
				FileReader fr = new
FileReader(arquivo);

				BufferedReader br = new
BufferedReader(fr);

				String texto = "";

				//equanto houver mais linhas
				while (br.ready()) {
					//lê a proxima linha
					String linha = br.readLine();
					texto += linha+"\n";
				}

				br.close();
				fr.close();

				String [] dados = texto.split("\n");

				String [] dimensoes =
dados[1].split(":");
				dimensoes = dimensoes[1].split(",");
				tam_x = Integer.valueOf(dimensoes[0]);
				tam_y = Integer.valueOf(dimensoes[1]);
```

```
                        chao = new int[tam_x][tam_y];
                        objeto = new int[tam_x][tam_y];
                        mapa = new int[tam_x][tam_y];

                        for (int y = 0; y < tam_y; y++){
                              String [] linhachao =
dados[y+3].split(",");
                              String [] linhaobj =
dados[y+3+tam_y+1].split(",");


                              for (int x = 0; x < tam_x; x++){
                                    chao[x][tam_y-y-1] =
pegarInteiro(linhachao[x]);
                                    objeto[x][tam_y-y-1] =
pegarInteiro(linhaobj[x]);

                              }
                        }

                        criarMapaIA();

                        return true;

                  }else{
                        System.out.println("\n---\nMapa não
pode ser carregado\n---\n");
                        return false;
                  }

            } catch (IOException ex) {
                  ex.printStackTrace();
            }
            return false;
      }
}
```

2. Ponto.java

Esta classe contém as informações referentes a cada posição da matriz do mapa, como exemplo: tipo de terreno (obstáculo ou recurso) e qual o ponto anterior a ele, segundo o caminho do herói.

```
package br.ufla.americatribal.campo;


public class Ponto {
	public int x = 0;
	public int y = 0;
	public int qtde=0;

	public String tipo;
	int g = 0;
	int h = 0;
	int f = 0;

	int distancia;
	int custo = 10;

	public Ponto pai = null;

	public Ponto(){}

	public Ponto(int x, int y){
		this.x = x;
		this.y = y;
	}
	public Ponto(int x, int y, Ponto pai){
		this.x = x;
		this.y = y;
		this.pai = pai;
	}
	public Ponto(Ponto p){
		this.x = p.x;
		this.y = p.y;

		this.g = p.g;
		this.h = p.h;
		this.f = p.f;

		this.custo = p.custo;
```

```
            this.pai = p.pai;
            this.tipo = p.tipo;
        }
        public void custos(Ponto destino){
            g = custo;

            if (pai != null){
                g += pai.g;
            }

            int dx = Math.abs(this.x - destino.x);
            int dy = Math.abs(this.y - destino.y);

            h = (dx + dy) * custo;

            f = g + h;
        }
        public boolean igual(Ponto p){
            return igual(p.x, p.y);
        }
        public boolean igual(int x, int y){
            if (this.x == x && this.y == y)
                return true;

            return false;
        }
        public Ponto cima(){
            return new Ponto(x, y - 1);
        }
        public Ponto baixo(){
            return new Ponto(x, y + 1);
        }
        public Ponto esquerda(){
            return new Ponto(x - 1, y);
        }
        public Ponto direita(){
            return new Ponto(x + 1, y);
        }
}
```

3. Mapa.java

Esta classe armazena a matriz que representa o mapa bem como funções e variáveis auxiliares a movimentação.

```
package br.ufla.americatribal.campo;

import java.util.Random;

import br.ufla.americatribal.geral.Jogador;

public class Mapa {

	public static final String E = "E";
	public static final String D = "D";
	public static final String C = "C";
	public static final String B = "B";
	final float recursoPermanente = 0.3f;

	public int mapa[][];
	public int chao[][];

	public int ty = 0;
	public int tx = 0;

	public Ponto casa = new Ponto();
	public Ponto heroi = new Ponto();
	Ponto destino = new Ponto();
	public Ponto casa_inimigo = new Ponto();
	public Ponto inimigo = new Ponto();
	public Lista comidas = new Lista();
	public Lista madeiras = new Lista();
	public Lista pedras = new Lista();
	Random sorteio = new Random();

	public Ponto aldeiaJogador;
	public Ponto aldeiaComputador;
	public Ponto exploradorJogador;
	public Ponto exploradorComputador;

	public static int obstaculo = 1;
	public static final int JOG_ALDEIA = 10;
	public static final int JOG_EXPLOR = 11;
```

```
	public static final int COM_ALDEIA = 20;
	public static final int COM_EXPLOR = 21;

	public static final int COMIDA = 2;
	public static final int MADEIRA = 3;
	public static final int PEDRA = 4;
	public static final int COMIDA_COL = 5;
	public static final int MADEIRA_COL = 6;
	public static final int PEDRA_COL = 7;

	int perigo = 9;

	public Mapa() {
	}

	public Mapa(String nomemapa) {
		carregarDeArquivo(nomemapa);
	}

	public void carregarDeArquivo(String nomemapa) {
		CarregarMapa cm = new CarregarMapa();
		cm.carregar(nomemapa);

		this.chao = cm.chao;
		this.mapa = cm.mapa;
		this.tx = cm.tam_x;
		this.ty = cm.tam_y;

		listaDeRecursos();
	}

	public Mapa(int[][] mapa) {
		this.mapa = mapa;
	}

	public Mapa(int mapa[][], Ponto heroi, Ponto destino, int
obs) {
		this.mapa = mapa;
		this.heroi = heroi;
		this.destino = destino;
		this.ty = mapa.length;
		this.tx = mapa[0].length;
		obstaculo = obs;
	}
```

```
	public Mapa(int mapa[][], Ponto heroi, Ponto destino) {
		this.mapa = mapa;
		this.heroi = heroi;
		this.destino = destino;
		this.ty = mapa.length;
		this.tx = mapa[0].length;
	}

	public void listaDeRecursos() {
		System.out.println("Mapas " + tx + ", " + ty);
		for (int y = 0; y < this.ty; y++) {
			for (int x = 0; x < this.tx; x++) {
				switch (mapa[x][y]) {
				case (COMIDA):
					Ponto p1 = new Ponto(x, y);
					p1.tipo = "VAZIO";
					p1.qtde = 1;
					this.comidas.adicionar(p1);
					break;
				case (MADEIRA):
					Ponto p2 = new Ponto(x, y);
					p2.tipo = "VAZIO";
					p2.qtde = 1;
					this.madeiras.adicionar(p2);
					break;
				case (PEDRA):
					Ponto p3 = new Ponto(x, y);
					p3.tipo = "VAZIO";
					p3.qtde = 1;
					this.pedras.adicionar(p3);
					break;
				case (COMIDA_COL):
					Ponto p4 = new Ponto(x, y);
					p4.tipo = "UNICO";
					p4.qtde = sorteio.nextInt(3) +
1;
					this.comidas.adicionar(p4);
					break;
				case (MADEIRA_COL):
					Ponto p5 = new Ponto(x, y);
					p5.tipo = "UNICO";
					p5.qtde = sorteio.nextInt(3) +
1;
					this.madeiras.adicionar(p5);
					break;
```

```
                        case (PEDRA_COL):
                              Ponto p6 = new Ponto(x, y);
                              p6.tipo = "UNICO";
                              p6.qtde = sorteio.nextInt(3) +
1;
                              this.pedras.adicionar(p6);
                              break;
                        case (JOG_ALDEIA):
                              aldeiaJogador = new Ponto(x, y);
                              casa = new Ponto(aldeiaJogador);
                              break;
                        case (JOG_EXPLOR):
                              exploradorJogador = new Ponto(x,
y);
                              heroi = new
Ponto(exploradorJogador);
                              break;
                        case (COM_ALDEIA):
                              aldeiaComputador = new Ponto(x,
y);
                              casa_inimigo = new
Ponto(aldeiaComputador);
                              break;
                        case (COM_EXPLOR):
                              exploradorComputador = new
Ponto(x, y);
                              inimigo = new
Ponto(exploradorComputador);
                              break;
                        }

                  }
            }
      }

      public boolean passavel(int x, int y) {
            if (x < 0 || x >= tx || y < 0 || y >= ty)
                  return false;

            if (mapa[x][y] == obstaculo) {
                  return false;
            }
            return true;
      }
```

```
	public boolean passavel(Ponto p) {
		return passavel(p.x, p.y);
	}

	public boolean passavel(Ponto p, String lado) {
		switch (lado) {
		case E:
			return passavel(p.x - 1, p.y);
		case D:
			return passavel(p.x + 1, p.y);
		case C:
			return passavel(p.x, p.y - 1);
		case B:
			return passavel(p.x, p.y + 1);
		}
		return false;
	}

	public boolean noMapa(Ponto p) {
		if (p.x < 0 || p.x >= tx || p.y < 0 || p.y >= ty)
			return false;
		return true;

	}

	// retorna o ponto da lista mais proximo da origem
	public Ponto destinoOp(Jogador j) {// testado
		Jogador fake = new Jogador();
		fake.origem = new Ponto(j.origem);
		int tam = 10000, distancia;
		Ponto aux = null;
		for (Ponto p : j.opcoes.lista) {
			if (p.tipo.equals(j.id)) {
				continue;
			} else {
				fake.destino = p;
				distancia = Lista.aEstrela(this,
fake).lista.size() - 1;
				if (distancia == 0) {
					continue;
				} else {
					if (distancia < tam) {
						tam = distancia;
						aux = new Ponto(p);
					}
```

```
                        }
                    }
                }
                return aux;
        }
}
```

4. Lista.java

Esta classe é onde esta implementado o algoritmo A* bem como funções auxiliares para a execução desse algoritmo.

```
package br.ufla.americatribal.campo;

import java.util.ArrayList;

import br.ufla.americatribal.geral.Jogador;


public class Lista {
        public ArrayList<Ponto> lista = null;

        public Lista() {
                lista = new ArrayList<Ponto>();
        }

        public Lista(Ponto p) {
                lista = new ArrayList<Ponto>();
                lista.add(p);
        }

        public Lista(Lista l) {
                this.lista = l.lista;
        }

        public void adicionar(Ponto p) {
                lista.add(p);
        }
```

```
	public void adicionar(int x, int y){
		Ponto p= new Ponto(x,y);
		lista.add(p);
	}

	public void adicionar(Lista l) {
		if (l != null && l.lista != null &&
!l.lista.isEmpty()) {
			for (Ponto p : l.lista) {
				if (!tem(p))
					adicionar(p);
			}
		}
	}

	public void remover(Ponto p) {
		if (lista.isEmpty())
			return;
		for (Ponto pa : lista) {
			if (p.igual(pa)) {
				lista.remove(p);
				return;
			}
		}
	}

	public void remover(int x, int y) {
		if (this.lista.isEmpty())
			return;
		System.out.println(x+" "+y);
		for (Ponto pa : this.lista) {
			System.out.println("doido "+pa.x+" "+ pa.y);
			if (pa.igual(x, y)) {
				System.out.println("aquiiiiiii");
				this.lista.remove(pa);
				return;
			}
		}
	}
	public boolean tem(Ponto p) {
		if (lista.isEmpty())
			return false;

		for (Ponto pa : lista) {
```

```
                if (p.igual(pa))
                        return true;
        }

        return false;
}

public void trocarPai(Ponto p, Ponto pai, Ponto destino) {
        if (lista.isEmpty())
                return;

        for (Ponto pa : lista) {
                if (p.igual(pa)) {
                        pa.pai = pai;
                        pa.custos(destino);
                }
        }
}

public Ponto menor() {
        if (lista.isEmpty())
                return null;

        Ponto menor = new Ponto();
        menor.f = 10000000;
        boolean achou = false;

        for (Ponto pa : lista) {
                if (menor.f > pa.f) {
                        menor = pa;
                        achou = true;
                }
        }

        if (achou == false)
                menor = null;

        return menor;
}

public void inverter() {
        if (lista.isEmpty())
                return;

        ArrayList<Ponto> l = new ArrayList<Ponto>();
```

```
			for (int i = lista.size() - 1; i >= 0; i--) {
				l.add(lista.get(i));
			}

			lista = l;
		}

		public Ponto pegar(Ponto p) {
			for (Ponto pa : lista) {
				if (p.igual(pa)) {
					return pa;
				}
			}
			return null;
		}

		public Ponto pegar(int posicao){
			int i = 0;
			for (Ponto ponto : lista){
				if (i == posicao)
					return new Ponto(ponto);
				else
					i++;
			}
			return null;
		}

		public static Lista adjacentes(Ponto p, Mapa mapa, Lista
aberta,
				Lista fechada) {
			Lista li = new Lista();

			Ponto pc = p.cima();
			if (!fechada.tem(pc)) {
				if (mapa.passavel(pc)) {
					if (!aberta.tem(pc))
						pc.pai = p;
					else if (pc.g > p.g + pc.custo) {
						pc.pai = p;
						aberta.trocarPai(pc, p,
mapa.destino);
					}
					pc.custos(mapa.destino);
```

```
                        li.adicionar(pc);
                }
        }

        Ponto pb = p.baixo();
        if (!fechada.tem(pb)) {
                if (mapa.passavel(pb)) {
                        if (!aberta.tem(pb))
                                pb.pai = p;
                        else if (pb.g > p.g + pb.custo) {
                                pb.pai = p;
                                aberta.trocarPai(pb, p,
mapa.destino);
                        }
                        pb.custos(mapa.destino);

                        li.adicionar(pb);
                }
        }

        Ponto pe = p.esquerda();
        if (!fechada.tem(pe)) {
                if (mapa.passavel(pe)) {
                        if (!aberta.tem(pe))
                                pe.pai = p;
                        else if (pe.g > p.g + pe.custo) {
                                pe.pai = p;
                                aberta.trocarPai(pe, p,
mapa.destino);
                        }
                        pe.custos(mapa.destino);

                        li.adicionar(pe);
                }
        }

        Ponto pd = p.direita();
        if (!fechada.tem(pd)) {
                if (mapa.passavel(pd)) {
                        if (!aberta.tem(pd))
                                pd.pai = p;
                        else if (pd.g > p.g + pd.custo) {
                                pd.pai = p;
                                aberta.trocarPai(pd, p,
mapa.destino);
```

```
                        }
                        pd.custos(mapa.destino);

                        li.adicionar(pd);
                }
        }

        return li;
}


public static Lista aEstrela(Mapa mapa, Jogador j) {
        Lista la = new Lista();
        Lista lf = new Lista();
        Lista adj = null;
        Lista caminho = null;
        la.adicionar(j.origem);

        adj = adjacentes(j.origem, mapa, la, lf);

        la.adicionar(adj);

        lf.adicionar(j.origem);
        la.remover(j.origem);

        while (!lf.tem(j.destino) && !la.lista.isEmpty()) {
                Ponto atual = la.menor();
                if (atual == null){
                        System.out.println("atual vazio");
                        return null;
                }

                lf.adicionar(atual);
                la.remover(atual);
                adj = adjacentes(atual, mapa, la, lf);
                la.adicionar(adj);

        }

        if (la.lista.isEmpty()){
                return null;
        }

        caminho = new Lista();
```

```
			Ponto p = new Ponto(lf.pegar(j.destino));

			while ((p!= null)) {
				caminho.adicionar(p);
				p = p.pai;
			}
			caminho.inverter();
			return caminho;
		}
		public boolean contem(Ponto p) {
			for (Ponto o : this.lista) {
				if (p.x == o.x) {
					if (p.y == o.y) {
						return true;
					}
				}
			}
			return false;

		}
		public int posicao(int x,int y){
			int pos = 0;
			for (Ponto o : this.lista) {
				if((o.x == x)&&(o.y==y))
					return pos;
				pos++;
			}
			return -1;

		}

		public void imprimir(){
			System.out.println("--Caminho-- Tamanho:
"+lista.size());
			for (Ponto p : lista){
				System.out.print(" p("+p.x+","+p.y+")");
			}
			System.out.println();
			System.out.println("--");
		}

		public void imprimir(String texto){
			System.out.println("--"+texto+"-- Tamanho:
"+lista.size());
			for (Ponto p : lista){
```

```
                System.out.print(" p("+p.x+","+p.y+")");
            }
            System.out.println();
            System.out.println("--");
        }

        public void imprimir(int passos){
            if (passos >= lista.size())
                passos = lista.size() - 1;

            System.out.println("--Caminho-- Tamanho: "+passos);
            for (int i = 0; i < passos; i++){
                Ponto p = lista.get(i);
                System.out.print(" p("+p.x+","+p.y+")");
            }
            System.out.println();
            System.out.println("--");
        }

}
```

9 Apêndice C

Informações referentes a dados do Jogador

1. Jogador.java

Esta classe contém todas as informações referentes ao jogador, como a posição atual, qual sua vila, número de soldados no herói e prédios possuídos.

```
package br.ufla.americatribal.geral;

import java.util.Random;

import br.ufla.americatribal.batalha.Combatente;
import br.ufla.americatribal.campo.Lista;
import br.ufla.americatribal.campo.Ponto;
import br.ufla.americatribal.vila.ListaV;
import br.ufla.americatribal.vila.Predio;
import br.ufla.americatribal.vila.Regras;
import br.ufla.americatribal.vila.Vila;

public class Jogador {
	public Vila v = new Vila();
	public Combatente[] soldados = { null, null, null };
	public Ponto casa = new Ponto();
	public Ponto origem;
	public Ponto destino;
	public String id;
	public int compra;
	public int precisaRecurso = 0;
	public int nTec = 2;
	public int rc = 1;

	public Ponto explorador = new Ponto();

	Regras r = new Regras();
	public ListaV possuidos = new ListaV();
	public ListaV restantes = new ListaV();
	public ListaV possiveis = new ListaV();
```

```
public Lista opcoes = new Lista();

Random sorteio = new Random();
public int recursoMilitar;
public int recursoConstrucao;
public int PesoPredio = 0;
public int PesoSoldadoBase = 0;
public int PesoSoldadoHeroi = 17;
public int limitador;

public Jogador() {

}

public Jogador(String id) {
	this.id = new String(id);
	this.possuidos.lista.add(new Predio("a11"));
	this.restantes.inicializaRestasnte(possuidos);
	this.soldados[0] = new Combatente(1, 3);
	this.soldados[1] = new Combatente(2, 1);
}

public void atribuirCasa(Ponto ponto) {
	this.casa.x = ponto.x;
	this.casa.y = ponto.y;

}

public void atribuirExplorador(Ponto ponto) {
	this.explorador.x = ponto.x;
	this.explorador.y = ponto.y;

	this.origem = new Ponto(this.explorador);
}

public void alterarCarencia(String rec) {
	switch (rec) {
	case "co":
		this.v.carece[0] = 0;
		break;
	case "ma":
		this.v.carece[1] = 0;
		break;
	case "pe":
		this.v.carece[2] = 0;
```

```
                        break;
                }
        }

        public void Status() {
                System.out.println("Comida: " + this.v.recursos[0] +
" Madeira: "
                                + this.v.recursos[1] + " Pedra: " +
this.v.recursos[2]);
                System.out.println("Soldados na base");
                for (int i = 0; i < 5; i++) {
                        if (this.v.base[i] != 0)
                                System.out.println(this.v.base[i] + "
de classe: " + i);
                }
                System.out.println("Soldados no heroi");
                for (int i = 0; i < 3; i++) {
                        if (this.soldados[i] != null)

        System.out.println(this.soldados[i].qtde + " de classe: "
                                                +
this.soldados[i].p.classe);
                }
                for (Predio p : this.possuidos.lista) {
                        System.out.println("Predios de nivel " +
p.nivel + ": " + p.id
                                        + ". ");
                }

        }
   }
```

10 Apêndice D

Implementação do Sistema de Casos

Esta é a implementação do sistema de casos, do sistema de batalha e das funções auxiliares para as respectivas execuções. Este apêndice contém quatro classes que são: Combatente.java, Guerra.Java, ListaB.java e Personagem.java e é referente ao pacote: br.ufla.americatribal.batalha

1. Guerra.java

Esta classe contém o sistema de casos que é responsável pela escolha do soldado alvo e contém a implementação da mecânica de batalha e funções auxiliares.

```
package br.ufla.americatribal.batalha;

import br.ufla.americatribal.geral.Jogador;
import java.util.HashMap;
import java.util.Random;

import br.ufla.americatribal.vila.Vila;

public class Guerra {
	public Combatente[] time1 = new Combatente[3];
	public Combatente[] time2 = new Combatente[3];
	ListaB lutadores = new ListaB();
	ListaB timeA = new ListaB();
	ListaB timeB = new ListaB();
	public HashMap<String, Float> bd = new HashMap<String,
Float>();

	ListaB l = new ListaB();

	public Guerra() {

	}

	public Guerra(Combatente[] time1, Combatente[] time2) {
		this.time1 = time1;
		this.time2 = time2;
	}

	public boolean morte(Combatente c) {
		if (c.vidaAtual <= 0) {
			return true;
		}
```

```
			return false;
		}

		// grupo de combatentes da vila
		// numero de recurtas na base convertidos para combatentes
		public Combatente[] combatentesVila(Vila v, Jogador j) {
			Combatente[] base = new Combatente[5];
			for (int i = 0; i < 5; i++) {
				base[i] = new Combatente(i, v.base[i]);
			}
			return base;

		}

		public String disputa(Combatente[] grupo1, Combatente[]
grupo2) {
			Combatente cVez = new Combatente();
			Combatente cI = new Combatente();
			Iniciativa(grupo1);
			Iniciativa(grupo2);
			this.setPos(grupo1, grupo2);
			lutadores.addAll(grupo1, grupo2);
			int qtde1 = 0;
			int qtde2 = 0;

			for (int i = 0; i < grupo1.length; i++) {
				if (grupo1[i] != null && grupo1[i].vidaAtual
> 0)
					qtde1++;
			}
			for (int i = 0; i < grupo2.length; i++) {
				if (grupo2[i] != null && grupo2[i].vidaAtual
> 0)
					qtde2++;
			}

			while (qtde1 > 0 && qtde2 > 0) {
				mostrarTimes(grupo1, grupo2);

				cVez = this.vez(lutadores, qtde1, qtde2);//
verifica de quem é a vez
				// tirar os mortos das escolhas
				cI = this.escolhaBd(cVez, lutadores);
				if (cI == null) {
					cI = new Combatente();
```

```
                        cI = cVez.escolha(lutadores);
                }
                float vida = cI.vidaAtual;
                cVez.atacar(cI);
                if (morte(cI)) {
                        if (cI.pos.equals("A")) {
                                qtde1--;
                        } else {
                                qtde2--;
                        }
                        System.out.println(qtde1 + " " +
qtde2);
                }
                vida /= cI.vidaAtual;
                bd.put(this.gerarChave(cVez, cI, lutadores),
1 / vida);

        }
        mostrarTimes(grupo1, grupo2);
        System.out.println("qtd1: "+qtde1 + ", qtd2: " +
qtde2);

        if (qtde2 == 0) {
                System.out.println("Time A venceu!");
                return "A";
        } else {
                System.out.println("Time B venceu!");
                return "B";
        }

    }

    private void mostrarTimes(Combatente[] grupo1,
Combatente[] grupo2){
        System.out.println("Time 1      Time 2");

        if (grupo1[0] != null)
                System.out.print(grupo1[0].vidaAtual);
        System.out.print("             ");
        if (grupo2[0] != null)
                System.out.print(grupo2[0].vidaAtual);
        System.out.println();


        if (grupo1[1] != null)
```

```
                System.out.print(grupo1[1].vidaAtual);
            System.out.print("              ");
            if (grupo2[1] != null)
                System.out.print(grupo2[1].vidaAtual);
            System.out.println();


            if (grupo1[2] != null)
                System.out.print(grupo1[2].vidaAtual);
            System.out.print("              ");
            if (grupo2[2] != null)
                System.out.print(grupo2[2].vidaAtual);
            System.out.println();
      }

      public String gerarChave(Combatente atq, Combatente cI,
ListaB inimigos) {
            String chave = new String();// key that will be
construct
            Integer aux = new Integer(atq.p.classe); //
auxiliary for conversion

                  // into String
            chave = aux.toString(); // Addition of Integer for
String
            aux = new Integer(atq.qtde);
            chave += aux.toString();
            aux = new Integer(cI.p.classe); // conversion into
String
            chave += aux.toString();
            for (Combatente c : inimigos.lista) {
                  if ((c != cI) || (atq.pos != cI.pos)) { //
deferente do defensor e

                  // dos aliados
                        if (c == null)
                              continue;
                        aux = new Integer(c.p.classe);
                        chave += aux.toString();
                  }
            }
            return chave;
      }
```

```
	public void setPos(Combatente[] grupo1, Combatente[]
grupo2) {
		for (int y = 0; y < grupo1.length; y++) {
			if (grupo1[y] != null) {
				if (grupo1[y].qtde != 0) {
					grupo1[y].pos = new String("A");
				}
			}
		}
		for (int y = 0; y < grupo2.length; y++) {
			if (grupo2[y] != null) {
				if (grupo2[y].qtde != 0) {
					grupo2[y].pos = new String("B");
				}
			}
		}
	}

	public Combatente vez(ListaB l, int qtde1, int qtde2) {
		while (!l.lista.isEmpty()) {
			Combatente aux = l.lista.get(0);
			l.lista.remove(0);
			if (aux.vidaAtual == 0) {
				if (aux.pos == "A") {
					qtde1--;
				} else {
					qtde2--;
				}
			} else {
				l.lista.add(aux);
				return aux;
			}
		}
		return null;

	}

	public void Iniciativa(Combatente[] grupo) {
		Random iniciativa = new Random();
		for (int i = 0; i < grupo.length; i++) {
			if (grupo[i] != null) {
				if (grupo[i].qtde != 0) {
					grupo[i].inicitiva =
iniciativa.nextFloat()
```

```
                                                *
grupo[i].p.iniciativa;
                              }
                        }
                  }

      }

      public Combatente escolhaBd(Combatente vez, ListaB
lutadores) {
            // procura do banco de dados todas os possiveis
casos e escolhe o melhor
            Combatente escolhido = new Combatente();
            float nota = 0;
            boolean achou = false;
            for (Combatente c : lutadores.lista) {
                  if (c != null) {
                        if ((!c.pos.equals(vez.pos)) &&
!(morte(c))) {
                              String chave =
this.gerarChave(vez, c, lutadores);
                              if (bd.containsKey(chave)) {
                                    if (bd.get(chave) > nota)
{
                                          escolhido = c;
                                          nota =
bd.get(chave);
                                          achou = true;
                                    }
                              }
                        }
                  }

            }
            if (achou) {
                  return escolhido;
            }
            return null;
      }

}
```

2. Personagem.java

Esta classe armazena os valores básicos dos soldados como o custo, a vida, ataque, defesa entre outros.

```
package br.ufla.americatribal.batalha;

public class Personagem {
	int ataque;
	boolean atq_mag;
	int vit;
	int def_fis;
	int def_mag;
	float iniciativa;
	int prob_acer;
	int prob_crit;
	int taxa_crit;
	public int peso;
	public int classe;
	public int[] custo = { 0, 0, 0 };
	String predioRequisito = null;
	int qtde = 0;

	public Personagem() {
	}

	public Personagem(int classe) {
		switch (classe) {
		case 1:
			basico();
			break;
		case 2:
			longe();
			break;
		case 3:
			animal();
			break;
		case 4:
			mistico();
			break;
```

```
		case 5:
			mitico();
			break;
		}
	}

	// dados das classe dos personagens
	public void basico() {
		ataque = 5;
		atq_mag = false;
		classe = 1;
		def_fis = 15;
		def_mag = 0;
		iniciativa = 0.21f;
		prob_acer = 90;
		prob_crit = 5;
		taxa_crit = 50;
		vit = 20;
		custo[1] = 5;
		predioRequisito = new String("ar1");
		peso = 4;

	}
	public void longe() {
		ataque = 11;
		atq_mag = false;
		classe = 2;
		def_fis = 15;
		def_mag = 25;
		iniciativa = 0.6f;
		prob_acer = 60;
		prob_crit = 50;
		taxa_crit = 75;
		vit = 25;
		custo[0] = 5;
		custo[1] = 5;
		predioRequisito = new String( "ar2");
		peso = 5;
	}

	public void animal() {
		ataque = 10;
		atq_mag = false;
		classe = 3;
		def_fis = 25;
```

```
        def_mag = 5;
        iniciativa = 0.31f;
        prob_acer = 80;
        prob_crit = 10;
        taxa_crit = 80;
        vit = 50;
        custo[0] = 10;
        predioRequisito = new String("es1");
        peso = 6;
}

public void mistico() {
        ataque = 12;
        atq_mag = true;
        classe = 4;
        def_fis = 5;
        def_mag = 50;
        iniciativa = 0.1f;
        prob_acer = 70;
        prob_crit = 40;
        taxa_crit = 30;
        vit = 10;
        custo[0] = 10;
        custo[2] = 10;
        predioRequisito = new String("tem");
        peso = 8;
}

public void mitico() {
        ataque = 15;
        atq_mag = true;
        classe = 5;
        def_fis = 55;
        def_mag = 85;
        iniciativa = 0.93f;
        prob_acer = 75;
        prob_crit = 25;
        taxa_crit = 100;
        vit = 90;
        custo[0] = 15;
        custo[1] = 15;
        custo[2] = 15;
        predioRequisito = new String("to3");
        peso = 12;
}
```

```
}
```

3. Combatente.java

Completa os dados da classe Personagem com valores referentes a batalha, como a vida atual, a qual time pertence, quantidade e iniciativa. Também possui uma função de escolha que alimenta a base de casos e a função de atacar.

```
package br.ufla.americatribal.batalha;

import java.util.Random;

public class Combatente {
	public Personagem p =new Personagem();
	float inicitiva;
	public int qtde;
	public int vidaAtual;
	String pos;

	public Combatente(int classe, int qtde) {
		this.p = new Personagem(classe);
		this.qtde = qtde;
		this.vidaAtual = this.p.vit * this.qtde;
	}

	public Combatente() {
	}

	public void atacar(Combatente defensor) {

		Random acerto = new Random();
		int acerto_atq = acerto.nextInt(100);
```

```
            int acerto_crit = acerto.nextInt(100);
            int aux = this.p.ataque + (int) (this.p.ataque *
(this.qtde - 1) * 0.3);
            if (acerto_atq <= this.p.prob_acer) {
                  System.out.println("soldado da classe "+
this.p.classe+" do time "+ this.pos+" atacou");
                  System.out.println("      soldado da classe
"+ defensor.p.classe+" do time "+ defensor.pos);

                  if (acerto_crit <= this.p.prob_crit) {
                        aux += aux * this.p.taxa_crit / 100;

                  }
                  if (this.p.atq_mag) {
                        aux -= aux * defensor.p.def_mag / 100;
                  } else {
                        aux -= aux * defensor.p.def_fis / 100;
                  }
                  System.out.print("Dano de "+ aux);
                  defensor.vidaAtual -= aux;
                  if (defensor.vidaAtual < 0) {
                        defensor.vidaAtual = 0;
                        System.out.println(" Morto!");
                  }else{
                        System.out.println(" Vida atual: "+
defensor.vidaAtual);
                  }

            }else{
                  System.out.println("soldado da classe "+
this.p.classe+" do time "+ this.pos+" errou!");
            }
      }

      public Combatente escolha(ListaB alvos) {
            int defesa;
            float aux;
            int aux1 = 0;
            float maior = 0;
            int pos = 0;
            aux = this.p.ataque * this.p.prob_acer / 100; //
quanto o ataque sera

                        // efetivo levando em
```

```
                        // conta a taxa de

                        // acerto
            for (Combatente c : alvos.lista) {
                  if (c.vidaAtual != 0) {// verifica se esta
vivo
                        if (!c.pos.equals(this.pos) ) {//
verifica se é inimigo
                              if (this.p.atq_mag) {// verifica
o tipo do ataque para

      // selecionar a defesa
                                    defesa = c.p.def_mag;
                              } else {
                                    defesa = c.p.def_fis;
                              }
                              aux = aux * (100 - defesa) /
100;
                              if (aux > maior) {
                                    maior = aux;
                                    aux1 = pos;
                              }
                        }
                  }
                  pos++;
            }
            return alvos.lista.get(aux1);
      }
}
```

4. ListaB.java

Responsável por estruturar a classe combatente em forma de lista para estabelecer na classe guerra a ordem de iniciativa.

```
package br.ufla.americatribal.batalha;
```

```
import java.util.ArrayList;

public class ListaB {
	public ArrayList<Combatente> lista = null;

	public ListaB() {
		lista = new ArrayList<Combatente>();
	}

	public void adicionar(Combatente c) {
		if (this != null && this.lista != null &&
!this.lista.isEmpty()) {
			int cont = 0;
			for (Combatente i : this.lista) {
				if (c == null) {
					continue;
				}
				if (c.qtde != 0) {
					continue;
				}
				System.out.println(c == null);
				System.out.println(c.qtde == 0);
				if (i.inicitiva >= c.inicitiva) {//
menor q o atual
					this.lista.add(cont, c);
					return;
				}
				cont++;
			}
			this.lista.add(c);
		} else {
			this.lista.add(c);
		}
	}

	public void addAll(Combatente[] grupo1, Combatente[]
grupo2) {
		for (int y = 0; y < grupo1.length; y++) {
			System.out.println(grupo1[y]==null);
			if (grupo1[y] != null){
				System.out.println("www");
				adicionar(grupo1[y]);
			}
```

```
        }
        for (int y = 0; y < grupo2.length; y++) {
              System.out.println(grupo2[y]==null);
              if (grupo2[y] != null)
                    adicionar(grupo2[y]);
        }
}
                              }
```

11 Apêndice E

Neste apêndice são mostrados a implementação das classes do pacote br.ufla.americatribal.ia: IA.java e config.java que são resposáveis por, respectivamente, carregar o módulo gráfico do mapa e estruturar a alternâcia de turno e configurar a relosução do tela.

1. IA.java

```
package br.ufla.americatribal.ia;

import java.util.Random;

import br.ufla.americatribal.campo.Mapa;
import br.ufla.americatribal.campo.Ponto;
import br.ufla.americatribal.geral.Jogador;
import br.ufla.americatribal.vila.Regras;
import br.ufla.americatribal.vila.Regras.Dificuldade;

import com.badlogic.gdx.ApplicationListener;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputProcessor;
import com.badlogic.gdx.graphics.GL10;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.g2d.TextureAtlas;
import com.badlogic.gdx.graphics.g2d.TextureRegion;

public class IA implements ApplicationListener, InputProcessor {
	// Visualização
	private OrthographicCamera camera;
	private SpriteBatch batch;
	private BitmapFont font;

	private static float CAM_TX = 16.0f;
	private static float CAM_TY = 9.0f;

	private static final float CAM_SIZE = 10;

	private static final float CEL_TX = 1.0f;
```

```
private static final float CEL_TY = 1.0f;

public static final int ALDEIA_A = 8;
public static final int ALDEIA_B = 9;
public static final int EXPLORADOR_A = 10;
public static final int EXPLORADOR_B = 11;

private static final String CAMINHO_IMAGENS =
            "data/imagens/america-tribal-imagens.pack";

private static final String ARQUIVO_MAPA = "teste";

// para mover o mapa
public static float escala = 1;

private static int tribo_a = 0;
private static int tribo_b = 2;

private float deslocx = 0;
private float deslocy = 0;


// mouse
/*
private int xNoMapa = -1;
private int yNoMapa = -1;

private int mousex = 0;
private int mousey = 0;
*/

// imagens
private TextureRegion [] aldeia;
private TextureRegion [] explorador;

private TextureRegion [] cenario_chao;
private TextureRegion [] cenario_objetos;

private TextureRegion [] aura;

Mapa mapa;

// variáveis de controle do teste
int turno = 1;
Regras r = new Regras();
```

```
	Jogador Primeiro = null;
	Jogador Segundo = null;
	Random jogadorUmPrimeiro = new Random();

	boolean podeAvancar = false;
	public boolean estaForaDeCombate = true;

	// inicialização
	@Override
	public void create() {
		mapa = new Mapa(ARQUIVO_MAPA);
		//CAM_TX = mapa.tx + 5;
		//CAM_TY = mapa.ty;

		//int tx = Gdx.graphics.getWidth();
		//int ty = Gdx.graphics.getHeight();

		//Gdx.graphics.setDisplayMode((int) CAM_TX * 10,
(int)CAM_TY * 10, false);

		ajeitarEscala();

		camera = new OrthographicCamera(CAM_TX, CAM_TY);
		camera.position.set(CAM_TX * 0.5f, CAM_TY * 0.5f,
0);
		camera.update();
		batch = new SpriteBatch();

		font = new BitmapFont();
		//font.setScale(0.009f);
		font.setScale(escala * 0.03f);
		font.setColor(0.8f,0.8f,0.8f,1);
		font.setUseIntegerPositions(false);

		carregarTexturas();

		prepararLoop();

		Gdx.input.setInputProcessor(this);
	}

	private void prepararLoop(){
		if (jogadorUmPrimeiro.nextBoolean()) {
			Primeiro = new Jogador("um");
			Segundo = new Jogador("dois");
```

```
            } else {
                  Primeiro = new Jogador("dois");
                  Segundo = new Jogador("um");
            }

            Primeiro.atribuirCasa(mapa.aldeiaJogador);
            Primeiro.atribuirExplorador(mapa.exploradorJogador);

            Segundo.atribuirCasa(mapa.aldeiaComputador);

      Segundo.atribuirExplorador(mapa.exploradorComputador);

            System.out.println("Base jog " + Primeiro.id + ": "
+ Primeiro.casa.x
                        + " " + Primeiro.casa.y + ". Base jog "
+ Segundo.id + ": "
                        + Segundo.casa.x + " " +
Segundo.casa.y);
            r.nivel = Dificuldade.FACIL;
            r.Atratividade(Primeiro,Segundo, true, mapa);
            r.Atratividade(Segundo,Primeiro, true, mapa);
      }

      private void executarLoop(){
            //Gdx.app.log("Executou loop", "podeAvancar =
"+podeAvancar);
            //while (!r.FIMDEJOGO && turno < 80) {
            if (podeAvancar){

                  if (estaForaDeCombate){
                        System.out.println("\n\nInicio do
turno: " + turno
                                    + "\n\n\nvez do jogadaor:
" + Primeiro.id);

                        Primeiro.Status();
                        r.Escolher(Primeiro, Segundo, mapa);
                        Primeiro.v.fimDeTurno(turno);
                        Primeiro.Status();


                        System.out.println("fim da vez do
jogador "+ Primeiro.id
                                    + "\n\n\nvez do jogador: "
+ Segundo.id);
```

```
                        //r.caminho.imprimir();

                        Segundo.Status();
                        r.Escolher(Segundo, Primeiro, mapa);
                        Segundo.v.fimDeTurno(turno);
                        Segundo.Status();

                        System.out.println("fim da vez do
jogador "+ Segundo.id
                                        + "\nFim do turno: " +
turno);

                        //r.caminho.imprimir();

                        turno++;
                        podeAvancar = false;
                  }else{

                  }

                  // mostrar coisas pra testar
                  /*
                  mapa.comidas.imprimir("Comidas");
                  mapa.madeiras.imprimir("Madeiras");
                  mapa.pedras.imprimir("Pedras");
                  */
                  //mostrarDonos();
            }
      }

      /*
      private void mostrarDonos(){
            System.out.println("--COMIDAS POSSUIDAS POR--");
            for (Ponto c : mapa.comidas.lista){
                  System.out.print("("+c.x+","+c.y+"):
"+c.tipo+", ");
            }
            System.out.println();
            System.out.println("-- --");
      }
      /**/

      // para mudar de turno apenas quando for possível
      private void avancar(){
```

```
            if (podeAvancar == false)
                  podeAvancar = true;
      }

      // desenho
      @Override
      public void render() {
            Gdx.gl.glClearColor(0.3f,0.3f,0.3f,1);
            Gdx.gl.glClear(GL10.GL_COLOR_BUFFER_BIT);

            executarLoop();
            desenhar();
      }

      private void desenhar(){
            batch.setProjectionMatrix(camera.combined);
            batch.begin();
                  desenharBlocos();
            batch.end();
      }

      private void desenharBlocos(){
            desenharChao();
            desenharObjetos();
      }

      /*
      private void texto(String txt, float x, float y){
            font.draw(batch, txt, x, y);
      }
      */

      private void desenharChao(){
            for (int x = 0; x < mapa.tx; x++){
                  for (int y = 0; y < mapa.ty; y++){
                        float xfinal = x * escala + deslocx;
                        float yfinal = y * escala + deslocy;
                        TextureRegion chao =
cenario_chao[mapa.chao[x][y]];
                        batch.draw(chao, xfinal, yfinal,
                                    CEL_TX * escala, CEL_TY *
escala);
                  }
            }
      }
```

```
	private void desenharObjetos(){
		// comidas
		for (Ponto o: mapa.comidas.lista){
			float xfinal = o.x * escala + deslocx;
			float yfinal = o.y * escala + deslocy;
			TextureRegion objeto = null;
			if (o.tipo == "UNICO"){
				objeto = cenario_objetos[5];
			}else{
				objeto = cenario_objetos[1];
				if (o.tipo.equals("um")){
					batch.draw(aura[1], xfinal,
yfinal,
							CEL_TX * escala,
CEL_TY * escala);
				}else if (o.tipo.equals("dois")){
					batch.draw(aura[2], xfinal,
yfinal,
							CEL_TX * escala,
CEL_TY * escala);
				}
			}
			batch.draw(objeto, xfinal, yfinal,
					CEL_TX * escala, CEL_TY *
escala);

			font.draw(batch, ""+o.qtde,
					xfinal + escala * 0.5f,
					yfinal + escala * 0.5f);
		}

		// madeiras
		for (Ponto o: mapa.madeiras.lista){
			float xfinal = o.x * escala + deslocx;
			float yfinal = o.y * escala + deslocy;
			TextureRegion objeto = null;
			if (o.tipo == "UNICO"){
				objeto = cenario_objetos[6];
			}else{
				objeto = cenario_objetos[2];
				if (o.tipo.equals("um")){
					batch.draw(aura[1], xfinal,
yfinal,
```

```
                                                CEL_TX * escala,
CEL_TY * escala);
                        }else if (o.tipo.equals("dois")){
                                batch.draw(aura[2], xfinal,
yfinal,
                                                CEL_TX * escala,
CEL_TY * escala);
                        }
                }
                batch.draw(objeto, xfinal, yfinal,
                                CEL_TX * escala, CEL_TY *
escala);
                font.draw(batch, ""+o.qtde,
                                xfinal + escala * 0.5f,
                                yfinal + escala * 0.5f);
        }

        // pedras
        for (Ponto o: mapa.pedras.lista){
                float xfinal = o.x * escala + deslocx;
                float yfinal = o.y * escala + deslocy;
                TextureRegion objeto = null;
                if (o.tipo == "UNICO"){
                        objeto = cenario_objetos[7];
                }else{
                        objeto = cenario_objetos[3];
                        if (o.tipo.equals("um")){
                                batch.draw(aura[1], xfinal,
yfinal,
                                                CEL_TX * escala,
CEL_TY * escala);
                        }else if (o.tipo.equals("dois")){
                                batch.draw(aura[2], xfinal,
yfinal,
                                                CEL_TX * escala,
CEL_TY * escala);
                        }
                }
                batch.draw(objeto, xfinal, yfinal,
                                CEL_TX * escala, CEL_TY *
escala);
                font.draw(batch, ""+o.qtde,
                                xfinal + escala * 0.5f,
                                yfinal + escala * 0.5f);
        }
```

```
                // aldeia do jogador 1
                float xfinal = mapa.casa.x * escala + deslocx;
                float yfinal = mapa.casa.y * escala + deslocy;
                batch.draw(cenario_objetos[8], xfinal, yfinal,
                                CEL_TX * escala, CEL_TY * escala);

                batch.draw(aura[1], xfinal, yfinal,
                                CEL_TX * escala, CEL_TY * escala);

                // aldeia do jogador 2
                xfinal = mapa.casa_inimigo.x * escala + deslocx;
                yfinal = mapa.casa_inimigo.y * escala + deslocy;
                batch.draw(cenario_objetos[9], xfinal, yfinal,
                                CEL_TX * escala, CEL_TY * escala);
                batch.draw(aura[2], xfinal, yfinal,
                                CEL_TX * escala, CEL_TY * escala);

                // explorador jogador 1
                xfinal = Primeiro.origem.x * escala + deslocx;
                yfinal = Primeiro.origem.y * escala + deslocy;
                batch.draw(cenario_objetos[10], xfinal, yfinal,
                                CEL_TX * escala, CEL_TY * escala);
                batch.draw(aura[1], xfinal, yfinal,
                                CEL_TX * escala, CEL_TY * escala);

                // explorador jogador 2
                xfinal = Segundo.origem.x * escala + deslocx;
                yfinal = Segundo.origem.y * escala + deslocy;
                batch.draw(cenario_objetos[11], xfinal, yfinal,
                                CEL_TX * escala, CEL_TY * escala);
                batch.draw(aura[2], xfinal, yfinal,
                                CEL_TX * escala, CEL_TY * escala);
        }

        private void ajeitarEscala(){

        ajeitarEscala(Gdx.graphics.getWidth(),Gdx.graphics.getHeig
ht());
        }

        private void ajeitarEscala(int w, int h){
                CAM_TX = CAM_SIZE;
                CAM_TY = (float) (CAM_SIZE * h/w);
```

```
		float pCam = CAM_TX / CAM_TY;
		float mCam = mapa.tx / mapa.ty;
		float dif = pCam - mCam;

		if (dif < 0f){
			escala = CAM_TY / mapa.ty;
			//escala = CAM_TX / mapa.tx;
		}else{
			escala = CAM_TX / mapa.tx;
			//escala = CAM_TY / mapa.ty;
		}
		escala = CAM_TY / mapa.ty;;
		//escala = 0.04f;
		//System.out.println("escala "+escala);
	}

	private void carregarTexturas(){
		TextureAtlas atlas = new TextureAtlas(
				Gdx.files.internal(CAMINHO_IMAGENS));

		aldeia = new TextureRegion[3];
		aldeia[0] = atlas.findRegion("aldeia-norte");
		aldeia[1] = atlas.findRegion("aldeia-centro");
		aldeia[2] = atlas.findRegion("aldeia-sul");

		explorador = new TextureRegion[3];
		explorador[0] = atlas.findRegion("explorador-
norte");
		explorador[1] = atlas.findRegion("explorador-
centro");
		explorador[2] = atlas.findRegion("explorador-sul");

		aura = new TextureRegion[3];
		aura[0] = atlas.findRegion("vazio");
		aura[1] = atlas.findRegion("aura-time-a");
		aura[2] = atlas.findRegion("aura-time-b");

		cenario_chao = new TextureRegion[6];
		cenario_chao[0] = atlas.findRegion("cenario-grama");
		cenario_chao[1] = atlas.findRegion("cenario-areia");
		cenario_chao[2] = atlas.findRegion("cenario-terra");
		cenario_chao[3] = atlas.findRegion("cenario-agua");
		cenario_chao[4] = atlas.findRegion("cenario-pedra");
		cenario_chao[5] = atlas.findRegion("cenario-
floresta");
```

```
            cenario_objetos = new TextureRegion[12];
            cenario_objetos[0] = atlas.findRegion("vazio");
            cenario_objetos[1] = atlas.findRegion("recurso-
fazenda");
            cenario_objetos[2] = atlas.findRegion("recurso-
floresta");
            cenario_objetos[3] = atlas.findRegion("recurso-
mina");
            cenario_objetos[4] = atlas.findRegion("construcao-
observatorio");
            cenario_objetos[5] = atlas.findRegion("objeto-
carne");
            cenario_objetos[6] = atlas.findRegion("objeto-
madeira");
            cenario_objetos[7] = atlas.findRegion("objeto-
pedra");

            cenario_objetos[8] = aldeia[tribo_a];
            cenario_objetos[9] = aldeia[tribo_b];
            cenario_objetos[10]= explorador[tribo_a];
            cenario_objetos[11]= explorador[tribo_b];
      }

      // funções extras
      @Override
      public void dispose() {
            batch.dispose();
            font.dispose();
      }

      @Override
      public void resize(int width, int height) {
            ajeitarEscala(width, height);
      }

      @Override
      public void pause() {
      }

      @Override
      public void resume() {
      }

      @Override
```

```
	public boolean keyDown(int keycode) {
		return false;
	}

	@Override
	public boolean keyUp(int keycode) {
		avancar();
		return false;
	}

	@Override
	public boolean keyTyped(char character) {
		return false;
	}

	@Override
	public boolean touchDown(int screenX, int screenY, int
pointer, int button) {
		return false;
	}

	@Override
	public boolean touchUp(int screenX, int screenY, int
pointer, int button) {
		avancar();
		return false;
	}

	@Override
	public boolean touchDragged(int screenX, int screenY, int
pointer) {
		return false;
	}

	@Override
	public boolean mouseMoved(int screenX, int screenY) {
		return false;
	}

	@Override
	public boolean scrolled(int amount) {
		return false;
	}
}
```

2. config.java

```
package br.ufla.americatribal.ia;

public class Config {
	public static final int RES_LAR = 700;
	public static final int RES_ALT = 700;
}
```

12 Anexo A

GAME DESIGN DO JOGO AMÉRICA TRIBAL

Fonte: LIMA (2013)

1 Conceito do Jogo

1.1 Premissa

No jogo o jogador comanda seu exército tribal através dos mapas no intuito de dominar os seus inimigos. Deverá fazer uma administração dos seus recursos e criar uma estratégia para conquistar o mapa e derrotar todos os seus inimigos.

1.2 Motivação do Jogador

O jogador deve explorar os mapas em busca de recursos para evoluir sua tribo e construir um exército para dominar o seu inimigo. O jogo termina quando um dos jogadores tem sua aldeia tomada pela tribo inimiga. Sendo vitorioso aquele que conquistar todas as aldeias do mapa. O jogador pode perder o jogo quando tem sua aldeia dominada por uma tribo inimiga e tem seu exército eliminado, ou senão através da desistência.

1.3 Diferencial

O jogo tem como protagonistas os povos indígenas da América, um tema pouco abordado no mercado de jogos. O tema possui um valor educacional, podendo assim ser utilizado como uma ferramenta educacional.

1.4 Público-alvo

O público-alvo do jogo são os jovens adultos, nascidos nas décadas de 1980 e 1990. Em especifico os jovens que tiveram um grande contato com os primeiros jogos de estratégia.

1.5 Gênero

O gênero escolhido para o desenvolvimento do jogo foi o de estratégia baseada em turnos, apresentando alguns elementos do RPG.

2 Projeto do Jogo

2.1 Tribos

Devido ao grande numero de tribos indígenas na América, as tribos serão agrupadas pela sua semelhança cultural. Assim as tribos foram divididas em três grupos, as tribos do norte, centro e sul da América. Mesmo nesta divisão existe uma grande diferença cultural dentre as tribos dentro de cada grupo, porém, a mérito de simplificação estas diferenças serão ignoradas.

- **Norte:** as tribos do norte serão representadas pelas culturas norte-americanas. Dando ênfase nas tribos Cherokee, Navarro e Apache.
- **Centro:** as tribos do centro serão representadas pelas culturas mesoamericanas. Dando ênfase nas tribos Astecas, Incas e Maias.
- **Sul:** as tribos do sul serão representadas pelas culturas sul-americanas. Dando ênfase nas tribos Guarani, Ianomâmis e Pataxós.

2.2 Mapas

Os mapas serão formados por um conjunto de quadrados. Ordenados de forma de grade, de altura e largura iguais. Os quadrados são classificados em quatro grupos distintos: terreno, bloqueio, interação primaria e interação secundária. Cada jogador pode ocupar apenas um quadrado por vez, e apenas um jogador pode ocupar um quadrado por vez.

2.2.1 Terreno

Os quadrados de terreno serão constituídos por terra, grama e areia. Os jogadores utilizaram este tipo de quadrado para a exploração do mapa. O quadrado de terreno é acessível se pelo menos um quadrado adjacente ortogonalmente também é acessível. Quando um quadrado de terreno apresenta uma entidade interativa ele é classificado como um quadrado de interação primária ou secundária.

2.2.2 Bloqueio

Os quadrados de bloqueio tem a finalidade de limitar a área de exploração do jogador. Eles serão representados por montanha, árvore e água. E eles não serão acessíveis de nenhuma direção.

a) Interação Primária

Os quadrados de interação primária são uma variação dos quadrados de terreno, porém possuem uma entidade interativa. Estas entidades são: construções e recursos ilimitados. Quando o jogador caminha para um quadrado de interação primária ele ativa o gatilho da entidade interativa, executando uma ação. Os quadrados de interação primária podem ser ativados mais de uma vez.

b) Interação Secundaria

Os quadrados de interação secundária são como os de interação primária, porém permitem uma única utilização. Suas entidades interativas são: unidades, recursos finitos e bônus temporários.

c) Recursos

Os recursos são a moeda do jogo. Eles são necessários para erguer novas construções, recrutar novas unidades, fazer melhorias nas construções. Eles podem ser obtidos através da dominação dos quadrados de interação primária, garantindo uma quantidade daquele recurso por turno enquanto dominar aquele quadrado. Ou senão através dos quadrados de interação secundária, recebendo pequenas quantidades de recurso.

- **Ilimitados:** os recursos ilimitados são fornecidos pelos quadrados de interação primária. A pedra será fornecido pelas minas, a madeira será fornecida pelas florestas e os alimentos pelas fazendas. Uma vez ativado o quadrado interativo por um jogador, ele passará a fornecer uma quantidade de seu recurso a cada turno àquele jogador, até que outro jogador faça uma nova ativação naquele quadrado. Os valores e tipos dos recursos fornecidos serão gerados aleatoriamente na criação do mapa.
- **Finitos: o**s recursos finitos serão fornecidos pelos quadrados de interação secundária. Quando ativados pela primeira vez os quadrados fornecem uma pequena quantidade de um recurso e se tornam um quadrado de terreno normal. Os valores e tipos de recursos serão gerados aleatoriamente na criação do mapa.

d) Construções

As construções serão encontradas nos quadrados de interação primária. Existem três tipos de construções que podem ser encontradas pelo mapa, as bases dos jogadores, os observatórios, e as aldeias.

Cada jogador possui uma base no mapa, e não existem bases vazias pelo mapa. O jogador pode interagir quantas vezes forem necessárias com sua base. E quando ele interage com a base do inimigo ele irá entrar em modo de combate com aquele jogador. Se vitorioso o jogador passa a controlar a base de seu inimigo.

Os observatórios tem a função de revelar mapa quando ativados, podendo ser utilizado por qualquer jogador. Ele revela um numero aleatório de quadrados a sua volta, determinado na criação do mapa.

As aldeias são pequenas construções onde o jogador pode recrutar novas unidades, sendo renovada a cada sete turnos. Qualquer jogador pode recrutar unidades, sendo limitado apenas pela quantidade de unidades disponíveis na aldeia.

e) Unidades

As unidades serão encontradas nos quadrados se interação secundária. Existindo duas variações no tipo de unidade, ela pode ser aliada ou agressiva. Quando o jogador interagir com uma unidade aliada ela irá dar ao jogador a oportunidade de entrar para o grupo do jogador. E quando o jogador interage com uma unidade agressiva ele irá entrar em modo de combate com aquela unidade.

2.3 Aldeia e Melhorias

Cada jogador inicia o jogo com uma base sobre o seu controle. Quando o jogador interage com a sua base ele irá sair do modo de exploração e irá para o modo de administração de sua base. Lá ele poderá criar novas construções para a sua aldeia, fazer melhorias para as construções existentes, e recrutar novas unidades.

a) Aldeia

A aldeia é a principal construção da base. Ela é um pré-requisito para quase todas as outras construções da base, e a cada nível fornece uma quantidade maior de recursos por turno. E também possibilita a pesquisa de novas tecnologias para o aumento da produção de recursos.

- **Aldeia I → Custo:** 0C 0M 0P
  Bônus: +2C +2M +2P /dia
  Pré-requisito: Nenhum
- **Aldeia II→Custo:** 0C 10M 10P
  Bônus: +4C +3M +3P /dia
  Pré-requisito: Aldeia I
- **Aldeia III→Custo:** 0C 30M 30P
  Bônus: +6C +4M +4P /dia
  Pré-requisito: Aldeia II
- **Aldeia IV→Custo:** 0C 60M 60P
  Bônus: +8C +5M +5P /dia
  +1 Unidade Lendária /7dias
  Pré-requisito: Aldeia III

b) Totem

O totem é a construção que fornece proteção mística (DefM) para as unidades presentes na aldeia. Sendo também um pré-requisito para a construção do templo.

- **Totem I→ Custo:** 0C 10M 0P
  Bônus: +5% DefM
  Pré-requisito: Arena II
- **Totem II→Custo:** 0C 10M 5P
  Bônus: +15% DefM
  Pré-requisito: Totem I, Templo
- **Totem III→Custo:** 40C 10M 0P
  Bônus: +25% DefM
  Pré-requisito: Totem II, Aldeia III

c) Defesa

As construções de defesa tem o intuito de proteger a base. A cada nível ela é capaz de absorver certa porcentagem do dano físico (Def) causado às unidades presentes na aldeia.

- **Defesa I→ Custo:** 0C 10M 0P
  Bônus: +10% Def
  Pré-requisito: Arena I
- **Defesa II→Custo:** 0C 0M 15P
  Bônus: +15% Def
  Pré-requisito: Defesa I, Aldeia II, Tec. Madeira I
- **Defesa III→Custo:** 15C 10M 0P

Bônus: +25% Def
Pré-requisito: Defesa II, Totem III

d) Arena

A arena irá produzir a unidade básica, a unidade de longo alcance e a unidade animal. No seu primeiro nível ela irá produzir apenas unidades básicas, sendo que cada tribo possui uma unidade básica diferente. No seu segundo nível ela irá produzir as unidades de longo alcance e terá um aumento na produção de unidades básicas por ciclo.

- **Arena I → Custo:** 5C 5M 5P
  Bônus: +5 Unidades Básicas /7dias
  Pré-requisito: Arena I
- **Arena II→ Custo:** 15C 15M 15P
  Bônus: +7 Unidades Básicas /7dias
  +5 Unidades Longo Alcance /7dias
  Pré-requisito: Arena I, Aldeia II
- **Arena III→Custo:** 15C 10M 0P
  Bônus: +9 Unidades Básicas /7dias
  +7 Unidades Longo Alcance /7dias
  +4 Unidades Animal /7dias
  Pré-requisito: Arena II

e) Templo

O templo irá produzir as unidades místicas. E quando for construído irá fornecer a um bônus de proteção mística para a base.

- **Templo → Custo:** 20C 20M 20P
  Bônus: +3 Unidades Místicas /7dias
  Pré-requisito: Totem I

f) Tecnologia em Carne

Quando pesquisada a tecnologia de carne irá permitir que mais aldeões trabalhem na extração de comida da aldeia. Alocando mais aldeões nas fazendas dominadas pela tribo.

- **Tec. Carne I → Custo:** 5C 0M 0P
  Bônus: +10% C /dia
  Pré-requisito: Aldeia I
- **Tec. Carne II → Custo:** 10C 0M 0P
  Bônus: +20% C /dia

Pré-requisito: Tec. Carne I, Aldeia II

- **Tec. Carne III →** **Custo:** 15C 0M 0P
  Bônus: +30% C /dia
  Pré-requisito: Tec. Carne II, Aldeia III

g) Tecnologia em Madeira

Quando pesquisada a tecnologia de madeira irá permitir que mais aldeões trabalhem na extração de madeira da aldeia. Alocando mais aldeões nas florestas dominadas pela tribo.

- **Tec. Madeira I →** **Custo:** 0C 5M 0P
  Bônus: +10% M /dia
  Pré-requisito: Aldeia I
- **Tec. Madeira II→** **Custo:** 0C 10M 0P
  Bônus: +20% M /dia
  Pré-requisito: Tec. Madeira I, Aldeia II
- **Tec. Madeira III→** **Custo:** 0C 15M 0P
  Bônus: +30% M /dia
  Pré-requisito: Tec. Madeira II, Aldeia III

h) Tecnologia em Pedra

Quando pesquisada a tecnologia de pedra irá permitir que mais aldeões trabalhem na extração de pedra da aldeia. Alocando mais aldeões nas minas dominadas pela tribo.

- **Tec. Pedra I →** **Custo:** 0C 0M 5P
  Bônus: +10% P /dia
  Pré-requisito: Aldeia I
- **Tec. Pedra II →** **Custo:** 0C 0M 10P
  Bônus: +20% P /dia
  Pré-requisito: Tec. Pedra I, Aldeia II,
- **Tec. Pedra III →** **Custo:** 0C 0M 15P
  Bônus: +30% P /dia
  Pré-requisito: Tec. Pedra II, Aldeia III

2.4 Personagens

O jogo iniciará com cada jogador comandando um explorador e um pequeno grupo de unidades, variando o conjunto de unidades para cada tribo. Cada jogador possui apenas um explorador, e quando uma batalhe e perdida e o desbravador do jogador é eliminado ele reaparece na aldeia do jogador. Novas

unidades só poderão ser adicionadas ao grupo do desbravador quando o mesmo estiver presente no quadrado de uma de suas aldeias.

Cada unidade possui sete atributos básicos: vitalidade, ataque, defesa física, defesa mística, iniciativa, destreza e crítico.

- **Vitalidade (Vit):** determina a quantidade de pontos de vida da unidade. Quando sua vitalidade chega a zero a unidade morre e é retirada do grupo do desbravador.
- **Ataque (Atq):** composto de dois valores, um mínimo e um máximo. Quando um ataque é realizado, um valor aleatório é gerado entre o valor mínimo e o máximo do ataque para determinar a quantidade de pontos de vida será subtraído da unidade atacada.
- **Defesa física (Def):** determina a taxa de absorção do dano físico. Reduz o dano em uma porcentagem pré-definida.
- **Defesa mística (DefM):** determina a taxa de absorção do dano místico. Reduz o dano em uma porcentagem pré-definida.
- **Iniciativa (Ini):** determina a ordem em que as unidades irão fazer suas ações na batalha. Cada unidade possui um valor único entre 0 e 1 como iniciativa. No inicio da cada turno um numero aleatório entre 0 e o numero de unidades em batalha é atribuído a cada unidade. O valor da iniciativa é utilizado como critério de desempate.
- **Destreza (Des):** probabilidade de acerto do ataque. Definido por uma porcentagem. Toda vez que um ataque é realizado um numero aleatório entre 0 e 100 é gerado para avaliar se o ataque atingiu o alvo. Em caso negativo nenhum dano é causado ao alvo, e nenhum outro teste é realizado.
- **Crítico (Crit):** verifica se o ataque realizado irá realizar um dano critico ou um dano comum. Ela é definida por uma porcentagem. Toda vez que é confirmado um ataque é gerado um numero aleatório entre 0 e 100 para verificar o critico. Se confirmado o dano é aumentado em 50%.

a) Unidade Básica

As unidades básicas são os guerreiros mais comuns de cada tribo. Eles são a linha de frente em qualquer batalha. São as unidades mais baratas e as que são produzidas em maior quantidade a cada ciclo.

Nome	Vit	Atq	Def	DefM	Ini	Des	Crit

Tribo do Norte: Guerreiro com Machado	20	5	15	0	0,21	90	5
Tribo do Centro: Guerreiro com Honda	20	5	15	0	0,23	90	5
Tribo do Sul: Guerreiro com Lança	20	5	15	0	0,22	90	5

b) Unidade de Longo Alcance

As unidades de longo alcance são os caçadores de cada tribo. Eles possuem armas para atacar os seus inimigos a distancia.É a segunda unidade mais barata e com mais disponibilidade de recrutamento de cada tribo.

Nome	**Vit**	**Atq**	**Def**	**DefM**	**Ini**	**Des**	**Crit**
Tribo do Norte: Guerreiro com *Tomahawk*	25	15	15	25	0,63	60	50
Tribo do Centro: Guerreiro com Boleadeira	25	15	15	25	0,61	60	50
Tribo do Sul: Arqueiro	25	15	15	25	0,62	60	50

c) Unidade Animal

A unidade animal é uma fera domesticada pela tribo para auxiliar nas batalhas. É uma unidade com um custo relativamente baixo e de grande poder em batalha, porém poucas são produzidas por ciclo para o recrutamento.

Nome	**Vit**	**Atq**	**Def**	**DefM**	**Ini**	**Des**	**Crit**
Tribo do Norte: Búfalo	50	15	25	10	0,31	80	10
Tribo do Centro: Jaguar	50	15	25	10	0,30	80	10

Tribo do Sul: Lobo Guará	50	15	25	10	0,32	80	10

d) Unidade Mística

As unidades místicas são representadas pelos feiticeiros de cada tribo. São as unidades mais próximas dos deuses que as tribos veneram. E possuem poderes místicos para atacar os seus inimigos. As unidades místicas possuem um alto dano, porém são muito frágeis e não resistem a muitos ataques.

Nome	**Vit**	**Atq**	**Def**	**DefM**	**Ini**	**Des**	**Crit**
Tribo do Norte: Xamã	10	25	5	50	0,13	70	40
Tribo do Centro: Sacerdote	10	25	5	50	0,11	70	40
Tribo do Sul: Pajé	10	25	5	50	0,12	70	40

a) Unidade Mítica

A unidade mítica é a última unidade possível de ser recrutada, e é a mais forte de cada tribo. Elas são as figuras veneradas por cada uma das tribos, consideradas deuses. Elas são as unidades mais caras para se recrutar e apenas uma é disponibilizada por turno. Porém cada uma possui um grande poder, capaz de destruir grandes grupos inimigos.

Nome	**Vit**	**Atq**	**Def**	**DefM**	**Ini**	**Des**	**Crit**
Tribo do Norte: *Skin Walker* (Troca-peles)	90	35	55	85	0,93	75	25
Tribo do Centro: Quetzalcatl	90	35	55	85	0,91	75	25
Tribo do Sul: Tupã/Garaci	90	35	55	85	0,92	75	25

13 Anexo B

Exemplo do arquivo de entrada de mapa desenvolvido no projeto de Jeferson Barrile Tomazella, usado para testes do Algoritmo de Busca neste projeto.

Nível America Tribal

Dimensões:30,30

Blocos Chão:

5, 5, 0, 0, 2, 2, 1, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,

5, 0, 0, 2, 2, 1, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 2, 2, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

2, 2, 2, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 5, 0, 0, 0, 0, 0, 0, 0, 0,

2, 1, 1, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 5, 5, 0, 0, 0, 0, 0, 0, 0, 4,

3, 3, 3, 3, 1, 1, 1, 1, 1, 4, 4, 1, 1, 1, 1, 3, 3, 5, 5, 5, 5, 0, 0, 0, 0, 0, 0, 0, 4, 4,

3, 1, 1, 1, 1, 1, 1, 1, 4, 4, 1, 1, 1, 1, 3, 3, 5, 5, 5, 5, 0, 0, 5, 0, 0, 0, 0, 4, 4, 3,

1, 1, 1, 1, 1, 1, 1, 4, 4, 1, 1, 1, 1, 1, 3, 5, 5, 5, 5, 0, 0, 0, 0, 0, 0, 0, 4, 4, 3, 3,

1, 1, 1, 1, 1, 1, 4, 4, 1, 1, 1, 1, 1, 1, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 3, 3, 3,

1, 1, 1, 1, 1, 4, 4, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 3, 3, 3, 0, 0, 0, 0, 0, 0,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 5, 3, 3, 3, 3, 0, 0, 0, 0, 0,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 4, 4, 4, 0, 0, 0, 5, 5, 3, 3, 5, 0, 0, 0, 0, 0, 0,

1, 1, 1, 1, 1, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 5, 5, 5, 5, 0, 0, 0, 0, 0, 4,

1, 1, 1, 1, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4,

1, 1, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 2,

1, 4, 4, 4, 4, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 5, 5, 4, 4, 4, 2, 2,

4, 4, 4, 5, 5, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 5, 5, 4, 4, 4, 2, 2, 2,

4, 4, 5, 5, 5, 5, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 5, 5, 5, 5, 4, 4, 4, 2, 2, 2, 2,

4, 5, 5, 5, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 5, 5, 4, 4, 4, 2, 2, 2, 2, 2,

5, 5, 5, 5, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 5, 5, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2,

5, 5, 5, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 5, 5, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2,

5, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 4, 4, 4, 4, 2, 2, 2, 2, 2, 2, 4, 4, 2,

0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 2, 2,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 2, 2, 2,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 2, 2, 2, 2, 2, 2, 2,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2,

Blocos Objetos:

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 0, 0,

0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 6, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 5, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 6, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
7, 0, 0, 0, 0,
0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0,
0, 3, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0,
0, 0, 0, 0, 5,
0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 3, 0, 0, 0, 6, 0, 0, 0, 0, 2,
0, 0, 0, 7, 0,
0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 7, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 6, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 6, 0,

0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

Fim

14 Anexo C

Exemplo do arquivo de entrada de mapa desenvolvido no projeto de Jeferson Barrile Tomazella, usado para testes do Algoritmo de Busca neste projeto.

Nível America Tribal

Dimensões:13,7

Blocos Chão:

0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 2, 2, 2,
0, 0, 0, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0,
0, 0, 0, 4, 0, 1, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5,

Blocos Objetos:

5, 0, 2, 0, 2, 0, 0, 0, 0, 1, 0, 0, 0,
5, 0, 6, 0, 0, 0, 0, 0, 0, 0, 11, 9, 0,
5, 0, 6, 0, 6, 0, 0, 0, 0, 0, 0, 0, 5,
5, 7, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 5,
5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5,
0, 8, 10, 0, 0, 0, 3, 0, 0, 6, 0, 0, 7,

0, 0, 0, 0, 1, 0, 0, 0, 0, 6, 0, 0, 0,
Fim