



JEFERSON BARRILE TOMAZELLA

**DESENVOLVIMENTO DE UM EDITOR DE
NÍVEIS PARA UM JOGO DE ESTRATÉGIA,
BASEADO EM TURNOS PARA ANDROID**

LAVRAS - MG

2014

JEFERSON BARRILE TOMAZELLA

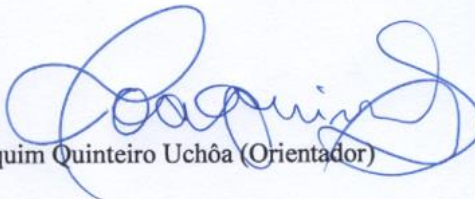
**DESENVOLVIMENTO DE UM EDITOR DE
NÍVEIS PARA UM JOGO DE ESTRATÉGIA,
BASEADO EM TURNOS PARA ANDROID**

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

Este trabalho apresenta um editor de níveis para um jogo de estratégia baseado em turnos para a plataforma Android, assim como as etapas para a criação do editor, e os testes realizados após a implementação. Para a criação do editor foram utilizados como base dois editores de nível já consolidados no mercado de jogos, dos quais puderam ser observados elementos comuns de interface e usabilidade, que foram utilizados neste projeto. Além da função de criar, importar e exportar níveis, o editor possui um módulo de verificação de eficácia, para garantir que o nível seja equilibrado em termos de recursos, e também para garantir que seja possível vencer no nível criado. Após a implementação foram feitos testes de usabilidade, para comparar a criação de um nível em um editor de níveis, e em um editor de texto.

Palavras-chave: Editor de Nível, Jogo de estratégia baseado em turnos

LISTA DE TABELAS

TABELA 1 TESTES DE USABILIDADE.....	48
-------------------------------------	----

LISTA DE FIGURAS

FIGURA 1 ARQUITETURA DO ANDROID.....	12
FIGURA 2 NÍVEL EM FORMATO DE TEXTO.....	15
FIGURA 3 NÍVEL EM UM EDITOR	15
FIGURA 4 AMBIENTE DO RPG MAKER VX ACE.....	21
FIGURA 5 AMBIENTE DO HEROES OF MIGHT AND MAGIC® III MAP EDITOR	22
FIGURA 6 ELEMENTOS DA INTERFACE DO EDITOR DE NÍVEIS	24
FIGURA 7 ELEMENTOS DA CAMADA INFERIOR DO CENÁRIO.....	24
FIGURA 8 ELEMENTOS DA CAMADA SUPERIOR DO CENÁRIO	25
FIGURA 9 EXEMPLO DE CENÁRIO.....	26
FIGURA 10 REPRESENTAÇÃO DE UM NÍVEL EM UM ARQUIVO TEXTO	27
FIGURA 11 REPRESENTAÇÃO DE UM NÍVEL NO EDITOR	27
FIGURA 12 INTERFACE DE CRIAÇÃO DE PROJETOS LIBGDX.....	31
FIGURA 13 DIAGRAMA DE CLASSES	34
FIGURA 14 NÍVEL PARA TESTES DE USABILIDADE	41
FIGURA 15 EDITOR FINALIZADO	43
FIGURA 16 ÁREA DO CENÁRIO.....	44
FIGURA 17 ÁREA DOS BLOCOS - CAMADA <i>CHÃO</i>	45
FIGURA 18 ÁREA DOS BLOCOS - CAMADA <i>OBJETOS</i>	45
FIGURA 19 ÁREA DAS FERRAMENTAS.....	47

SUMÁRIO

1	INTRODUÇÃO	8
1.1	OBJETIVOS.....	10
2	REFERENCIAL TEÓRICO	11
2.1	ANDROID.....	11
2.2	JOGOS	12
2.2.1	JOGOS DE ESTRATÉGIA BASEADOS EM TURNOS.....	12
2.3	NÍVEL EM UM JOGO	13
2.4	EDITOR DE NÍVEIS.....	14
2.5	<i>FRAMEWORK</i> LIBGDX.....	16
2.6	AMÉRICA TRIBAL	16
2.6.1	TRIBOS	17
2.6.2	MECÂNICA DO JOGO	17
2.7	ALGORITMO A*	18
3	METODOLOGIA	20
3.1	PROJETO DO EDITOR	20
3.1.1	ANÁLISE DE EDITORES DO MERCADO DE JOGOS	20
3.1.2	PROJETO DE ELEMENTOS DE INTERFACE	23
3.1.3	PROJETO TÉCNICO	24
3.1.3.1	ESPECIFICAÇÃO DE UM ARQUIVO DE NÍVEL	26
3.1.4	PLATAFORMA ALVO	28
3.1.5	VERIFICAÇÃO DE EFICÁCIA	28
3.2	MATERIAIS E RECURSOS TECNOLÓGICOS	29
3.3	UTILIZAÇÃO DA LIBGDX.....	30
3.3.1	DIAGRAMA DE CLASSES.....	32

3.3.1.1	DESCRIÇÃO DAS CLASSES	33
3.3.1.1.1	PACOTE “BR.UFLA.AMERICATRIBAL.EDITOR.AESTRELA”	33
3.3.1.1.1.1	CLASSE “AESTRELA.JAVA”	35
3.3.1.1.1.2	CLASSE “PONTO.JAVA”	35
3.3.1.1.1.3	CLASSE “LISTA.JAVA”	36
3.3.1.1.2	PACOTE “BR.UFLA.AMERICATRIBAL.EDITOR”	36
3.3.1.1.2.1	CLASSE “EDITOR.JAVA”	36
3.3.1.1.2.2	CLASSE “TELAPRINCIPAL.JAVA”	37
3.3.1.1.2.3	CLASSE “DESENHISTADATELAPRINCIPAL.JAVA”	38
3.3.1.1.2.4	CLASSE “MAPA.JAVA”	39
3.3.1.1.2.5	CLASSE “CONF.JAVA”	40
3.4	TESTES	40
3.4.1	TESTES DE FUNCIONALIDADE	40
3.4.2	TESTES DE USABILIDADE	41
4	RESULTADOS E DISCUSSÃO	43
4.1	RESULTADOS	43
4.1.1	EDITOR DE NÍVEIS	43
4.1.1.1	ÁREA DO CENÁRIO	43
4.1.1.2	ÁREA DOS BLOCOS	44
4.1.1.3	ÁREA DAS FERRAMENTAS	45
4.1.1.4	MENSAGENS DO EDITOR	46
4.1.1.5	NÍVEIS EXTERNOS	46
4.1.1.6	TESTES DE USABILIDADE	47
4.2	DISCUSSÃO	48
5	CONCLUSÃO	51
6	REFERÊNCIAS BIBLIOGRÁFICAS	53
7	APÊNDICE A	55

8	APÊNDICE B.....	68
9	ANEXO A	120

1 Introdução

O Android^{TM1} é um sistema operacional para dispositivos móveis amplamente utilizado nesse nicho de mercado, e o impulso para a criação de *software* para esse sistema também é grande, pois os aplicativos podem ser codificados em Java^{TM2}, uma linguagem *open source* muito utilizada. A criação de jogos para essa plataforma também é ampla, pois aliado ao crescente mercado de jogos para dispositivos móveis, as aplicações criadas para esta plataforma funcionam são independentes de dispositivo, pois a aplicação é feita para o sistema Android, e não para um dispositivo específico (ZECHNER, 2011).

Por outro lado, o mercado de jogos, que teve início durante a década de 1970, vem crescendo e se consolidando no mercado mundial desde então. Esse mercado passou por alguns períodos ruins, mas, principalmente com o crescente uso da internet, continua se expandindo cada vez mais. O Brasil ainda é jovem na área da criação de jogos, mas tem grande potencial para se tornar um gigante no ramo (LOBO, VERDI & ELIAS, 2012).

Este projeto faz parte de um projeto maior, que é a construção de um jogo de estratégia baseado em turnos para Android. O projeto foi dividido em três módulos: interface³, inteligência artificial e editor de níveis. O módulo de inteligência artificial foi criado por Renan Faraco Teixeira, e o módulo de interface foi criado por Marlon Jonas de Oliveira Lima, ambos estudantes da Universidade Federal de Lavras (UFLA). Neste trabalho foi criado o módulo do editor de níveis para tal jogo.

Um editor de níveis se assemelha muito a uma IDE (*Integrated Development Environment*). Ambos facilitam a criação e gerenciamento de

¹ O Android (<http://www.android.com/>) é um sistema operacional, de código livre, para dispositivos móveis, desenvolvido pela empresa Google Inc.

² http://www.java.com/pt_BR/

³ A descrição do módulo interface pode ser encontrado em LIMA (2013)

projetos, a ligação entre os elementos dele, e o encapsulamento do produto. Um editor de níveis geralmente é associado a um jogo específico, mas pode integrar um motor de jogos, ou mesmo ser genérico. Alguns exemplos comerciais consolidados no mercado são o *Valve Hammer Editor*⁴, que é utilizado para criar jogos que utilizam o motor *Goldsource*⁵ (como por exemplo o *Counter Strike*⁶ e o *Half Life*⁷), e o Unity⁸, que além de criar níveis, é um ambiente de desenvolvimento de jogos. O editor criado neste projeto será específico para um jogo de estratégia baseado em turnos para Android.

O jogo para qual o editor irá criar os níveis irá apresentar três tribos indígenas fictícias das Américas baseadas nos índios da América Central, do Norte e do Sul. O jogador irá controlar uma dessas tribos, gerenciando os recursos disponíveis para derrotar uma tribo inimiga.

A criação de um nível para um jogo não necessita de um editor de níveis especializado, o mapa pode ser criado diretamente em um arquivo texto, desde que seja gravado em um formato que o jogo consiga decodificar. Porém, é muito trabalhoso e frustrante criar um nível desta maneira, pois é mais difícil visualizar erros e inconsistências usando apenas números ou símbolos para representar os objetos. Utilizando-se de um editor de níveis é possível visualizar o nível exatamente como será visto dentro do jogo, tornando a criação muito mais intuitiva e consistente. Criar um nível visualmente também facilita o processo de edição, e o torna muito mais rápido, e deste modo também é possível observar possíveis falhas mais facilmente (PEMMARAJU, 2012).

⁴ https://developer.valvesoftware.com/wiki/Valve_Hammer_Editor

⁵ <https://developer.valvesoftware.com/wiki/Goldsource>

⁶ <http://store.steampowered.com/app/10>

⁷ <http://store.steampowered.com/app/70>

⁸ <http://unity3d.com/pt>

Para auxiliar na criação do editor foi utilizado o *framework* libGDX⁹, uma ferramenta de código livre multiplataforma que auxilia no gerenciamento de imagens, sons, dispositivos de entrada e outras funções, para projetos em Java.

1.1 Objetivos

O objetivo geral deste trabalho foi o desenvolvimento de um editor de níveis para um jogo de estratégia baseado em turnos para Android. Uma série de objetivos específicos foram definidos para a realização deste objetivo, que são:

1. Criação do projeto do editor. Para tal foram analisados dois editores de nível existentes no mercado de jogos, e criados alguns requisitos:
 - a. O editor deve apresentar os níveis visualmente, da mesma maneira que serão apresentados durante o jogo.
 - b. Também será necessário que o editor possibilite a visualização de uma área maior que a vista durante o jogo, para facilitar a manutenção do nível.
 - c. O editor deverá exportar os níveis criados para um arquivo externo.
 - d. Deverá ser possível importar níveis de arquivos externos.
2. Implementação do código do editor, seguindo as especificações criadas no projeto do mesmo.
3. Realização de testes da usabilidade do editor, medindo a proporção de tempo utilizada para a criação de um nível com o auxílio de um editor, para a criação de um nível utilizando um editor de texto.

⁹ <http://libgdx.badlogicgames.com/>

2 Referencial teórico

2.1 Android

Segundo Lee (2011), o Android é sistema operacional baseado em uma versão modificada do Linux para dispositivos móveis. A maior parte do código fonte do sistema é livre (*open source*), possibilitando que diversas empresas possam desenvolver seu *hardware* e adaptar o Android para tal. Uma grande vantagem deste sistema é que os desenvolvedores podem se focar em produzir aplicações para Android, e não para um dispositivo específico. Além disso, elas são codificadas em Java, uma linguagem bastante consolidada no mercado.

O Android tem sua arquitetura separada em quatro partes, como mostrado na Figura 1. São elas: 1) Kernel do Linux (*LINUX KERNEL*), que gerencia os módulos mais básicos, deixando o sistema bastante robusto; 2) Bibliotecas (*LIBRARIES*) e Execução (*ANDROID RUNTIME*), que são, respectivamente, as bibliotecas básicas de fontes, gráfico, som, entre outras e, a máquina virtual do Android, que é uma modificação da máquina virtual do Java; 3) Quadro de Aplicações (*APPLICATION FRAMEWORK*), que são os gerenciadores de mais alto nível, como o gerenciador de pacotes e o de janelas; 4) Aplicações (*APPLICATIONS*), que são as aplicações dos usuários e de mais alto nível, como o *Browser* de internet, gerenciador de contatos (para *smartphone*), entre outros (LEE, 2011; ZECHNER, 2011).

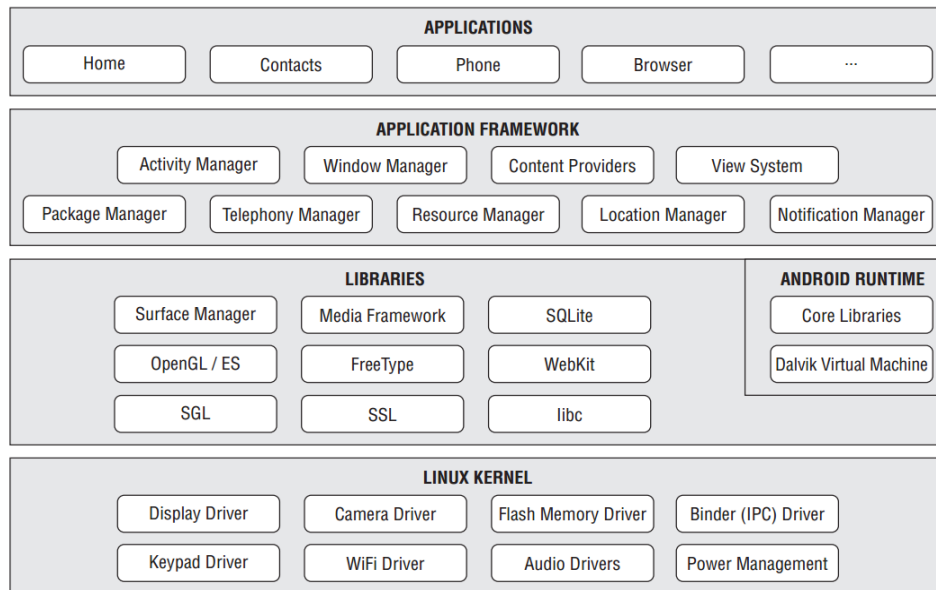


Figura 1 Arquitetura do Android
 Fonte (LEE, 2011)

2.2 Jogos

Koster (2005) apresenta diversas definições sobre jogos. Unindo essas definições pode-se chegar à conclusão que jogos são ferramentas de aprendizado baseadas em regras, simulando situações do mundo real, contendo desafios e premiações, com o objetivo de proporcionar entretenimento para quem joga.

2.2.1 Jogos de estratégia baseados em turnos

Dentre os diversos gêneros de jogos, os de estratégia são os que menos dependem de sorte, e mais da habilidade dos jogadores, que devem gerenciar os recursos oferecidos pelo jogo para construir seus exércitos e enfrentar os outros jogadores. Para vencer nesse tipo de jogo, geralmente é necessário exterminar o

jogador rival (ou jogadores rivais), e o jogador que gerenciar melhor seus recursos e unidades geralmente vence. Em um jogo de estratégia baseado em turnos cada jogador age em sua vez (turno), separadamente dos outros. Em seu turno, o jogador faz uma quantidade limitada de opções, geralmente movendo suas unidades e criando novas. Após acabar o turno, começa o do próximo jogador, e após todos os jogadores acabarem seus turnos, o ciclo é reiniciado e acontece mais um turno para cada jogador, e isso é repetido até algum jogador vencer. O xadrez é um consolidado exemplo de um jogo de estratégia baseado em turnos, onde cada jogador tem como recurso dezesseis peças, cada qual com sua peculiaridade, e pode mover apenas uma peça por vez durante seu turno, vencendo o jogador que abater a peça “rei” do adversário (BATES, 2004).

2.3 Nível em um jogo

No exemplo do xadrez, um nível é o próprio tabuleiro, com as dezesseis peças de cada jogador posicionadas seguindo um conjunto de regras pré-estabelecidas. Se as torres fossem trocadas por bispos e vice-versa, poderíamos considerar essa disposição das peças como outro nível. Ao retirar todos os peões do jogo, entre diversas outras possibilidades, também teríamos outro nível no xadrez.

Um nível é uma seção do jogo, uma junção do cenário com os personagens, suas posições e ações, é onde o jogo acontece realmente, e completar um nível geralmente leva ao próximo, até que o último seja completado, e o jogo é finalizado.

Em um jogo eletrônico de estratégia um nível pode ser representado por três camadas: a) Camada do cenário: onde é representado o mundo do jogo, com sua geografia, como rios e florestas; b) Camada dos objetos: onde são colocadas as unidades dos jogadores, os recursos coletáveis entre outras coisas, e; c)

Eventos: uma camada abstrata, onde são programados os acontecimentos, tais como receber unidades extra, caso o jogador chegue a um determinado local com as suas próprias unidades.

2.4 Editor de níveis

Segundo Rouse (2005), um editor de níveis é uma ferramenta essencial para um jogo que possua mais de um nível. Um editor de níveis mostra o cenário como será visto para o jogador durante o jogo. Ele também mostra informações adicionais, como por exemplo, um caminho que as unidades controladas pelo computador devem seguir. Um editor geralmente é específico para um jogo, feito para facilitar a criação de níveis pelo projetista, ou *level designer*, o qual transpõe a ideia criada, geralmente em papel, do nível para o jogo, utilizando o editor.

A Figura 2 mostra a representação de dois objetos de um nível em formato de texto, que é o modo como tal nível será salvo em disco. É uma maneira eficiente, porém pouco intuitiva e de difícil manutenção ou mesmo criação. Contrastando com esta, a Figura 3 mostra uma parte do mesmo nível, visualizado em um editor de níveis. A visualização é muito semelhante a que será apresentada ao jogador durante o próprio jogo, sendo muito mais fácil encontrar erros e inconsistências no nível, do que na versão texto do mesmo. Com este contraste, Pemmaraju (2012) aponta as vantagens oferecidas por um editor de níveis, em relação a não usar um, que são a praticidade, rapidez e facilidade da criação, visualização, edição e manutenção de um nível.

```

01 Object
02     Texture rocks4
03     Position 78 540
04     Rotation 0
05     Scale 1 1
06     Color 255 255 255 255
07     ScrollSpeed 1 1
08     CustomProperties
09     End
10 End
11 Object
12     Texture grass1
13     Position 60 450
14     Rotation 0
15     Scale 1 1
16     Color 255 255 255 255
17     ScrollSpeed 1 1
18     CustomProperties
19     End
20 End

```

Figura 2 Nível em formato de texto
Fonte (PEMMARAJU, 2012)

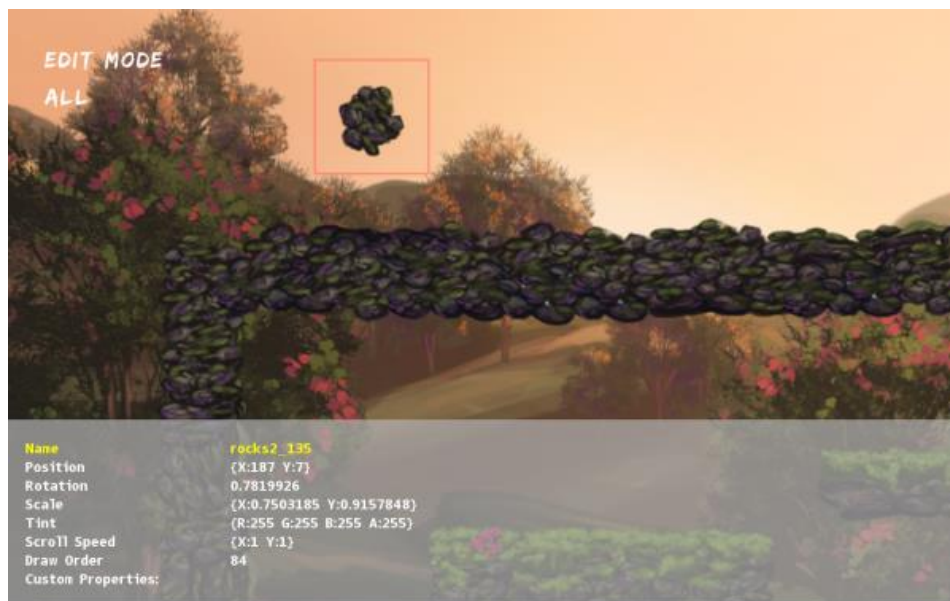


Figura 3 Nível em um editor
Fonte (PEMMARAJU, 2012)

2.5 Framework libGDX

O libGDX é um *framework* produzido pela empresa Badlogic Games¹⁰, para auxiliar no desenvolvimento de jogos eletrônicos em Java. É uma ferramenta multiplataforma onde o código é feito apenas uma vez, e pode ser exportado para diversas plataformas diferentes, como *desktop*, Android e HTML5.

Este *framework* é apenas uma ferramenta para facilitar a criação de jogos, pois ele possui diversas API que gerenciam os códigos em baixo nível, como sons, músicas, figuras e texto, álgebra e geometria, entre outros. O código final é feito em Java, aproveitando as funções oferecidas pela ferramenta para agilizar a criação de jogos ou outras aplicações gráficas.

Uma aplicação em libGDX é subdivida em dois ou mais projetos: projeto base e projeto Android, que são obrigatórios, e projetos de outras plataformas, que são opcionais. O projeto base contém todo o código da aplicação. O projeto Android contém o código para a exportação da aplicação para uma plataforma Android, e todos os recursos, como imagens e sons, são armazenados neste projeto. Os demais projetos apenas contém o código para a exportação da aplicação para a devida plataforma.

2.6 América Tribal

América Tribal¹¹ é um jogo de estratégia baseado em turnos para o sistema Android, tematizado nas tribos indígenas americanas, onde o jogador

¹⁰ <http://www.badlogicgames.com/wordpress/>

¹¹ O Anexo A apresenta o documento de *Game Design* do jogo América Tribal.

controla uma das três tribos existentes, gerenciando seus recursos e criando exércitos, para derrotar um outra tribo, controlada pelo computador.

2.6.1 Tribos

As três tribos presentes no jogo foram criadas baseadas nas tribos indígenas das Américas, separadas por semelhança cultural e região de atuação. As tribos são Norte, Centro e Sul, sendo que a tribo do Norte se assemelha às tribos norte-americanas, como Cherokee e Apache; a do Centro é uma união das culturas de tribos da América central e do sul, como Astecas e Incas; a do Sul é representada principalmente pelas tribos brasileiras, como a Guaraní e a Pataxó.

2.6.2 Mecânica do jogo

O América Tribal segue as regras de um jogo de estratégia baseado em turnos, ou seja, cada jogador tem seu turno, onde pode fazer uma quantidade limitada de ações, e deve gerenciar esses recursos para derrotar o jogador adversário.

O jogo é apresentado sobre um tabuleiro de células quadradas, onde são representados as unidades e o cenário. O cenário é dividido em recursos, passagem ou bloqueio, sendo que os recursos podem ser coletados pelos jogadores; o bloqueio impede qualquer tipo de movimentação dos personagens, e; a passagem possibilita a movimentação.

Existem duas unidades principais no jogo, que são a *aldeia* e o *explorador*. A aldeia é um objeto fixo no mapa do jogo, sendo que cada jogador possui a sua, e nela acontece todo o gerenciamento dos recursos. O explorador é um objeto que pode se movimentar pelo mapa do jogo, tal que cada jogador

também possui o seu, e ele é o responsável por coletar recursos e atacar o adversário.

As ações possíveis para um jogador são: mover o explorador pelo mapa; coletar recursos com o explorador; entrar em combate com o explorador do adversário; invadir a aldeia adversária, usando seu próprio explorador; construir melhorias na aldeia; criar novos edifícios na aldeia; criar novos soldados (personagens que fazem o combate), na aldeia, e recrutar os soldados criados.

O combate também é dividido em turnos, e ocorre quando os exploradores de jogadores adversários se encontram, ou quando um explorador de um jogador encontra a aldeia do outro. Cada jogador possui unidades de combate, que atacam as unidades adversárias em seu próprio turno, podendo danificá-las ou destruí-las. O jogador que perder todas as suas unidades perde o combate.

2.7 Algoritmo A*

O algoritmo A*¹² é um algoritmo de busca que utiliza uma heurística para encontrar o menor caminho em um grafo não-ponderado, ou o caminho de menor custo em um grafo ponderado.

Considerando-se o grafo como uma matriz bidimensional, onde as conexões ocorrem apenas com os nós horizontalmente e verticalmente adjacentes, o algoritmo pode ser dividido nas seguintes etapas (LESTER, 2005):

1. Seja A o nó de partida; seja B o nó final e; seja o nó Atual, inicialmente, o mesmo que o nó A.
2. Para cada nó C adjacente ao nó Atual, calcule $F = G + H$.
 - a. G é o custo para alcançar C, partindo do nó Atual;

¹² Comumente pronunciado como a-estrela.

- b. H é a soma dos custos para se deslocar de C até B , apenas com movimentos horizontais e verticais, sem considerar obstáculos.
- 3. Escolha o nó com o menor valor F . Este agora é o nó Atual.
- 4. Repita os itens 2 e 3 até que:
 - a. Seja encontrado o nó B . Este é o menor caminho; ou
 - b. Até que não haja nós disponíveis e B ainda não foi encontrado. Faça o caminho em ordem reversa até encontrar um nó que tenha caminhos disponíveis e repita o item 3; ou
 - c. Se B não foi encontrado, e nenhum nó possui caminhos disponíveis, então não existe um caminho entre A e B .

3 Metodologia

A produção deste trabalho foi realizada em três etapas. A primeira parte foi o projeto do editor, ou seja, definir tudo que será possível fazer com ele, e tudo que será necessário para ele rodar – imagens para o cenário, e as bibliotecas que foram utilizadas. A segunda etapa foi a implementação do editor. Na terceira e última parte foram realizados os testes.

3.1 Projeto do editor

O projeto do editor foi dividido em três etapas: análise de editores do mercado de jogos; projeto de elementos de interface e; projeto técnico.

3.1.1 Análise de editores do mercado de jogos

Para iniciar o projeto do editor, foram analisados dois editores de nível já consolidados no mercado de jogos.

O primeiro foi o RPG Maker VX Ace¹³, da empresa Enterbrain, Inc, apresentado na Figura 4. Mesmo que este não seja um editor de níveis para jogos de estratégia, ele apresenta elementos essenciais para um editor, segundo os critérios apresentados por Rouse (2005), que podem ser resumidos em apresentar o nível como este será visto no jogo, e possibilitar a visualização de uma área maior do que a apresentada para jogador durante o jogo, facilitando a criação de um mapa mais consistente e a visualização de erros. A interface do programa é dividida em quatro partes: barra superior, contendo os menus e ferramentas para edição; o quadro à direita, onde é mostrado o cenário; o quadro

¹³ <http://www.rpgmakerweb.com/>

inferior esquerdo, onde são mostrados os níveis existentes no jogo, e o quadro superior esquerdo, onde são mostrados as imagens possíveis para o mapa.

O outro editor analisado foi o Heroes of Might and Magic® III Map Editor, disponível em conjunto com o jogo Heroes of Might and Magic® III¹⁴, da empresa New World Computing, Inc, apresentado na Figura 5. A interface deste *software* pode ser dividida em quatro partes: 1) parte superior: contém os menus e as ferramentas do programa; 2) parte esquerda: apresenta o cenário; 3) parte superior direita: apresenta um pequeno mapa do cenário todo, para uma localização mais ágil; 4) parte inferior direita: apresenta os blocos de imagens, utilizados para povoar o cenário.



Figura 4 Ambiente do RPG Maker VX Ace

¹⁴ http://www.gog.com/game/heroes_of_might_and_magic_3_complete_edition

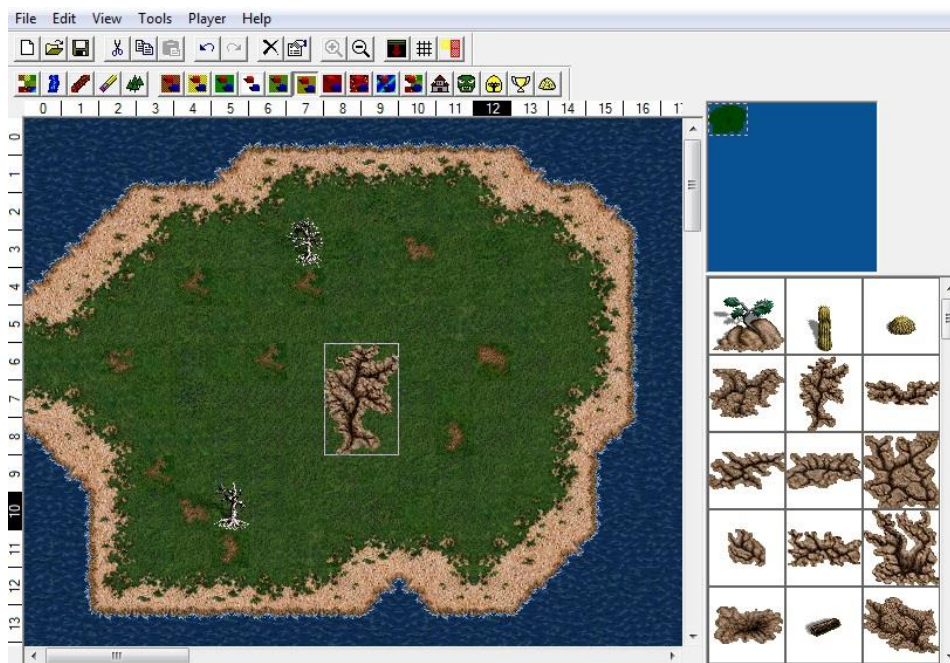


Figura 5 Ambiente do Heroes of Might and Magic® III Map Editor

Os dois editores analisados apresentam elementos de interface semelhantes. Ambos os editores possuem áreas de ferramentas para efetivamente desenhar o mapa, e uma área para o visualizar o mapa criado. Os editores também apresentam recursos para visualizar o nível da mesma maneira que ele será visto durante o jogo, e também possibilitam visualizar mais que o jogador poderá ver. Tanto o primeiro quanto o segundo editor podem exportar e importar os níveis criados, assim como alterá-los facilmente. O RPG Maker VX Ace é um editor de níveis genérico, já o Heroes of Might and Magic® III Map Editor é específico para o jogo Heroes of Might and Magic® III, e todos os níveis criados com este segundo editor necessitam deste jogo específico para funcionar. O editor criado neste projeto será específico para o jogo América Tribal.

3.1.2 Projeto de elementos de interface

Com base nos editores apresentados e nas especificações do jogo América Tribal, o editor de níveis criado neste projeto apresenta as seguintes especificações de interface:

- a) Uma área de desenho, situada na parte esquerda da tela, onde o cenário é apresentado. Nesta parte o usuário poderá adicionar elementos ao cenário e mover o cenário para visualizar uma parte específica do mesmo, caso este ultrapasse os limites da área.
- b) Uma área de ferramentas básicas, situada na parte direita superior da tela, onde são apresentadas as ferramentas para editar os quadros do mapa, divididas em camada inferior e camada superior. Nesta seção o usuário pode escolher entre os blocos possíveis para o cenário, usados para alterar o mesmo na área de desenho.
- c) Na parte direita inferior da tela, são apresentadas as ferramentas para mudar o nome do mapa, mudar o tamanho do mapa, criar um novo mapa, exportar o mapa criado para um arquivo, importar um mapa existente para o programa, e para verificar a eficácia do mapa.

A interface básica do editor pode ser visualizada na Figura 6.



Figura 6 Elementos da interface do editor de níveis

3.1.3 Projeto técnico

O tabuleiro do jogo, ou o mapa, é o local onde são representadas todos os objetos do nível, e este mapa é dividido em duas camadas: chão e objetos.

A camada *chão*, ou camada inferior, é representada por uma matriz de duas dimensões de números inteiros. Cada célula desta matriz possui um valor variando de zero a cinco, e estes valores representam as imagens do chão do cenário: 0 – grama; 1 – areia; 2 – terra; 3 – água; 4 – rocha e; 5 – floresta. As imagens de cada representação podem ser observadas na Figura 7.

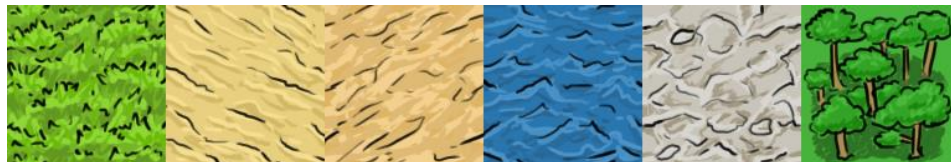


Figura 7 Elementos da camada inferior do cenário

A camada *objetos*, ou camada superior, também é representada por uma matriz bidimensional de números inteiros, onde cada célula da matriz possui um

valor que representa um tipo de objeto, segundo a classificação: 0 – vazio; 1 – fazenda; 2 – floresta; 3 – mina; 4 – observatório; 5 – carne; 6 – madeira; 7 – pedra; 8 – aldeia da tribo do jogador; 9 – aldeia da tribo do computador; 10 – explorador da tribo do jogador; 11 – explorador da tribo do computador. Na Figura 8 são apresentados os objetos, sendo que para a representação das tribos foi escolhida a do sul para o jogador, e a do norte para o computador.



Figura 8 Elementos da camada superior do cenário

As matrizes de ambas as camadas possuem mesma dimensão, e são apresentadas sobrepostas no cenário: a camada inferior é desenhada, e a camada superior é colocada acima da primeira. Todas as imagens da camada superior possuem fundo transparente, sendo que a imagem 0 (zero) – vazio – é totalmente transparente. A Figura 9 mostra um exemplo de cenário, juntando as camadas inferior e superior. Todos os blocos da camada superior, onde aparecem apenas os elementos da camada inferior, são preenchidas com o bloco *vazio*. Este bloco, na realidade, é apenas um indicador para o editor não desenhar nada na camada superior naquela célula do cenário.

Os níveis criados pelo editor serão exportados para o jogo América Tribal no momento de lançamento do jogo, pois são arquivos internos do jogo.



Figura 9 Exemplo de cenário

3.1.3.1 Especificação de um arquivo de nível

Um nível criado no editor é gravado em um arquivo texto, com extensão “.atm” (sigla para América Tribal Mapa), que segue um padrão simples: o nome do nível é o próprio nome do arquivo, porém sem considerar a extensão; na primeira linha do arquivo sempre será escrito “Nível América Tribal”; a segunda linha mostra as dimensões do mapa, em quadros horizontais por verticais; na terceira linha será escrito “Blocos Chão”, para mostrar visualmente de qual matriz se trata; as próximas linhas são a matriz da camada inferior do mapa, separadas por vírgulas e linhas, para facilitar uma visualização; na próxima linha será escrito “Blocos Objetos”, também para facilitar a visualização; as linhas subsequentes são a matriz da camada superior, representadas da mesma maneira que a camada inferior, e; na última linha é escrito “Fim”, para facilitar a visualização. A Figura 10 mostra um exemplo de um arquivo texto de um nível, enquanto a Figura 11 mostra a representação visual do mesmo nível, no editor.

```

mapa.atm x
1  Nível America Tribal
2  Dimensões:13 7
3  Blocos Chão:
4      1  1  1  1  4  4  1  1  1  1  1  1  1
5      1  1  1  1  4  4  1  1  1  1  1  1  1
6      1  1  1  4  4  4  4  1  1  1  1  1  1
7      1  1  4  4  4  4  4  4  1  1  1  1  1
8      1  1  1  1  1  1  4  4  4  4  4  4  1
9      1  1  1  1  1  1  1  4  4  4  4  1  1
10     1  1  1  1  1  1  1  1  1  3  1  1  1
11  Blocos Objetos:
12     0  0  0  0  0  0  0  0  0  0  0  0  2
13     0  0  0  0  0  0  3  0  0  0  11  9  0
14     0  0  1  0  0  0  0  0  0  0  0  0  0
15     0  0  0  0  0  0  0  0  0  1  0  0  0
16     0  0  0  8  0  3  0  0  0  0  0  0  0
17     0  0  0  0  10  0  0  0  0  0  0  0  0
18     2  0  0  0  0  0  0  0  0  0  0  0  0
19  Fim

```

Figura 10 Representação de um nível em um arquivo texto



Figura 11 Representação de um nível no editor

3.1.4 Plataforma alvo

O jogo América Tribal é um jogo desenvolvido para a plataforma Android, porém, mesmo que este editor crie níveis para tal jogo, ele foi desenvolvido para ser utilizado em computadores *desktop* com sistemas operacionais convencionais para tal tipo de computador (como Windows e Linux, por exemplo), pois estes geralmente possuem um monitor de dimensões maiores que um dispositivo móvel, além de maior capacidade de processamento. É interessante que o *level designer*¹⁵ tenha uma visão mais ampla do nível que o jogador, pois assim será mais fácil verificar sua consistência e integridade, e também facilitará a sua manutenção (ROUSE, 2005).

3.1.5 Verificação de eficácia

O editor possui um módulo de verificação de eficácia do nível, e este módulo é responsável por encontrar erros básicos que impossibilitariam o jogador de vencer naquele nível. A verificação de eficácia ocorre em três etapas: na primeira, é verificado se ambos o jogador e o computador possuem exatamente uma vila e um explorador; a segunda etapa verifica os recursos, se existem ao menos uma fazenda, uma mina e uma floresta para cada tribo, e; a última etapa é responsável por verificar se existe um caminho entre as duas tribos, e tal verificação ocorre em duas partes:

- a) Criação de um mapa de passagem e bloqueio: é criada uma outra matriz de mesma dimensão que as camadas inferior e superior, povoada apenas com os valores: 0 – passagem; 1 – bloqueio; 2 – ponto origem, e; 3 – ponto destino. O ponto origem é a posição da

¹⁵ Pessoa responsável por projetar e desenvolver os níveis do jogo.

aldeia do jogador, e o ponto destino é a posição da aldeia do computador. Todos os quadros que não são considerados como passagem nas camadas inferior e superior são marcadas com 1, e as demais com 0. Os blocos considerados como passagem na camada inferior são: 0 – grama; 1 – areia e; 2 – terra. Os blocos considerados passagem na camada superior são: 0 – vazio; 5 – carne; 6 – madeira; 7 – pedra; 10 – explorador da tribo do jogador, e; 11 – explorador da tribo do computador.

- b) Busca por um caminho entre as tribos. Para tal é utilizado o algoritmo A*. A matriz criada na etapa “a)” é o ponto de partida para este algoritmo, que verifica se existe um caminho entre o ponto origem e o ponto destino, utilizando o mapa de passagem criado naquele mesmo item para percorrer o caminho.

Se qualquer etapa de verificação falhar, então o nível não é eficaz, e é exibida uma mensagem de erro apropriada para o usuário.

3.2 Materiais e recursos tecnológicos

Para a implementação deste trabalho foram utilizadas diversas ferramentas, as quais são descritas a seguir: para implementação e testes do editor foi utilizado um computador *desktop* com 2GB de memória RAM, processador Intel® Core™2 Duo de 2.66 GHz, com 1TB de espaço de armazenamento, utilizando um Sistema Operacional de 32-bit.

A codificação foi feita na linguagem Java, e a IDE Eclipse¹⁶ foi utilizada para gerenciar o projeto. A versão utilizada da IDE foi a 4.3.1 embutida no ADT

¹⁶ <http://www.eclipse.org/>

Bundle¹⁷ do Android, que é uma versão que engloba o ambiente Eclipse e o Android SDK¹⁸ (*Software Development Kit*), o AVD¹⁹ (*Android Virtual Device*), que é um emulador do sistema Android.

Em conjunto com a linguagem Java foi utilizado o *framework* libGDX, para auxiliar nas funções gráficas como desenhar as imagens na tela, gerenciar a atualização de imagens, gerenciar a inicialização, a utilização da biblioteca gráfica e resolução da aplicação, e gerenciar o carregamento de arquivos de imagens para a memória. As funções de gerenciamento de entrada, para o uso do *mouse* e teclado no editor também foram utilizadas, assim como as funções de gerenciamento de arquivos texto, para importar e exportar os níveis criados.

As imagens e desenhos utilizadas no desenvolvimento do editor de níveis foram criadas ou editadas no editor de imagens GIMP²⁰, utilizando a versão 2.8.10 do *software*. As imagens do cenário e dos objetos utilizadas no editor são as mesmas utilizadas no jogo América Tribal. As figuras dos botões e outros elementos do editor feitas separadamente, utilizando-se o GIMP, ou diretamente em linhas de código, na implementação do editor. Com exceção das imagens dos botões e das imagens que representam a cor dos times, todas as outras foram criadas anteriormente, para o jogo América Tribal, e foram reutilizadas neste projeto.

3.3 Utilização da libGDX

O libGDX foi escolhido como *framework* para este trabalho pois ele segue a mesma base do *framework* criado por ZECHNER (2011), utilizado em LIMA (2013) para a implementação de seu projeto de interface. Porém o

¹⁷ <http://developer.android.com/sdk/index.html>

¹⁸ Parte integrante do ADT Bundle do Android

¹⁹ Parte integrante do ADT Bundle do Android

²⁰ <http://www.gimp.org/>

libGDX é muito mais completo, em termos de funcionalidades, que a versão original apresentada por Zechner. A versão do *framework* utilizada foi a 0.9.8.

Inicialmente o projeto foi criado utilizando a interface de criação de projetos do libGDX. Esta interface cria e interliga projetos da IDE Eclipse, sendo necessário apenas importa-los para a IDE, agilizando a criação do projeto do editor. A Figura 12 mostra a parte principal da interface de criação de projetos, onde é necessário definir apenas o nome do projeto, o pacote principal, o nome da classe principal, para quais plataformas exportar, qual versão do *framework* utilizar, e a pasta destino. Existem mais opções avançadas, que não foram abordadas neste projeto.

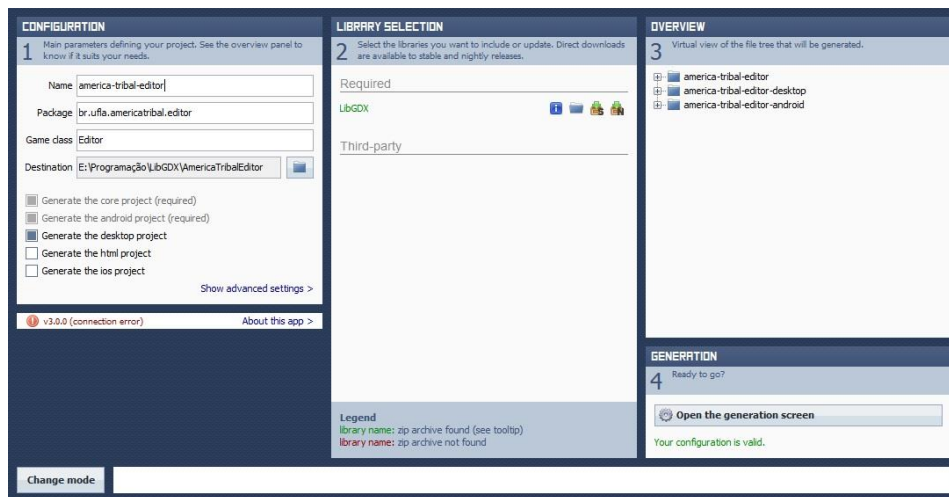


Figura 12 Interface de criação de projetos libGDX

Através deste *software* foram gerados os projetos “américa-tribal-editor”, “américa-tribal-editor-android” e “américa-tribal-editor-desktop”, sendo que todo o código do editor é colocado no primeiro projeto. O segundo serve para exportar o código para Android, além de armazenar todos os recursos do editor, como sons e imagens, por exemplo. O terceiro projeto apenas é uma

ponte entre o código base, no primeiro projeto, e a execução do mesmo em computadores *desktop*, ou seja, ele apenas exporta o código para que funcione em tal plataforma. No projeto deste editor, o projeto “américa-tribal-editor-android” foi utilizado apenas para armazenar os recursos necessários para o editor, e o projeto “américa-tribal-editor-desktop” para a funcionalidade padrão, ou seja, converter o projeto para funcionar em computadores *desktop* equipados com a tecnologia Java. O projeto “américa-tribal-editor” é projeto base, no qual todo o código do editor reside, e as classes deste projeto são descritas mais detalhadamente no tópico Diagrama de classes, adiante.

3.3.1 Diagrama de classes

Esse projeto é constituído de oito classes, separadas em dois pacotes, além das classes e interfaces utilizadas do *framework* libGDX. O diagrama de classes deste projeto é apresentado na Figura 13. Dentre as oito classes, três são classes auxiliares que implementam o algoritmo A*, colocadas no pacote “br.ufla.americatribal.editor.aestrela”. Os detalhes sobre a implementação do algoritmo A* podem ser encontrados no Apêndice A. As cinco classes restantes são divididas em classe inicializadora: “Editor”; tela principal: “TelaPrincipal” e; classes auxiliares: “DesenhistaDaTelaPrincipal”, “Mapa” e “Conf”. O código e descrição das classes pode ser encontrado no Apêndice B.

A classe “TelaPrincipal” é a mais importante, e é onde ocorre todo o gerenciamento dos recursos do editor. Ela implementa as interfaces “Screen”, “InputProcessor” e “TextInputListener”, provenientes do libGDX. A interface “Screen” é responsável por apresentar os elementos gráficos para o usuário, assim como gerenciar o estado²¹ atual da aplicação. A interface “InputProcessor”

²¹ Os estados possíveis são “pausado” e “não pausado”, e as transições entre eles.

é responsável por gerenciar toda entrada de periféricos, como o *mouse* e o teclado, sendo possível verificar, por exemplo, se uma tecla acabou de ser pressionada, acabou de ser solta, ou continua pressionada. A interface “`TextInputListener`” é responsável por apresentar uma caixa de diálogo ao usuário, onde este pode entrar com um texto, e aceitar ou cancelar a mensagem. Utilizado neste projeto para renomear, alterar as dimensões, verificar a eficácia e salvar o mapa atual, assim como criar um novo mapa, ou carregar um existente.

A classe “`Mapa`” também é muito importante no projeto, pois ela armazena as informações, e gerencia os níveis criados pelo editor. Nela também é feita a verificação de eficácia do nível, assim como a exportação e a importação de níveis.

A classe “`DesenhistaDaTelaPrincipal`” é uma classe auxiliar que apenas carrega as imagens e é responsável pela disposição destas imagens na tela. Este código é colocado separadamente da classe “`TelaPrincipal`” pois aumenta a modularização do projeto, e caso seja necessário apenas alterar os métodos de desenho ou de carregamento de imagens, não é necessário nenhuma alteração na classe principal.

3.3.1.1 Descrição das classes

3.3.1.1.1 Pacote “`br.ufla.americatribal.editor.aestrela`”

Neste pacote estão contidas as classes que implementam o algoritmo A*: “`AEstrela.java`”, “`Lista.java`” e “`Ponto.java`”. O código implementado está presente no Apêndice A.

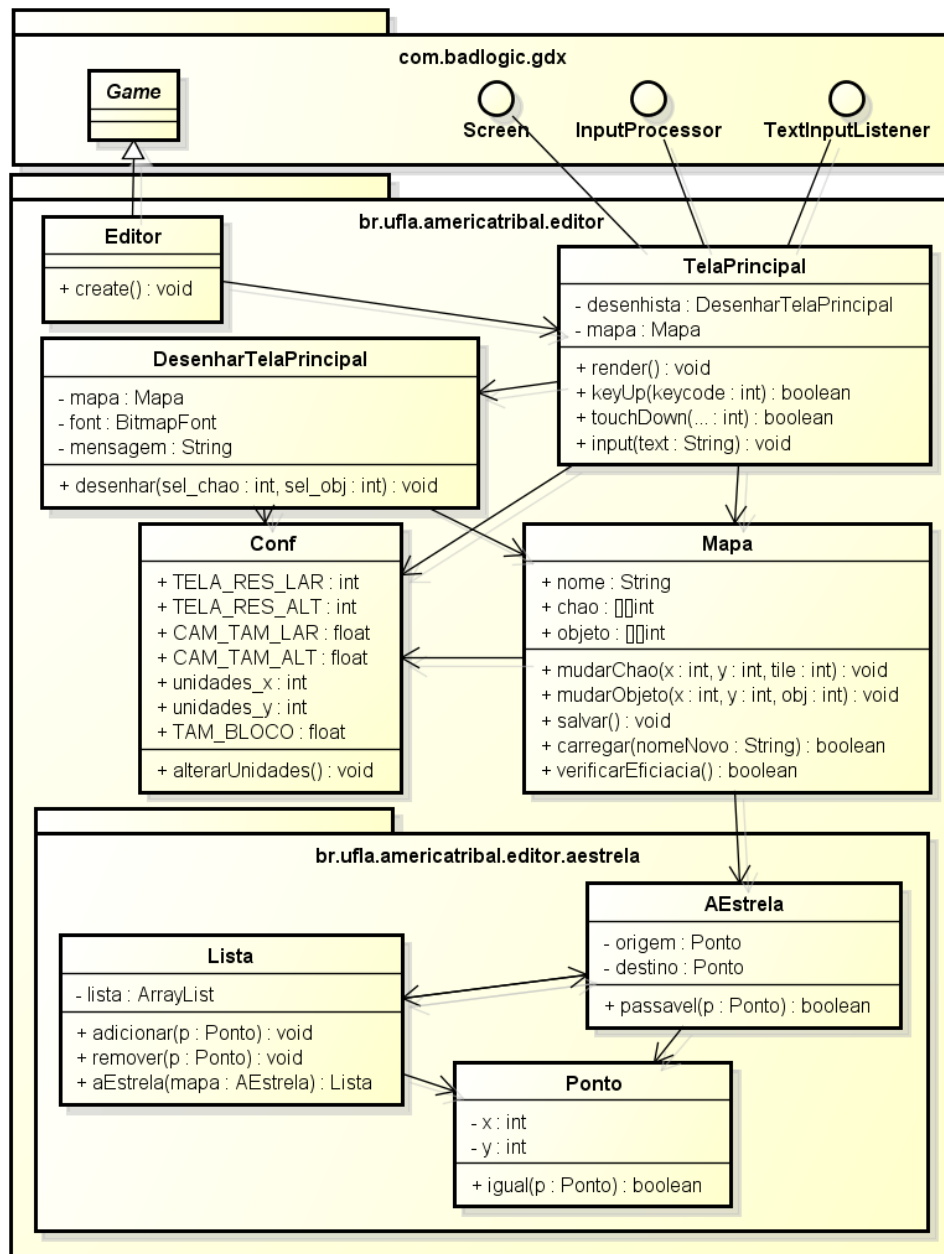


Figura 13 Diagrama de classes

3.3.1.1.1.1 Classe “AEstrela.java”

Esta classe armazena os dados necessários para o funcionamento do algoritmo, e passa os parâmetros essenciais para a classe “Lista.java”. O atributo “mapa” é uma matriz bidimensional do tipo “inteiro”, e é responsável por armazenar o cenário. As células que tenham o valor “0” (zero) são blocos passáveis²², enquanto as que possuem o valor “1” (um) são bloqueio. Também são armazenadas as dimensões da matriz, horizontal e vertical, respectivamente em “tx” e “ty”, dois atributos do tipo “inteiro”, e os pontos de origem, destino e ponto atual, utilizados durante o processamento principal do algoritmo, são armazenados como atributos do tipo “Ponto”. A classe possui métodos para verificar se um bloco do cenário é considerado passável ou bloqueio, utilizando o método “passavel”, que retorna uma variável do tipo “booleano”, sendo que “verdadeiro” significa que o bloco é passável, e “false” indica o contrário.

3.3.1.1.1.2 Classe “Ponto.java”

Esta classe simplesmente armazena as informações de um ponto para o A*. Ela possui dois atributos, do tipo inteiro, “x” e “y”, que representam a posição horizontal e vertical, respectivamente, deste ponto na matriz do cenário. Também armazena os valores G, H e F, para cálculo do melhor caminho, pelo algoritmo A*, em atributos homônimos, do tipo “inteiro”. O atributo “pai”, do tipo “Ponto” armazena o ponto pelo qual o algoritmo A* chegou a esse, para armazenar informações sobre o caminho percorrido. Os principais métodos utilizados nesta classe são o método “custos”, que atribui os valores G, H e F do

²² Célula que o algoritmo considera como possível caminho, em detrimento das células “bloqueio”, as quais o algoritmo simplesmente ignora.

ponto, e os métodos “cima”, “baixo”, “esquerda” e “direita”, que retornam o ponto correspondente à direção desejada, em relação a este ponto.

3.3.1.1.1.3 Classe “Lista.java”

É nesta classe que o processamento principal do algoritmo acontece. Nela é armazenada uma lista do caminho escolhido pelo algoritmo. Esta lista é armazenada no atributo “lista”, do tipo “ArrayList<Ponto>”, uma lista de pontos, onde cada ponto, exceto o primeiro, tem como pai o ponto anterior da lista, sendo possível percorrer a lista facilmente utilizando-se apenas os pontos. Seus principais métodos são o “adjacentes”, um método de retorna uma lista dos possíveis pontos a serem escolhidos para o algoritmo como o próximo da lista principal, e o método “aEstrela”, que faz todo o cálculo do algoritmo, e retorna a lista do caminho encontrado.

3.3.1.1.2 Pacote “br.ufla.americatribal.editor”

Este é o pacote principal do editor. Nele estão contidos todas as classes e métodos de gerenciamento do editor de níveis. O código da implementação do editor pode ser encontrado no Apêndice B.

3.3.1.1.2.1 Classe “Editor.java”

Esta classe estende a classe abstrata “Game” do libGDX, e é a primeira classe a ser chamada quando a aplicação é iniciada. Ela é responsável pelo gerenciamento da aplicação como um todo. O único método presente é o “create”, que é responsável por instanciar um objeto da classe “TelaPrincipal”, e transferir o controle da tela para tal classe.

3.3.1.1.2.2 Classe “TelaPrincipal.java”

Esta classe é responsável pela maior parte do gerenciamento das funções do editor, como gerenciamento da tela, dos dispositivos de entrada (*mouse* e teclado) e das caixas de diálogo como o usuário. Ela possui diversos atributos para o gerenciamento de ações, como os do tipo “booleano”: “shift_pressionado”, responsável para armazenar o estado da tecla “Shift”, utilizada para escolher qual tipo de bloco será selecionado com os botões de seleção; “control_pressionado”, que armazena o estado da tecla “Ctrl”, que se estiver com o valor “verdadeiro”, indica que o movimento do *mouse* deve também mover o cenário; “movendo” indica se o cenário está sendo movido pelo *mouse*, seja pela tecla “Ctrl” apertada, ou pelo botão do meio do *mouse*; “caps_ligado”, armazena o estado da tecla “Caps Lock”, que se estiver ligada, altera a variável “movendo” da mesma maneira que a tecla “Ctrl” faria; e o atributo “botao1_objeto”, que indica se o botão esquerdo do mouse irá desenhar blocos do tipo “objeto” ou do tipo “chão” no cenário, respectivamente com o valor “verdadeiro” e “falso” do atributo. Possui dois atributos do tipo “inteiro”, responsáveis por armazenar qual bloco de cada tipo, “chão” e “objeto”, estão selecionados no momento. Um atributo do tipo inteiro, “desenhando”, indica qual tipo de bloco está sendo desenhado no momento. O atributo “mapa”, do tipo “Mapa” armazena as informações do cenário. Os atributos do tipo “String”, “nomeMapa” e “entrada”, armazenam, respectivamente, o nome do mapa atual, e a última entrada recebida pelo usuário, por alguma caixa de diálogo. Por fim, o atributo “desenhista” armazena um objeto da classe “DesenhistaDaTelaPrincipal”.

O método “show” é o primeiro a ser chamado quando a classe é criada, e é proveniente da interface “Screen” do libGDX, e responsável por toda a

inicialização de atributos e métodos da classe. O método “render” é chamado toda vez que o sistema desenha algo na tela, e neste método é limpada a tela, com uma cor cinza escura, e então desenhado todo o cenário. Os métodos “keyDown” e “keyUp” são chamados quando alguma tecla é pressionada ou solta, respectivamente, e processam as informações recebidas pelo teclado. Os métodos “touchDown” e “touchUp” são semelhantes aos anteriores, porém para as entradas de dados do *mouse* ou da tela de *touch*. O método “touchDragged” é responsável por gerenciar o movimento do *mouse* enquanto algum botão deste é pressionado, em contrapartida, o método “mouseMoved” é responsável pelo gerenciamento do movimento do *mouse* enquanto nenhum botão do mesmo é pressionado. O método “input” é responsável pela criação e comunicação com as caixas de diálogo com o usuário, como as opções de salvar e carregar um nível.

3.3.1.1.2.3 Classe “DesenhistaDaTelaPrincipal.java”

Classe responsável pelo carregamento das imagens utilizadas no editor, assim como a apresentação destas imagens na tela. Ela possui diversos atributos, e os principais são os atributos de posicionamento: “xNoMapa”, “yNoMapa”, que armazenam a célula correspondente à posição do *mouse* sobre o cenário, os atributos “deslocx” e “deslocy”, responsáveis por armazenar a posição do canto inferior esquerdo do cenário, em relação à área do cenário, os atributos “aldeia”, “explorador”, “cenário_chao”, “cenário_objeto” e “aura”, todos vetores do tipo “TextureRegion”, que servem para armazenar as imagens de cada tipo, o atributo “batch”, do tipo “SpriteBatch”, responsável por efetivamente desenhar as imagens na tela, de uma maneira eficiente, e o atributo “camera”, do tipo “OrthographicCamera”, responsável pela exibição de tudo no editor.

Os métodos principais são o “carregarTexturas”, que como o próprio nome diz, é responsável por carregar todas as imagens utilizadas no editor, os

métodos de desenho, “desenhar”, “desenharTexto”, “desenharInterface”, “desenharChao”, “desenharObjetos” e “desenharRetangulos”, que são responsáveis por desenhar cada parte do editor, o método “moverMapa”, responsável pelo movimento do cenário dentro da área de cenário, com o *mouse*, o método “selecionarItemObjetos”, responsável pelo gerenciamento de qual célula é selecionada pelo mouse, e o método “adicionarMensagem”, responsável pelo gerenciamento de mensagens salvas no arquivo de registro e na tela.

3.3.1.1.2.4 Classe “Mapa.java”

A classe mapa armazena os valores do nível atual, assim como possui os métodos de salvar e carregar um nível. Os atributos são o “nome”, um atributo do tipo “String” que armazena o nome do mapa, “tam_x” e “tam_y”, atributos do tipo “inteiro” que armazenam o tamanho horizontal e vertical do mapa, respectivamente, e os atributos “chao” e “objeto”, matrizes bidimensionais do tipo “inteiro” que armazenam os valores de cada camada do cenário. O atributo “passavel”, uma matriz bidimensional do tipo “inteiro” armazena o mapa de passagem, criado para ser utilizado no algoritmo A*.

Os métodos são o “mudarTamanho”, que altera o tamanho das matrizes do mapa e as reiniciam, o “mudarChao” e “mudarObjeto”, que alteram o valor na matriz da célula escolhida, para um novo valor, os métodos “salvar” e “carregar”, que exportam e importam um nível, respectivamente, o método “criarMapaPassavel”, que cria um mapa de passagem para ser utilizado no algoritmo A*, o método “verificarCaminhoEntreTribos”, no qual é chamado o algoritmo A*, o método “verificarJogadores” garante que existam uma aldeia e um explorador para cada tribo, o método “verificarRecursos”, responsável pela verificação dos recursos para o verificador de eficácia, e o método “verificarEficaci”, que junta todos os métodos de verificação.

3.3.1.1.2.5 Classe “Conf.java”

Essa classe apenas armazena valores globais utilizados pela aplicação. Ela possui apenas um método, “alterarUnidades”, que recalcula a proporção de tamanho entre o tamanho da câmera, e a resolução da tela. Nesta classe são armazenadas constantes que possuem o valor da resolução padrão da aplicação, do tamanho padrão da câmera, do tamanho de um bloco de cenário e do local aonde estão armazenadas as imagens.

3.4 Testes

Após a implementação do editor, foram realizadas duas etapas de testes: funcionalidade e usabilidade.

3.4.1 Testes de funcionalidade

Na primeira etapa foram feitos testes de funcionalidade, verificando se todas as funções propostas foram implementadas corretamente. Esta etapa contou com duas iterações de aperfeiçoamento, sendo que na primeira foram encontrados erros de representação dos dados das camadas, na qual alguns tipos de células não estavam presentes na apresentação do nível, ocorrendo também erros de endereçamento de memória. Estes erros foram corrigidos e, na segunda iteração, foram encontrados erros de gerenciamento de arquivo, havendo incompatibilidade entre o formato de arquivos exportados e o formato esperado para a importação de arquivos. Após as duas iterações não foram encontrados mais erros de funcionalidade, porém o programa não foi submetido a testes mais rigorosos para verificar outros tipos de problemas.

3.4.2 Testes de usabilidade

Os testes de usabilidade consistem em uma coleta de dados de usuários, para definir uma diferença entre criar um nível utilizando um editor, e criar um nível diretamente em um arquivo texto. Para tal foi criado um nível base, com as menores dimensões possíveis para o editor, apresentado na Figura 14, e foram selecionadas cinco pessoas para o teste. Todos os usuários foram treinados para criar um nível no editor, e também para criar um nível em um editor de texto²³.

Cada usuário criou o mesmo nível base, duas vezes, uma utilizando o editor de níveis, e outra utilizando o editor de texto. Ambas as etapas do teste foram supervisionadas, e os usuários tinham sempre à sua disposição o mapa base, mostrado na Figura 14, e uma tabela relacionando o cada bloco com seu respectivo valor numérico.



Figura 144 Nível para testes de usabilidade

²³ O editor de texto utilizado foi o Notepad++ versão 6.3.3. <http://notepad-plus-plus.org/>

4 Resultados e discussão

4.1 Resultados

4.1.1 Editor de níveis

O editor criado possui uma única tela, apresentada na Figura 14. Ela pode ser dividida em três partes: área do cenário, área dos blocos e área das ferramentas.



Figura 155 Editor finalizado

4.1.1.1 Área do cenário

A área do cenário, mostrada na Figura 15, apresenta o nível de forma visual, da mesma maneira que este será visto durante o jogo, e também de maneira mais ampla. A interação com o cenário é feita através do *mouse* e,

opcionalmente, do teclado. Similar a programas gráficos, o usuário pode “desenhar” blocos no cenário ao clicar em uma área do mesmo com o botão direito ou esquerdo do *mouse*. O botão direito representa o bloco principal, e o botão esquerdo, o secundário, explicados do item “4.1.1.2 Área dos blocos”. Também é possível mover o cenário dentro desta área ao mover o *mouse* com o botão central pressionado, ou segurando-se a tecla “CTRL” no teclado e movendo-se o *mouse*, podendo-se assim, visualizar todas as partes de um mapa de dimensões muito grandes.

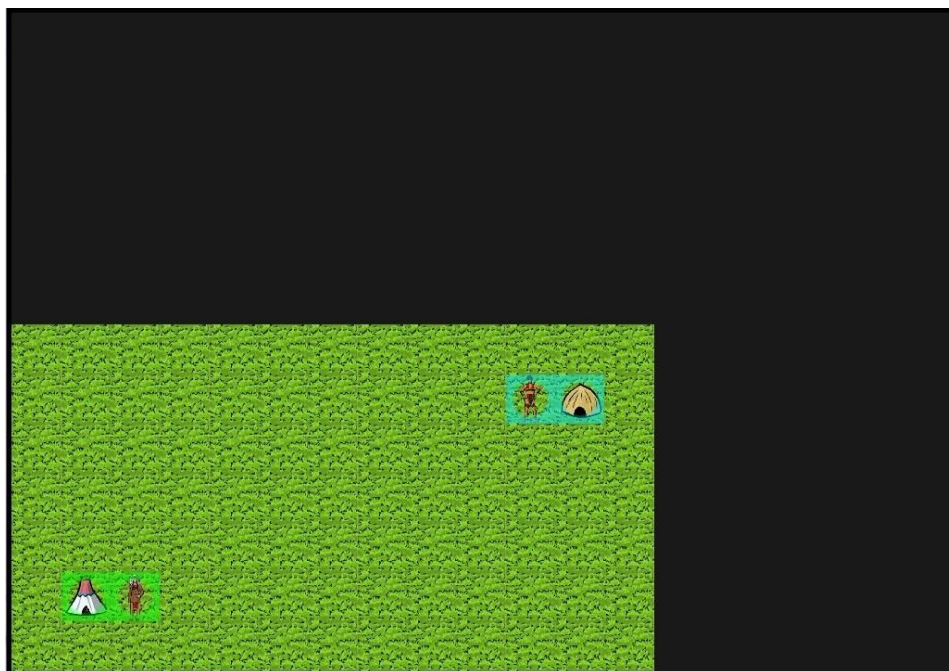


Figura 166 Área do cenário

4.1.1.2 Área dos blocos

Na área dos blocos podem ser alterados os tipos de blocos que serão “desenhados” na área do cenário. Esta área é subdivida em duas subáreas: a

superior, contendo os blocos da camada *chão* do cenário, representada na Figura 16, e; a inferior, que contém os blocos da camada *objetos* do cenário, representada na Figura 17. Em ambas as áreas, o quadrado maior apresenta o bloco selecionado para sua respectiva categoria, e ele pode ser alterado clicando-se em um dos blocos pequenos à direita.

O bloco que será desenhado no cenário é chamado de “bloco selecionado”. Existem dois tipos de “bloco selecionado”: o principal e o secundário. O principal é representado por um retângulo amarelo e um vermelho, mostrado na Figura 16, desenhado por cima de um dos blocos grandes; o secundário é o bloco que não tem esses retângulos. O bloco principal é escolhido ao clicar-se em algum bloco de uma das duas camadas, *chão* ou *objetos*, com o botão esquerdo do *mouse*, atribuindo o bloco principal àquela camada. Clicar com o botão direito do *mouse* não altera a seleção do bloco principal ou secundário, apenas o bloco selecionado para a camada correspondente.



Figura 177 Área dos blocos - camada *chão*

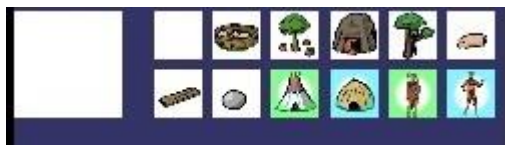


Figura 188 Área dos blocos - camada *objetos*

4.1.1.3 Área das ferramentas

A área das ferramentas, apresentada na Figura 18, possui todas as funções que não são englobadas nas áreas anteriores: renomear o mapa, alterar o tamanho do mapa, criar um novo mapa, salvar o nível em um arquivo, carregar um nível de um arquivo e verificar a eficácia do nível. Nesta área é apresentado o nome do mapa atual, e logo abaixo um botão para alterar este nome. Também é mostrado o tamanho, horizontal e vertical, do nível. O tamanho mínimo consiste de treze blocos horizontais por sete blocos verticais, e o máximo²⁴ é trinta por trinta blocos. O tamanho horizontal pode ser alterado clicando-se no interior do retângulo azulado escrito “hor:”, e o tamanho vertical pode ser alterado clicando-se no correspondente “ver:”. O botão “Novo Mapa” cria um mapa de tamanho mínimo, onde todos os blocos tem o valor zero, exceto os blocos das aldeias e exploradores. O botão “Salvar” exporta o nível criado para um arquivo externo. O botão “Carregar” importa um nível contido em um arquivo externo para o editor. O botão “Verificar Eficácia” faz a verificação de eficácia do nível, e apresenta uma mensagem de erro, caso falhe, ou uma mensagem de sucesso, caso o nível seja eficaz.

4.1.1.4 Mensagens do editor

O editor emite mensagens para diversas ações, como por exemplo se houve erro ou sucesso ao tentar salvar ou carregar um arquivo. Todas as mensagens são salvas em uma subpasta “log”, na pasta raiz do projeto, dentro do arquivo “mensagens.txt”.

4.1.1.5 Níveis externos

²⁴ O tamanho mínimo e o tamanho máximo de um nível foram discutidos e definidos empiricamente, no projeto do jogo América Tribal.

Os níveis exportados pelo editor são salvos em uma subpasta “níveis” na pasta raiz do projeto. Os níveis que serão importados também devem ser colocados nesta mesma pasta.



Figura 199 Área das ferramentas

4.1.1.6 Testes de usabilidade

Durante a etapa de testes no editor de texto, foi possível notar que a maioria dos usuários fazia frequentes confusões sobre qual número representava qual bloco, e também para contar a quantidade blocos restante. Os dados recolhidos durante os testes estão presentes na Tabela 1. Alguns dos usuários sugeriram melhorias no editor: adição de um gerenciador de projetos, semelhante ao gerenciador existente no RPG Maker VX Ace, e adição de um

minimapa, semelhante ao existente no Heroes of Might and Magic® III Map Editor.

Tabela 1 Testes de usabilidade

Usuário	Tempo utilizando o editor de níveis	Tempo utilizando o editor de texto	Proporção entre o tempo no editor de níveis pelo tempo no editor de texto
A	1:32 min	4:21 min	2,84
B	3:10 min	6:09 min	1,94
C	2:50 min	8:25 min	2,98
D	2:28 min	7:10 min	2,90
E	1:43 min	4:01 min	2,33
Média entre os usuários	2:20 min	6:01 min	2,60

Foi possível analisar que este editor de níveis apresenta menos recursos que os editores analisados no projeto. Aqueles editores são mais robustos e com mais facilidade, e segundo alguns dos usuários, mais “atraentes” para se trabalhar.

4.2 Discussão

Durante o desenvolvimento do editor foram encontradas diversas dificuldades para a criação do projeto do editor, pois não existem, ou são muito escassos, artigos ou qualquer material sobre a criação de um editor de nível. Existem diversos materiais discutindo sobre editores de níveis, ou tutoriais de como utilizar um editor específico, mas praticamente nada sobre como criar um

editor. Devido a esta escassez de material, foram analisados editores de nível já existentes utilizando-se métodos empíricos de observação, criando-se um projeto de editor baseado na experiência e nas especificações do jogo América Tribal. Esta etapa de criação do projeto do editor foi a etapa mais longa do desenvolvimento deste editor.

A utilização de um *framework* é encorajada durante a criação de um jogo, pois assim pode-se utilizar mais o tempo para a criação de partes do jogo, em vez de gastar este tempo lidando com questões técnicas (LIMA, 2013). Neste projeto não foi implementado um jogo, porém a área de exibição do nível funciona como em um jogo, sendo necessário exibir imagens, as quais podem ser trocadas dinamicamente, sendo necessário um motor gráfico. Por este motivo, principalmente, foi escolhido um *framework* para gerenciar as questões técnicas como, por exemplo, a exibição de imagens. Esta parte do desenvolvimento se mostrou bastante simples.

As etapas de testes ocorreram de forma distinta, sendo que a primeira etapa, testes de funcionalidade, ocorreu logo após a primeira implementação do editor e a segunda, testes de usabilidade, ocorreu durante a escrita do projeto. A primeira etapa foi feita de maneira empírica, apenas verificando se o editor realizava as funções propostas durante a execução. Esta etapa foi simples e rápida, e os erros foram verificados com agilidade. A segunda etapa foi um teste supervisionado, realizado para verificar a eficiência do uso de um editor de níveis, ao invés de criar um nível em um editor de texto. O teste foi realizado apenas uma vez com cada usuário, e não foram testados tempos com dimensões maiores. Durante este teste foi possível observar que o editor de níveis facilita muito a criação de um nível, pois, na média, os usuários levaram mais de duas vezes e meia mais tempo para criar um nível num editor de texto, do que em um editor de níveis. Este editor ainda pode ser considerado inferior aos editores analisados durante o projeto, mas tais editores contaram com programadores

especializados para a sua criação, além de estarem a vários anos no mercado, e já tiveram várias iterações de melhoramento.

5 Conclusão

Neste projeto foi criado um editor de níveis para o jogo América Tribal, utilizando editores já consolidados no mercado como base, e utilizando um *framework* para auxiliar a programação gráfica e de gerenciamento de recursos. O editor possui as funcionalidades de criar, exportar e importar um nível, assim como verificar se o nível é eficaz dentro do contexto do jogo²⁵. Após a implementação foram realizados testes de usabilidade, para verificar o funcionamento do editor, e possíveis erros de programação. Também foram realizados testes de usabilidade para avaliar a eficiência de editores de nível, em relação à criar um nível em um editor de texto.

Com a conclusão do projeto foi possível reforçar a afirmação de Rouse (2005), na qual o autor diz que um editor de níveis é uma ferramenta essencial para um jogo que possua mais de um nível. Os testes de usabilidade mostraram que esta afirmação pode ser considerada correta, pois mesmo com um nível de dimensões muito pequenas, foi possível observar uma diferença muito grande no tempo de criação utilizando um editor de níveis, com o tempo de criação utilizando um editor de texto.

A criação de um editor específico para um jogo também pode auxiliar bastante na criação de um nível, pois o *level designer* tem à mão diversas ferramentas especializadas diretamente para aquele jogo. Um editor de níveis genérico pode ser uma opção interessante para um projeto futuro, mas foge do escopo deste projeto.

Trabalhos futuros neste projeto podem melhorar a parte de interação do usuário com o cenário, colocando opções de ampliar ou reduzir a visualização do mapa, ou *zoom*. Também poderiam ser criadas outras funcionalidades no

²⁵ Verificar se existem recursos para ambos os jogadores, e verificar se existe ao menos um caminho entre as aldeias dos jogadores.

editor, para agilizar a criação de níveis, como um gerenciador de projetos, semelhante ao do RPG Maker VX Ace, e um minimapa, semelhante ao do Heroes of Might and Magic III Map Editor. Também seria interessante a criação de uma interface integrando o *framework* libGDX a alguma biblioteca de interface, como o AWT²⁶ ou o Swing²⁷.

²⁶ <http://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html>

²⁷ <http://docs.oracle.com/javase/7/docs/api/overview-summary.html>

6 Referências Bibliográficas

BATES, B. **Game Design**. 2ª. ed. Boston: Thomson Course Technology PTR, 2004. ISBN 1-59200-493-8.

KOSTER, R. **A Theory of Fun for Game Design**. Scottsdale: Paraglyph Press, Inc, 2005. ISBN 1-932111-97-2.

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

LESTER, P. A* Pathfinding for Beginners. **Policy Almanac**, 2005. Disponível em: <<http://www.policyalmanac.org/games/aStarTutorial.htm>>. Acesso em: 16 Setembro 2013.

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.

LOBO, C. Z.; VERDI, L. C. G.; ELIAS, P. C. Um estudo exploratório sobre o mercado de jogos eletrônicos no Brasil. **Revista Conteúdo**, Capivari, v. 2, p. 78-96, 2012. ISSN 1807-9539.

PEMMARAJU, V. Make Your Life Easier: Build a Level Editor. **gamedev tuts+**, 2012. Disponível em: <<http://gamedev.tutsplus.com/articles/workflow/make-your-life-easier-build-a-level-editor/>>. Acesso em: 30 Julho 2013.

ROUSE, R. **Game Design: Theory & Practice**. 2ª. ed. Plano: Wordware Publishing, Inc, 2005. 698 p. ISBN 1556229127.

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

7 Apêndice A

Implementação do algoritmo A*

Esta implementação do algoritmo A* foi dividida em três classes: “AEstrela.java”, “Ponto.java” e “Lista.java”. As classes foram implementadas no pacote “br.ufla.americatribal.editor.aestrela”.

1. AEstrela.java

Esta é a classe principal do algoritmo. Ela armazena os dados sobre o mapa e os pontos iniciais e finais. Ela é a classe usada para interagir com classes externas.

Código:

```
package br.ufla.americatribal.editor.aestrela;

public class AEstrela {

    public static final String E = "E";
    public static final String D = "D";
    public static final String C = "C";
    public static final String B = "B";

    int mapa[][] = { { 1, 0, 0, 0, 0, 0 },
                     { 1, 0, 1, 0, 0, 0 },
                     { 1, 0, 0, 1, 0, 0 },
                     { 1, 0, 0, 1, 1, 0 },
                     { 3, 0, 0, 1, 0, 0 },
                     { 1, 0, 1, 0, 2, 0 } };

    int ty = mapa.length;
    int tx = mapa[0].length;
```



```

        Ponto casa = new Ponto(0, 0);
        Ponto origem = new Ponto();
        Ponto destino = new Ponto();
        Ponto atual = new Ponto();

        int obstaculo = 1;

        public AEstrela() {
        }

        public AEstrela(int mapa[][], Ponto origem, Ponto destino,
int obstaculo) {
            this.mapa = mapa;
            this.origem = origem;
            this.destino = destino;
            this.ty = mapa.length;
            this.tx = mapa[0].length;
            this.obstaculo = obstaculo;
        }

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

        public void visao(Lista l, Lista l2) {
            Ponto pc = new Ponto();
            Ponto pe = new Ponto();
            Ponto pd = new Ponto();
            Ponto pb = new Ponto();
            for (Ponto p : l.lista) {
                pc = p.cima();
                pe = p.esquerda();
                pd = p.direita();
                pb = p.baixo();
                if(!(l2.contem(p))){
                    l2.lista.add(p);
                }
            }
        }
    }

```

```

        if ((noMapa(pc)) && !(l2.contem(pc))) {
            l2.lista.add(pc);
        }
        if (!(l2.contem(pe))&&(noMapa(pe))) {
            l2.lista.add(pe);
        }
        if (!(l2.contem(pd))&&(noMapa(pd))) {
            l2.lista.add(pd);
        }
        if (!(l2.contem(pb))&&(noMapa(pb))) {
            l2.lista.add(pb);
        }
    }
}

public Lista visao(){
    Lista aux = new Lista(this.atual);
    Lista aux2 = new Lista();
    this.visao(aux, aux2);
    this.visao(aux2,aux);
    return aux;
}

public void desenhar() {
    System.out.print("Origem: ");
    origem.desenhar();

    System.out.print("Destino: ");
    destino.desenhar();

    System.out.println("tx: " + tx + " - ty: " + ty);
    for (int y = 0; y < ty; y++) {
        for (int x = 0; x < tx; x++) {
            System.out.print(" " + mapa[y][x]);
        }
        System.out.println();
    }
}

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

    if (mapa[y][x] == 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;
    }
}

```

2. Ponto.java

Esta classe é responsável por armazenar os pontos de um caminho feito pelo A*.

Código:

```
package br.ufla.americatribal.editor.aestrela;
```

```

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

    int g = 0;
    int h = 0;
    int f = 0;

    int custo = 10;

    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;
    }
    public void setPonto(int x, int y){
        this.x = x;
        this.y = y;
    }
    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);
    }
    public void desenhar(){
        System.out.print("(" + x + ", " + y + ") G: " + g + " H: " + h + " F: " + f);

        if (pai != null){
            System.out.print(" pai: (" + pai.x + ", " + pai.y + ")");
        }
        System.out.println();
    }
}

```

3. Lista.java

Esta classe armazena um caminho de pontos, e é nela que é feito o processamento do algoritmo.

Código:

```
package br.ufla.americatribal.editor.aestrela;

import java.util.ArrayList;

public class Lista {
    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(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 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 void desenhar() {
        for (Ponto p : lista) {
            p.desenhar();
        }
    }

    public static Lista adjacentes(Ponto p, AEstrela 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(AEstrela mapa) {
    Lista la = new Lista();
    Lista lf = new Lista();
    Lista adj = null;
    Lista caminho = null;

    la.adicionar(mapa.origem);

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

    la.adicionar(adj);

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

    while (!lf.tem(mapa.destino) && !la.lista.isEmpty())
    {
        Ponto atual = la.menor();
        if (atual == null)
            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(mapa.destino));
    while (p != null) {
        caminho.adicionar(p);
        p = p.pai;
    }
    caminho.inverter();

    return caminho;
}

public void conhece(Lista l) {
    for (Ponto p : l.lista) {
        if (!(this.lista.contains(p))) {
            this.lista.add(p);
        }
    }
}

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 void junta(Lista l){
    for(Ponto p: l.lista){

```

```

        this.lista.add(p);
    }
}
public void opcoes (AEstrela m){
    int[] v = new int[16];
    int i=0;
    for(Ponto p:this.lista){
        if(!(m.passavel(p))){
            v[i]=this.lista.indexOf(p);
            i++;
        }
    }
}
}
}

```

8 Apêndice B

Implementação do editor

A código do editor é separado em cinco classes: “Editor.java”, “TelaPrincipal.java”, “DesenhistaDaTelaPrincipal.java”, “Mapa.java” e “Conf.java”. Além das classes, são necessários dois arquivos externos, colocados dentro da pasta “assets/data/imagens” do projeto Android do libGDX: “américa-tribal-editor-imagens.pack” e “américa-tribal-editor-imagens.png”. Todo o código do editor foi colocado no pacote “br.ufla.americatribal.editor”, e também é necessário uma integração com o pacote “br.ufla.americatribal.editor.aestrela”, disponível no Apêndice A.

Arquivo “américa-tribal-editor-imagens.pack”: é um arquivo texto que descreve as posições das imagens dentro do arquivo “américa-tribal-editor-imagens.png”.

```
america-tribal-editor-imagens.png
format: RGBA8888
filter: Nearest,Nearest
repeat: none
aldeia-centro
  rotate: false
  xy: 0, 728
  size: 180, 180
  orig: 180, 180
  offset: 0, 0
  index: -1
aldeia-norte
  rotate: false
  xy: 0, 546
  size: 180, 180
  orig: 180, 180
  offset: 0, 0
  index: -1
aldeia-sul
```

```

    rotate: false
    xy: 182, 728
    size: 180, 180
    orig: 180, 180
    offset: 0, 0
    index: -1
aura-time-a
    rotate: false
    xy: 0, 364
    size: 180, 180
    orig: 180, 180
    offset: 0, 0
    index: -1
aura-time-b
    rotate: false
    xy: 182, 546
    size: 180, 180
    orig: 180, 180
    offset: 0, 0
    index: -1
cenario-agua
    rotate: false
    xy: 364, 728
    size: 180, 180
    orig: 180, 180
    offset: 0, 0
    index: -1
cenario-areia
    rotate: false
    xy: 0, 182
    size: 180, 180
    orig: 180, 180
    offset: 0, 0
    index: -1
cenario-floresta
    rotate: false
    xy: 182, 364
    size: 180, 180
    orig: 180, 180
    offset: 0, 0
    index: -1
cenario-grama
    rotate: false
    xy: 364, 546
    size: 180, 180

```

```
    orig: 180, 180
    offset: 0, 0
    index: -1
cenario-pedra
    rotate: false
    xy: 546, 728
    size: 180, 180
    orig: 180, 180
    offset: 0, 0
    index: -1
cenario-terra
    rotate: false
    xy: 0, 0
    size: 180, 180
    orig: 180, 180
    offset: 0, 0
    index: -1
construcao-observatorio
    rotate: false
    xy: 182, 182
    size: 180, 180
    orig: 180, 180
    offset: 0, 0
    index: -1
explorador-centro
    rotate: false
    xy: 364, 364
    size: 180, 180
    orig: 180, 180
    offset: 0, 0
    index: -1
explorador-norte
    rotate: false
    xy: 546, 546
    size: 180, 180
    orig: 180, 180
    offset: 0, 0
    index: -1
explorador-sul
    rotate: false
    xy: 728, 728
    size: 180, 180
    orig: 180, 180
    offset: 0, 0
    index: -1
```

objeto-carne
rotate: false
xy: 182, 0
size: 180, 180
orig: 180, 180
offset: 0, 0
index: -1
objeto-madeira
rotate: false
xy: 364, 182
size: 180, 180
orig: 180, 180
offset: 0, 0
index: -1
objeto-pedra
rotate: false
xy: 546, 364
size: 180, 180
orig: 180, 180
offset: 0, 0
index: -1
recurso-fazenda
rotate: false
xy: 728, 546
size: 180, 180
orig: 180, 180
offset: 0, 0
index: -1
recurso-floresta
rotate: false
xy: 910, 728
size: 180, 180
orig: 180, 180
offset: 0, 0
index: -1
recurso-mina
rotate: false
xy: 364, 0
size: 180, 180
orig: 180, 180
offset: 0, 0
index: -1
unidade-animal-centro
rotate: false
xy: 546, 182


```
size: 180, 180
orig: 180, 180
offset: 0, 0
index: -1
unidade-animal-norte
rotate: false
xy: 728, 364
size: 180, 180
orig: 180, 180
offset: 0, 0
index: -1
unidade-animal-sul
rotate: false
xy: 910, 546
size: 180, 180
orig: 180, 180
offset: 0, 0
index: -1
unidade-basico-centro
rotate: false
xy: 1092, 728
size: 180, 180
orig: 180, 180
offset: 0, 0
index: -1
unidade-basico-norte
rotate: false
xy: 546, 0
size: 180, 180
orig: 180, 180
offset: 0, 0
index: -1
unidade-basico-sul
rotate: false
xy: 728, 182
size: 180, 180
orig: 180, 180
offset: 0, 0
index: -1
unidade-distancia-centro
rotate: false
xy: 910, 364
size: 180, 180
orig: 180, 180
offset: 0, 0
```

```
    index: -1
unidade-distancia-norte
  rotate: false
  xy: 1092, 546
  size: 180, 180
  orig: 180, 180
  offset: 0, 0
  index: -1
unidade-distancia-sul
  rotate: false
  xy: 1274, 728
  size: 180, 180
  orig: 180, 180
  offset: 0, 0
  index: -1
unidade-mistico-centro
  rotate: false
  xy: 728, 0
  size: 180, 180
  orig: 180, 180
  offset: 0, 0
  index: -1
unidade-mistico-norte
  rotate: false
  xy: 910, 182
  size: 180, 180
  orig: 180, 180
  offset: 0, 0
  index: -1
unidade-mistico-sul
  rotate: false
  xy: 1092, 364
  size: 180, 180
  orig: 180, 180
  offset: 0, 0
  index: -1
unidade-mitico-centro
  rotate: false
  xy: 1274, 546
  size: 180, 180
  orig: 180, 180
  offset: 0, 0
  index: -1
unidade-mitico-norte
  rotate: false
```

```

xy: 1456, 728
size: 180, 180
orig: 180, 180
offset: 0, 0
index: -1
unidade-mitico-sul
  rotate: false
  xy: 910, 0
  size: 180, 180
  orig: 180, 180
  offset: 0, 0
  index: -1
vazio
  rotate: false
  xy: 1092, 182
  size: 180, 180
  orig: 180, 180
  offset: 0, 0
  index: -1

```

Arquivo “américa-tribal-editor-imagens.png”: é um arquivo de imagem onde estão armazenadas todas as imagens externas utilizadas no editor.



1. Editor.java

É a classe inicial do editor. Ela estende a classe abstrata “*Game*” do libGDX, que é uma implementação da interface “*ApplicationListener*”, interface responsável por gerenciar a aplicação como um todo.

Código:

```
package br.ufla.americatribal.editor;

import com.badlogic.gdx.Game;

public class Editor extends Game {

    @Override
    public void create() {
        setScreen(new TelaPrincipal());
    }
}
```

2. TelaPrincipal.java

É a classe responsável por gerenciar o editor. Ela implementa as interfaces “*Screen*”, que é responsável por gerenciar os elementos gráficos apresentados na tela; a interface “*InputProcessor*”, que gerencia os periféricos, como *mouse* e teclado; e a interface “*TextInputListener*”, que é responsável pelo gerenciamento das caixas de diálogo para interação com o usuário.

Código:

```

package br.ufla.americatribal.editor;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Input.TextInputListener;
import com.badlogic.gdx.InputProcessor;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.Input.Keys;
import com.badlogic.gdx.graphics.GL10;

public class TelaPrincipal implements Screen, InputProcessor,
    TextInputListener
{
    DesenhistaDaTelaPrincipal desenhista;
    Mapa mapa;
    public static int selecionado_chao = 0;
    public static int selecionado_objeto = 0;
    int desenhando = 0;

    int x_ant = 0;
    int y_ant = 0;
    int x_atu = 0;
    int y_atu = 0;

    boolean shift_pressionado = false;
    boolean control_pressionado = false;
    boolean movendo = false;
    static boolean caps_ligado = false;

    public static boolean botao1_objeto = false;

    public static int tribo_a = 0;
    public static int tribo_b = 2;
    public static int tribo_c = 1;

    public static String nomeMapa = "mapa-01";

    String entrada = "vazio";

    @Override
    public void show() {
        /* deixa na resolução da tela (praticamente tela
cheia)
        Gdx.graphics.setDisplayMode(

```

```

        Gdx.graphics.getWidth(),
        Gdx.graphics.getHeight(), true);
    */

    mapa = new Mapa();
    desenhista = new DesenhistaDaTelaPrincipal(mapa);

    Gdx.input.setInputProcessor(this);
}

@Override
public void render(float delta) {
    Gdx.gl.glClearColor(0.1f, 0.1f, 0.1f, 1);
    Gdx.gl.glClear(GL10.GL_COLOR_BUFFER_BIT);

    desenhista.desenhar(selecionado_chao,
        selecionado_objeto);
}

@Override
public boolean keyDown(int keycode) {
    if (keycode == Keys.LEFT){
        mapa.x--;
    }else if(keycode == Keys.RIGHT){
        mapa.x++;
    }
    if (keycode == Keys.UP){
        mapa.y++;
    }else if(keycode == Keys.DOWN){
        mapa.y--;
    }
    if (keycode == Keys.NUM_0){
        mapa.x = 0;
        mapa.y = 0;
    }
    if (keycode == Keys.SHIFT_LEFT || keycode ==
Keys.SHIFT_RIGHT){
        shift_pressionado = true;
    }

    if (keycode == Keys.CONTROL_LEFT || keycode ==
Keys.CONTROL_RIGHT){
        control_pressionado = true;

```

```

    }

    return true;
}

@Override
public boolean keyUp(int keycode) {
    if (shift_pressionado == caps_ligado){
        if (keycode == Keys.NUM_1){
            seleccionado_chao = 0;
        }else if (keycode == Keys.NUM_2){
            seleccionado_chao = 1;
        }else if (keycode == Keys.NUM_3){
            seleccionado_chao = 2;
        }else if (keycode == Keys.NUM_4){
            seleccionado_chao = 3;
        }else if (keycode == Keys.NUM_5){
            seleccionado_chao = 4;
        }else if (keycode == Keys.NUM_6){
            seleccionado_chao = 5;
        }
    }else{
        if (keycode == Keys.NUM_1){
            seleccionado_objeto = 0;
        }else if (keycode == Keys.NUM_2){
            seleccionado_objeto = 1;
        }else if (keycode == Keys.NUM_3){
            seleccionado_objeto = 2;
        }else if (keycode == Keys.NUM_4){
            seleccionado_objeto = 3;
        }else if (keycode == Keys.NUM_5){
            seleccionado_objeto = 4;
        }else if (keycode == Keys.NUM_6){
            seleccionado_objeto = 5;
        }else if (keycode == Keys.NUM_7){
            seleccionado_objeto = 6;
        }
        else if (keycode == Keys.NUM_8){
            seleccionado_objeto = 7;
        }else if (keycode == Keys.Q){
            seleccionado_objeto = Conf.ALDEIA_A;
        }else if (keycode == Keys.W){
            seleccionado_objeto = Conf.ALDEIA_B;
        }else if (keycode == Keys.E){
            seleccionado_objeto = Conf.EXPLORADOR_A;
        }
    }
}

```

```

        }else if (keycode == Keys.R){
            selecionado_objeto = Conf.EXPLORADOR_B;
        }
    }

    if (shift_preSSIONado == false){
        if (keycode == Keys.PAGE_DOWN){
            if (tribo_a == 2)
                desenhista.trocarTribo(0, 0);
            else
                desenhista.trocarTribo(0,
tribo_a + 1);
        }else if (keycode == Keys.PAGE_UP){
            if (tribo_a == 0)
                desenhista.trocarTribo(0, 2);
            else
                desenhista.trocarTribo(0,
tribo_a - 1);
        }
    }else{
        if (keycode == Keys.PAGE_DOWN){
            if (tribo_b == 2)
                desenhista.trocarTribo(1, 0);
            else
                desenhista.trocarTribo(1,
tribo_b + 1);
        }else if (keycode == Keys.PAGE_UP){
            if (tribo_b == 0)
                desenhista.trocarTribo(1, 2);
            else
                desenhista.trocarTribo(1,
tribo_b - 1);
        }
    }

    // aspas e apóstrofo (em cima do tab)
    //if (keycode == 68){
    //    caps_ligado = !caps_ligado;
    //}

    if (keycode == Keys.TAB){
        botao1_objeto = !botao1_objeto;
        caps_ligado = !caps_ligado;
    }

    if (keycode == Keys.NUM_0){

```



```

        desenhista.recolocarMapa();
    }

    /*
    if (keycode == Keys.PLUS){
        Conf.tam_bloco_atual *= 2;
    }else if (keycode == Keys.MINUS){
        Conf.tam_bloco_atual *= 0.5;
    }
    */

    if (keycode == Keys.SHIFT_LEFT || keycode ==
Keys.SHIFT_RIGHT){
        shift_pressionado = false;
    }
    if (keycode == Keys.CONTROL_LEFT || keycode ==
Keys.CONTROL_RIGHT){
        control_pressionado = false;
    }
    return true;
}

@Override
public boolean keyTyped(char character) {
    // TODO Auto-generated method stub
    return false;
}

@Override
public boolean touchDown(int screenX, int screenY, int
pointer, int button) {
    //Gdx.app.log("mouse", "botao "+button+", pointer
"+pointer);
    x_ant = x_atu;
    y_ant = y_atu;
    x_atu = screenX;
    y_atu = screenY;

    desenhista.pegarMouse(screenX, screenY);

    if (control_pressionado == false){
        if (button == 0){
            desenhando = 1;
            desenhando();
        }else if (button == 1){

```

```

        desenhando = 2;
        desenhando();
    }else if (button == 2){
        movendo = true;
        desenhista.moverMapa(x_atu - x_ant,
y_atu - y_ant);
    }
    }
    return true;
}

@Override
public boolean touchUp(int screenX, int screenY, int
pointer, int button) {
    desenhista.pegarMouse(screenX, screenY);
    desenhando = 0;
    desenhista.selecionarItemObjetos(button);

    if (button == 0){
        if (desenhista.botaoRenomearMapa()){
            entrada = "mapa-nome";
            Gdx.input.getTextInput(this, "Renomear
Mapa", "mapa-");
        }else if (desenhista.botaoTamanhoX()){
            entrada = "mapa-x";
            Gdx.input.getTextInput(this, "Tamanho
Horizontal do Mapa (" +
                Mapa.MIN_X + " a
                "+Mapa.MAX_X+")", "" +Mapa.MIN_X);
        }else if (desenhista.botaoTamanhoY()){
            entrada = "mapa-y";
            Gdx.input.getTextInput(this, "Tamanho
Vertical do Mapa (" +
                Mapa.MIN_Y + " a
                "+Mapa.MAX_Y+")", "" +Mapa.MIN_Y);
        }else if (desenhista.botaoLimpar()){
            entrada = "mapa-limpar";
            Gdx.input.getTextInput(this,
"Alterações não salvas do mapa"
                + " atual serão perdidas",
                "Apenas OK ou Cancelar");
        }else if (desenhista.botaoSalvar()){
            entrada = "mapa-salvar";
            Gdx.input.getTextInput(this, "Escolha o
nome para"

```

```

        + " salvar o mapa:
", nomeMapa);
    }else if (desenhista.botaoCarregar()){
        entrada = "mapa-carregar";
        Gdx.input.getTextInput(this, "Escolha o
nome do mapa"
        + " que quer carregar: ",
        nomeMapa);
    }else if (desenhista.botaoVerificar()){
        entrada = "mapa-verificar";
        Gdx.input.getTextInput(this, "Verificar
se o mapa é eficaz?",
        "Se existe caminho entre
as tribos.");
    }
}

return true;
}

@Override
public boolean touchDragged(int screenX, int screenY, int
pointer) {
    x_ant = x_atu;
    y_ant = y_atu;
    x_atu = screenX;
    y_atu = screenY;

    desenhista.pegarMouse(screenX, screenY);

    if (control_pressionado == false){
        if (desenhando > 0){
            desenhista.desenhar();
        }else if (movendo == true){
            desenhista.moverMapa(x_atu - x_ant,
y_atu - y_ant);
        }
    }
    return true;
}

@Override
public boolean mouseMoved(int screenX, int screenY) {
    x_ant = x_atu;
    y_ant = y_atu;

```

```

        x_atu = screenX;
        y_atu = screenY;

        desenhista.pegarMouse(screenX, screenY);

        if (control_pressionado == true){
            desenhista.moverMapa(x_atu - x_ant, y_atu -
y_ant);
        }
        return true;
    }

    @Override
    public boolean scrolled(int amount) {
        //Gdx.app.log("Scroll", "q: "+amount);
        if (amount == 1){
            if (shift_pressionado == !caps_ligado){
                selecionado_objeto++;
                if (selecionado_objeto >
Conf.MAX_ITENS_OBJ - 1)
                    selecionado_objeto = 0;
            }else{
                selecionado_chao++;
                if (selecionado_chao >
Conf.MAX_ITENS_CHAO - 1)
                    selecionado_chao = 0;
            }
        }
        if (amount == -1){
            if (shift_pressionado == !caps_ligado){
                selecionado_objeto--;
                if (selecionado_objeto < 0)
                    selecionado_objeto =
Conf.MAX_ITENS_OBJ - 1;
            }else{
                selecionado_chao--;
                if (selecionado_chao < 0 )
                    selecionado_chao =
Conf.MAX_ITENS_CHAO - 1;
            }
        }
        return true;
    }
}

```

```

@Override
public void resize(int width, int height) {
    // TODO Auto-generated method stub

}

@Override
public void hide() {
    // TODO Auto-generated method stub

}

@Override
public void pause() {
    // TODO Auto-generated method stub

}

@Override
public void resume() {
    // TODO Auto-generated method stub

}

@Override
public void dispose() {
    // TODO Auto-generated method stub

}

private void desenhar(){
    // botao 1 pressionado
    if (desenhando == 1){
        if (botao1_objeto == false)

desenhista.mudarMapaChao(seleccionado_chao);
        else

desenhista.mudarMapaObjeto(seleccionado_objeto);
    }else if (desenhando == 2){
        if (botao1_objeto == false)

desenhista.mudarMapaObjeto(seleccionado_objeto);
        else

```

```

        desenhista.mudarMapaChao(selecionado_chao);
    }

}

public static void trocarSelecionado(boolean b1obj) {
    botao1_objeto = b1obj;
    caps_ligado = b1obj;
}

@Override
public void input(String text) {
    if (entrada == "mapa-nome"){
        nomeMapa = new String(text);
    }else if (entrada == "mapa-x"){
        int novo_tamanho = new Integer(text);
        mapa.mudarTamanho(novo_tamanho, mapa.tam_y,
false);
    }else if (entrada == "mapa-y"){
        int novo_tamanho = new Integer(text);
        mapa.mudarTamanho(mapa.tam_x, novo_tamanho,
false);
    }else if (entrada == "mapa-limpar"){
        mapa.nome = nomeMapa;
        mapa.mudarTamanho(mapa.tam_x, mapa.tam_y,
true);

        desenhista.recolocarMapa();
    }else if (entrada == "mapa-salvar"){
        mapa.nome = text;
        mapa.salvar();
        if (mapa.verificarEficacia()){
            mapa.nome = text;
            mapa.salvar();
            Gdx.app.Log("MAPA", "Mapa salvo com
sucesso!");
            desenhista.adicionarMensagem("Mapa
\""+mapa.nome+"\" salvo.");
        }else{
            Gdx.app.Log("MAPA", "Mapa salvo. NÃO
eficaz.");
            desenhista.adicionarMensagem("Mapa
\""+mapa.nome+"\" salvo,"
+ " porém NÃO é eficaz.");
        }
    }
}

```

```

        }else if (entrada == "mapa-carregar"){
            boolean carregou = mapa.carregar(text);
            if (carregou == true){
                Gdx.app.Log("MAPA", "Mapa carregado com
sucesso.");
                desenhista.adicionarMensagem("Mapa
carregado com sucesso.");
            }else{
                Gdx.app.Log("MAPA", "Mapa não pode ser
carregado.");
                desenhista.adicionarMensagem("Mapa não
pode ser carregado.");
            }
        }if (entrada == "mapa-verificar"){
            if (mapa.verificarEficacia()){
                Gdx.app.Log("MAPA", "O mapa é
eficaz!");
                desenhista.adicionarMensagem("O mapa é
eficaz!");
            }else{
                Gdx.app.Log("MAPA", "O mapa NÃO é
eficaz.");
                desenhista.adicionarMensagem("O mapa
NÃO é eficaz.");
            }
        }
    }

    @Override
    public void canceled() {
        // TODO Auto-generated method stub
    }
}

```

3. DesenhistaDaTelaPrincipal.java

Esta classe apenas carrega as imagens e as desenha no local correto na tela. Ela é separada da classe “TelaPrincipal”, pois caso seja necessário alterar os

métodos de desenho ou carregamento de imagem, a classe “TelaPrincipal” continua inalterada.

Código:

```
package br.ufla.americatribal.editor;

import java.util.Calendar;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.files.FileHandle;
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;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer.ShapeType;
import com.badlogic.gdx.math.Rectangle;

public class DesenhistaDaTelaPrincipal {
    private Rectangle cenario;
    private Rectangle objetos;
    //private Rectangle outros;

    private SpriteBatch batch;
    private ShapeRenderer debugRender;

    private OrthographicCamera camera;

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

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

    private TextureRegion [] aura;

    //private TextureRegion [][] unidades;
```



```

private Mapa mapa;

private int xNoMapa = -1;
private int yNoMapa = -1;

private int mousex = 0;
private int mousey = 0;

private float desloxc = 0;
private float deslocy = 0;

BitmapFont font;

// mensagens
//private static final int maxMensagens = 5;
private String mensagem = "";
private static final int TEMPO_MAX_MENSAGEM = 200;
private int tempoMensagem = TEMPO_MAX_MENSAGEM + 1;
//private int mensagemAtual = 0;

// carregamento
public DesenhistaDaTelaPrincipal(Mapa mapa){
    camera = new OrthographicCamera(Config.CAM_TAM_LAR,
Config.CAM_TAM_ALT);
    camera.position.set(Config.CAM_TAM_LAR * 0.5f,
Config.CAM_TAM_ALT * 0.5f, 0);
    camera.update();

    batch = new SpriteBatch();
    debugRender = new ShapeRenderer();

    this.mapa = mapa;

    carregarTexturas();

    font = new BitmapFont();
    font.setScale(0.03f);
    font.setUseIntegerPositions(false);

}

private void carregarTexturas(){
    criarAreas();

```

```

TextureAtlas atlas = new TextureAtlas(
Gdx.files.internal(Conf.PACOTE_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[TelaPrincipal.tribo_a];
cenario_objetos[9] = aldeia[TelaPrincipal.tribo_b];
cenario_objetos[10]=
explorador[TelaPrincipal.tribo_a];
cenario_objetos[11]=
explorador[TelaPrincipal.tribo_b];

/*
unidades = new TextureRegion[3][];
unidades[0] = new TextureRegion[6];
unidades[0][0] = atlas.findRegion("vazio");
unidades[0][1] = atlas.findRegion("unidade-basico-
norte");
unidades[0][2] = atlas.findRegion("unidade-
distancia-norte");
unidades[0][3] = atlas.findRegion("unidade-animal-
norte");
unidades[0][4] = atlas.findRegion("unidade-mistico-
norte");
unidades[0][5] = atlas.findRegion("unidade-mitico-
norte");

unidades[1] = new TextureRegion[6];
unidades[1][0] = atlas.findRegion("vazio");
unidades[1][1] = atlas.findRegion("unidade-basico-
centro");
unidades[1][2] = atlas.findRegion("unidade-
distancia-centro");
unidades[1][3] = atlas.findRegion("unidade-animal-
centro");
unidades[1][4] = atlas.findRegion("unidade-mistico-
centro");
unidades[1][5] = atlas.findRegion("unidade-mitico-
centro");

unidades[2] = new TextureRegion[6];
unidades[2][0] = atlas.findRegion("vazio");
unidades[2][1] = atlas.findRegion("unidade-basico-
sul");
unidades[2][2] = atlas.findRegion("unidade-
distancia-sul");

```

```

sul");
        unidades[2][3] = atlas.findRegion("unidade-animal-
sul");
        unidades[2][4] = atlas.findRegion("unidade-mistico-
sul");
        unidades[2][5] = atlas.findRegion("unidade-mitico-
sul");
        */
    }

    private void criarAreas(){
        float x, lar, y, alt;
        x = 0.1f;
        y = 0.1f;
        lar = Conf.CAM_TAM_LAR * 0.8f;
        alt = Conf.CAM_TAM_ALT - 0.2f;
        cenario = new Rectangle(x, y, lar, alt);

        x = Conf.CAM_TAM_LAR * 0.8f;
        y = 0.1f;
        lar = Conf.CAM_TAM_LAR * 0.2f - 0.1f;
        alt = Conf.CAM_TAM_ALT - 0.2f;
        objetos = new Rectangle(x, y, lar, alt);
    }

    // desenho
    public void desenhar(int sel_chao, int sel_obj){
        batch.setProjectionMatrix(camera.combined);
        batch.begin();
            desenharChao();
            desenharObjetos();
        batch.end();
        desenharRetangulos();
        batch.begin();
            desenharInterface(sel_chao, sel_obj);
            pegarPosicaoNoMapa();
            desenharTexto();
        batch.end();
        desenharRetanguloSelecaoObjetos();
    }

    private void desenharTexto() {
        // mensagens
        mostrarMensagens();
    }

```

```

0.2f;

// nome do mapa
float x = objetos.x + 0.2f;
float y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO * 3 -

float ty = font.getCapHeight();

font.setScale(0.025f);
font.setColor(1, 1, 1, 1);
font.draw(batch, "Mapa: ", x, y);

x += 1.1f;
font.setColor(0.9f, 0.9f, 0.5f, 1);
font.draw(batch, TelaPrincipal.nomeMapa, x, y);

y -= ty + 0.3f;
x = objetos.x + 0.2f + 1.2f;
font.setScale(0.02f);
font.setColor(0,1,1,1);
font.draw(batch, "Renomear Mapa", x, y);

// tamanho
x = objetos.x + 0.2f;
y -= 1f;
font.setFixedWidthGlyphs("0123456789");
font.setColor(1,1,1,1);
font.draw(batch, "Tamanho   hor:           ver:",

x, y);

x += 2.05f;
font.setColor(1,1,0,1);
font.draw(batch, (""+mapa.tam_x), x, y);

x += 1.55f;
font.setColor(1,1,0,1);
font.draw(batch, (""+mapa.tam_y), x, y);

// limpar mapa
x = objetos.x + 0.2f + 1.35f;
y -= 1.05f;
font.setColor(1,0.6f,0.6f,1);
font.draw(batch, "Novo Mapa", x, y);

// salvar e carregar
x = objetos.x + 0.2f;
y -= 1.2f;

```

```

        x += 0.65f;
        font.setColor(1,1,0,0.7f);
        font.draw(batch, "Salvar", x, y);

        x += 2.15f;
        font.setColor(1,1,0,0.7f);
        font.draw(batch, "Carregar", x, y);

        // verificador de eficácia
        y -= 1.2f;
        x = objetos.x + 0.2f + 1.2f;
        font.setColor(1,1,1,0.7f);
        font.draw(batch, "Verificar Eficácia", x, y);
    }

    public boolean botaoVerificar(){
        float x = objetos.x + 0.2f + 0.7f;
        float y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO * 3 -
1.2f - 1.2f - 1.1f - 2.1f;
        float tx = 3f;
        float ty = 0.5f;

        float mx = mousex * Conf.unidades_x;
        float my = mousey * Conf.unidades_y;

        if (mx >= x && mx <= x+tx && my >= y && my <= y +
ty){
            return true;
        }
        return false;
    }

    public boolean botaoSalvar() {
        float x = objetos.x + 0.3f;
        float y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO * 3 -
1.1f - 2.1f - 1.2f;
        float tx = 1.9f;
        float ty = 0.5f;

        float mx = mousex * Conf.unidades_x;
        float my = mousey * Conf.unidades_y;

        if (mx >= x && mx <= x+tx && my >= y && my <= y +
ty){

```

```

        return true;
    }
    return false;
}

public boolean botaoCarregar() {
    float x = objetos.x + 0.3f + 2.3f;
    float y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO * 3 -
1.1f - 2.1f - 1.2f;
    float tx = 1.9f;
    float ty = 0.5f;

    float mx = mousex * Conf.unidades_x;
    float my = mousey * Conf.unidades_y;

    if (mx >= x && mx <= x+tx && my >= y && my <= y +
ty){
        return true;
    }

    return false;
}

public boolean botaoLimpar(){
    float x = objetos.x + 0.2f + 0.7f;
    float y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO * 3 -
1.1f - 2.1f;
    float tx = 3f;
    float ty = 0.5f;

    float mx = mousex * Conf.unidades_x;
    float my = mousey * Conf.unidades_y;

    if (mx >= x && mx <= x+tx && my >= y && my <= y +
ty){
        return true;
    }
    return false;
}

public boolean botaoTamanhoX(){
    float x = objetos.x + 1.55f;
    float y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO * 3 -
2.15f;
    float ty = 0.5f;

```

```

float tx = 1.2f;

float mx = mousex * Conf.unidades_x;
float my = mousey * Conf.unidades_y;

if (mx >= x && mx <= x+tx && my >= y && my <= y +
ty){
    return true;
}
return false;
}

public boolean botaoTamanhoY(){
float x = objetos.x + 1.55f + 1.55f;
float y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO * 3 -
2.15f;

float ty = 0.5f;
float tx = 1.2f;

float mx = mousex * Conf.unidades_x;
float my = mousey * Conf.unidades_y;

if (mx >= x && mx <= x+tx && my >= y && my <= y +
ty){
    return true;
}
return false;
}

public boolean botaoRenomearMapa(){
float x = objetos.x + 0.2f + 0.75f;
float ty = 0.4f;
float y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO * 3 -
1.1f;

float tx = 3;

float mx = mousex * Conf.unidades_x;
float my = mousey * Conf.unidades_y;

if (mx >= x && mx <= x+tx && my >= y && my <= y +
ty){
    return true;
}
return false;
}

```



```

    }

    private void desenharInterface(int sel_chao, int sel_obj){
        if (sel_chao < Conf.MAX_ITENS_CHAO && sel_chao >=
0){
            float x = objetos.x + 0.1f;
            float y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO -
0.2f;

            batch.draw(cenario_chao[sel_chao], x, y,
Conf.TAM_BLOCO,
                Conf.TAM_BLOCO);
        }

        if (sel_obj >= 0 && sel_obj < Conf.MAX_ITENS_OBJ){
            float x = objetos.x + 0.1f;
            float y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO *
2 - 0.4f;

            batch.draw(cenario_objetos[sel_obj], x, y,
Conf.TAM_BLOCO,
                Conf.TAM_BLOCO);

            if (sel_obj == Conf.ALDEIA_A || sel_obj ==
Conf.EXPLORADOR_A)
                batch.draw(aura[1], x,
y, Conf.TAM_BLOCO, Conf.TAM_BLOCO);

            if (sel_obj == Conf.ALDEIA_B || sel_obj ==
Conf.EXPLORADOR_B)
                batch.draw(aura[2], x,
y, Conf.TAM_BLOCO, Conf.TAM_BLOCO);
        }

        float tam = Conf.TAM_BLOCO * 0.45f;
        float x = objetos.x + 0.2f + Conf.TAM_BLOCO + 0.2f;
        float y = Conf.CAM_TAM_ALT - tam - 0.2f;
        for (int i = 0; i < 6; i++){
            batch.draw(cenario_chao[i], x + i *
(tam+0.1f), y, tam, tam);
        }

        y = Conf.CAM_TAM_ALT - tam - 0.4f - Conf.TAM_BLOCO;
        for (int i = 0; i < 6; i++){
            batch.draw(cenario_objetos[i], x + i *
(tam+0.1f), y, tam, tam);
        }
    }

```

```

        y -= tam + 0.1f;
        for (int i = 0; i < 6; i++){
            batch.draw(cenario_objetos[i+6], x + i *
(tam+0.1f),y, tam, tam);
        }
        batch.draw(aura[1], x + 2 * (tam+0.1f),y, tam, tam);
        batch.draw(aura[2], x + 3 * (tam+0.1f),y, tam, tam);
        batch.draw(aura[1], x + 4 * (tam+0.1f),y, tam, tam);
        batch.draw(aura[2], x + 5 * (tam+0.1f),y, tam, tam);
    }

    private void desenharChao(){
        for (int x = 0; x < mapa.tam_x; x++){
            for (int y = 0; y < mapa.tam_y; y++){
                if (x + mapa.x > -2 && x + mapa.x <

cenario.x+cenario.width+mapa.tam_x + 1 &&
                    y + mapa.y > -2 && y + mapa.y <

cenario.y+cenario.height+mapa.tam_y+ 1)
                {
                    TextureRegion chao =
cenario_chao[mapa.chao[x][y]];
                    batch.draw(chao, x + mapa.x +
cenario.x + desloxc,
                                y +
mapa.y + cenario.y + deslocy,
                                Conf.TAM_BLOCO,
                                Conf.TAM_BLOCO);
                }
            }
        }

        private void desenharObjetos(){
            for (int x = 0; x < mapa.tam_x; x++){
                for (int y = 0; y < mapa.tam_y; y++){
                    if (x + mapa.x > -2 && x + mapa.x <

cenario.x+cenario.width+mapa.tam_x + 1 &&
                        y + mapa.y > -2 && y + mapa.y <

cenario.y+cenario.height+mapa.tam_y+ 1)
                    {
                        if (mapa.objeto[x][y] > 0){

```

```

cenario_objetos[mapa.objeto[x][y]];
TextureRegion obj =
batch.draw(obj, x + mapa.x
+ cenario.x + deslocx,
y + mapa.y + cenario.y + deslocoy,
Conf.TAM_BLOCO, Conf.TAM_BLOCO);

if (mapa.objeto[x][y] ==
Conf.ALDEIA_A ||
mapa.objeto[x][y] == Conf.EXPLORADOR_A)
batch.draw(aura[1],
x + mapa.x + cenario.x + deslocx,
y +
mapa.y + cenario.y + deslocoy,
Conf.TAM_BLOCO, Conf.TAM_BLOCO);

if (mapa.objeto[x][y] ==
Conf.ALDEIA_B ||
mapa.objeto[x][y] == Conf.EXPLORADOR_B)
batch.draw(aura[2],
x + mapa.x + cenario.x + deslocx,
y +
mapa.y + cenario.y + deslocoy,
Conf.TAM_BLOCO, Conf.TAM_BLOCO);
    }
    }
}

public void mudarMapaChao(int sel){
    if (xNoMapa > -1 && yNoMapa > -1){
        mapa.mudarChao(xNoMapa, yNoMapa, sel);
    }
}

public void mudarMapaObjeto(int sel){
    if (xNoMapa > -1 && yNoMapa > -1){
        mapa.mudarObjeto(xNoMapa, yNoMapa, sel);
    }
}

```

```

    }

    private void desenharRetangulos(){
        debugRender.setProjectionMatrix(camera.combined);
        debugRender.begin(ShapeType.FilledRectangle);
        //debugRender.setColor(1, 0, 0, 1);

        //debugRender.filledRect(mapab.x,mapab.y,mapab.width,mapab
        .height);
        //debugRender.setColor(1, 1, 1, 1);
        //debugRender.filledRect(objetos.x-
        0.1f,objetos.y-0.1f,
        //
        objetos.width+0.2f,objetos.height+0.2f);
        debugRender.setColor(0.2f, 0.2f, 0.4f, 1);

        debugRender.filledRect(objetos.x,objetos.y,objetos.width,o
        bjetos.height);
        //debugRender.setColor(0, 0, 1, 1);

        //debugRender.filledRect(outros.x,outros.y,outros.width,ou
        tros.height);

        // retangulos na parte de objetos
        {
            float tam = Conf.TAM_BLOCO * 0.45f;
            float x = objetos.x + 0.1f;
            float y = Conf.CAM_TAM_ALT -
            Conf.TAM_BLOCO * 2 - 0.4f;
            debugRender.setColor(1, 1, 1, 1);

            debugRender.filledRect(x,y,Conf.TAM_BLOCO,
            Conf.TAM_BLOCO);

            x = objetos.x + 0.2f + Conf.TAM_BLOCO +
            0.2f;
            y = Conf.CAM_TAM_ALT - tam - 0.4f -
            Conf.TAM_BLOCO;
            for (int i = 0; i < 6; i++){
                debugRender.filledRect(x + i *
            (tam+0.1f),y, tam, tam);
            }
            y -= tam + 0.1f;
            for (int i = 0; i < 6; i++){

```

```

                                debugRender.filledRect(x + i *
(tam+0.1f),y, tam, tam);
                                }
                                }
                                debugRender.end();

                                // retangulo de seleção
                                if (xNoMapa > -1 && yNoMapa > -1){
                                    float x = xNoMapa + cenario.x + mapa.x +
desloxx;
                                    float y = yNoMapa + cenario.y + mapa.y +
desloxy;
                                    debugRender.begin(ShapeType.Rectangle);
                                    debugRender.setColor(1, 1, 1, 1);
                                    debugRender.rect(x, y, Conf.TAM_BLOCO,
Conf.TAM_BLOCO);
                                    debugRender.setColor(0, 0, 0, 1);
                                    debugRender.rect(x+0.02f, y+0.02f,
Conf.TAM_BLOCO-0.04f,
Conf.TAM_BLOCO-0.04f);
                                    debugRender.end();
                                }

                                // desenhar botão de trocar nome mapa, de limpar
mapa,
                                // de salvar e carregar
                                float x = objetos.x + 0.2f + 0.75f;
                                float ty = 0.4f;
                                float y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO * 3 -
1.1f;
                                float tx = 3;

                                debugRender.begin(ShapeType.FilledRectangle);
                                debugRender.setColor(0.3f, 0.8f, 0.2f, 1);
                                debugRender.filledRect(x, y, tx, ty);
                                debugRender.setColor(0.2f, 0.5f, 0.2f, 0.5f);
                                debugRender.filledRect(x+0.03f, y+0.03f, tx-
0.06f, ty-0.06f);

                                // limpar mapa
                                x = objetos.x + 0.2f + 0.7f;
                                y -= 2.1f;
                                tx = 3f;
                                ty = 0.5f;
                                debugRender.setColor(0.8f, 0.3f, 0.2f, 1);

```

```

debugRender.filledRect(x, y, tx, ty);
debugRender.setColor(0.5f, 0.2f, 0.2f, 0.5f);
debugRender.filledRect(x+0.03f, y+0.03f, tx-
0.06f, ty-0.06f);

// carregar e salvar
x = objetos.x + 0.3f;
y -= 1.2f;
tx = 1.9f;
ty = 0.5f;

debugRender.setColor(0.3f, 0.8f, 0.2f, 1);
debugRender.filledRect(x, y, tx, ty);
debugRender.setColor(0.2f, 0.5f, 0.2f, 0.5f);
debugRender.filledRect(x+0.03f, y+0.03f, tx-
0.06f, ty-0.06f);

x += 2.3f;
debugRender.setColor(0.2f, 0.8f, 0.8f, 1);
debugRender.filledRect(x, y, tx, ty);
debugRender.setColor(0.2f, 0.5f, 0.5f, 0.5f);
debugRender.filledRect(x+0.03f, y+0.03f, tx-
0.06f, ty-0.06f);

// eficacia
x = objetos.x + 0.2f + 0.7f;
y -= 1.2f;
tx = 3f;
ty = 0.5f;
debugRender.setColor(0.8f, 0.8f, 0.2f, 1);
debugRender.filledRect(x, y, tx, ty);
debugRender.setColor(0.5f, 0.5f, 0.2f, 0.5f);
debugRender.filledRect(x+0.03f, y+0.03f, tx-
0.06f, ty-0.06f);

debugRender.end();

// desenhar botões de tamanho
x = objetos.x + 1.55f;
y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO * 3 - 2.135f;
ty = 0.5f;
tx = 1.2f;
debugRender.begin(ShapeType.Rectangle);
debugRender.setColor(0,1,0.5f,1);
debugRender.rect(x,y,tx,ty);

```

```

        debugRender.rect(x+1.55f,y,tx,ty);
        debugRender.end();

    }

    private void desenharRetanguloSelecaoObjetos(){
        float x = objetos.x + 0.1f;
        float y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO - 0.2f;

        if (TelaPrincipal.botao1_objeto == true)
            y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO * 2 -
0.4f;

        debugRender.begin(ShapeType.FilledRectangle);
        debugRender.setColor(0, 0, 0, 1);
        debugRender.filledRect(0, 0,
Conf.CAM_TAM_LAR, 0.1f);
        debugRender.filledRect(0, Conf.CAM_TAM_ALT-
0.1f, Conf.CAM_TAM_LAR, 0.1f);
        debugRender.filledRect(0, 0, 0.1f,
Conf.CAM_TAM_ALT);
        debugRender.filledRect(Conf.CAM_TAM_LAR-0.1f,
0, 0.1f, Conf.CAM_TAM_ALT);
        debugRender.filledRect(objetos.x, 0, 0.1f,
Conf.CAM_TAM_ALT);
        debugRender.end();

        debugRender.begin(ShapeType.Rectangle);
        debugRender.setColor(1, 0, 0, 1);
        debugRender.rect(x, y, Conf.TAM_BLOCO,
Conf.TAM_BLOCO);
        debugRender.setColor(0, 0, 0, 1);
        debugRender.rect(x+0.02f, y+0.02f,
Conf.TAM_BLOCO-0.04f,
Conf.TAM_BLOCO-0.04f);
        debugRender.setColor(1, 1, 0, 1);
        debugRender.rect(x+0.04f, y+0.04f,
Conf.TAM_BLOCO-0.08f,
Conf.TAM_BLOCO-0.08f);
        debugRender.end();
    }

    private void pegarPosicaoNoMapa() {
        xNoMapa = -1;
        yNoMapa = -1;
    }

```

```

        if (mousex * Conf.unidades_x >= cenario.x &&
            mousex * Conf.unidades_x < cenario.x +
cenario.width &&
            mousey * Conf.unidades_y >= cenario.y &&
            mousey * Conf.unidades_y < cenario.y +
cenario.height)
        {
            xNoMapa = (int) Math.floor(mousex *
Conf.unidades_x -
            mapa.x - cenario.x - deslocx);
            yNoMapa = (int) Math.floor(mousey *
Conf.unidades_y -
            mapa.y - cenario.y - deslocy);

            if (xNoMapa >= mapa.tam_x || xNoMapa < 0){
                xNoMapa = -1;
                yNoMapa = -1;
            }
            if (yNoMapa >= mapa.tam_y || yNoMapa < 0){
                xNoMapa = -1;
                yNoMapa = -1;
            }

            //Gdx.app.log("Coisa", "x: "+xNoMapa+ ", y:
"+yNoMapa);
        }
    }

    public void pegarMouse(int sx, int sy){
        mousex = sx;
        mousey = Conf.TELA_RES_ALT - sy;
    }

    public void moverMapa(int dx, int dy) {
        float desx = dx * Conf.unidades_x;
        float desy = dy * Conf.unidades_y;

        deslocx += desx;
        deslocy -= desy;

        if (deslocx >= Conf.TAM_BLOCO){
            deslocx = 0;
            mapa.x++;
        }
        if (deslocx <= -Conf.TAM_BLOCO){

```



```

        deslocx = 0;
        mapa.x--;
    }

    if (desloxy >= Conf.TAM_BLOCO){
        desloxy = 0;
        mapa.y++;
    }
    if (desloxy <= -Conf.TAM_BLOCO){
        desloxy = 0;
        mapa.y--;
    }
}

public void recolocarMapa() {
    mapa.x = 0;
    mapa.y = 0;
    deslocx = 0;
    desloxy = 0;
}

public void selecionarItemObjetos(int botao){
    float mx = mousex * Conf.unidades_x;
    float my = mousey * Conf.unidades_y;
    if (mx >= objetos.x &&
        mx < objetos.x + objetos.width &&
        my >= objetos.y &&
        my < objetos.y + objetos.height)
    {
        float x = objetos.x + 0.1f;
        float y = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO -
0.2f;
        float y2 = Conf.CAM_TAM_ALT - Conf.TAM_BLOCO
* 2 - 0.4f;

        if (mx >= x && mx <= x + Conf.TAM_BLOCO && my
>= y &&
            my <= y + Conf.TAM_BLOCO)
        {
            TelaPrincipal.trocarSelecionado(false);
            return;
        }
        else if (mx >= x && mx <= x + Conf.TAM_BLOCO
&& my >= y2 &&
            my <= y2 + Conf.TAM_BLOCO)

```

```

        {
            TelaPrincipal.trocarSelecioneado(true);
            return;
        }

        float tam = Conf.TAM_BLOCO * 0.45f;
        x = objetos.x + 0.2f + Conf.TAM_BLOCO + 0.2f;
        y = Conf.CAM_TAM_ALT - tam - 0.2f;
        for (int i = 0; i < 6; i++){
            float px = x + i * (tam+0.1f);
            if (mx >= px && mx <= px + tam && my >=
y && my <= y + tam){
                TelaPrincipal.selecionado_chao =
i;

                if (botao == 0)

                    TelaPrincipal.trocarSelecioneado(false);
                    return;
            }
        }

        y = Conf.CAM_TAM_ALT - tam - 0.4f -
Conf.TAM_BLOCO;
        for (int i = 0; i < 6; i++){
            float px = x + i * (tam+0.1f);
            if (mx >= px && mx < px + tam && my >=
y && my < y + tam){
                TelaPrincipal.selecionado_objeto
= i;

                if (botao == 0)

                    TelaPrincipal.trocarSelecioneado(true);
                    return;
            }
        }
        y -= tam + 0.1f;
        for (int i = 0; i < 6; i++){
            float px = x + i * (tam+0.1f);
            if (mx >= px && mx < px + tam && my >=
y && my < y + tam){
                TelaPrincipal.selecionado_objeto
= i + 6;

                if (botao == 0)

                    TelaPrincipal.trocarSelecioneado(true);

```

```

        return;
    }
}

}

    public void trocarTribo(int tribo, int novaTribo){
        if (tribo == 0){
            if (novaTribo == TelaPrincipal.tribo_a)
                return;
            if (novaTribo != TelaPrincipal.tribo_b){
                TelaPrincipal.tribo_c =
TelaPrincipal.tribo_a;
                TelaPrincipal.tribo_a = novaTribo;
            }else{
                TelaPrincipal.tribo_b =
TelaPrincipal.tribo_c;
                TelaPrincipal.tribo_c =
TelaPrincipal.tribo_a;
                TelaPrincipal.tribo_a = novaTribo;
            }
        }else if (tribo == 1){
            if (novaTribo == TelaPrincipal.tribo_b)
                return;
            if (novaTribo != TelaPrincipal.tribo_a){
                TelaPrincipal.tribo_c =
TelaPrincipal.tribo_b;
                TelaPrincipal.tribo_b = novaTribo;
            }else{
                TelaPrincipal.tribo_a =
TelaPrincipal.tribo_c;
                TelaPrincipal.tribo_c =
TelaPrincipal.tribo_b;
                TelaPrincipal.tribo_b = novaTribo;
            }
        }

        cenario_objetos[Conf.ALDEIA_A] =
aldeia[TelaPrincipal.tribo_a];
        cenario_objetos[Conf.ALDEIA_B] =
aldeia[TelaPrincipal.tribo_b];
        cenario_objetos[Conf.EXPLORADOR_A] =
explorador[TelaPrincipal.tribo_a];
        cenario_objetos[Conf.EXPLORADOR_B] =
explorador[TelaPrincipal.tribo_b];
    }
}

```

```

    }

    public void adicionarMensagem(String mensagem){
        this.mensagem = mensagem;
        tempoMensagem = 0;

        String saida =
        ""+Calendar.getInstance().get(Calendar.DAY_OF_MONTH);
        saida +=
        "/" +Calendar.getInstance().get(Calendar.MONTH);
        saida +=
        "/" +Calendar.getInstance().get(Calendar.YEAR);
        saida += " -
        "+Calendar.getInstance().get(Calendar.HOUR_OF_DAY);
        saida +=
        ":" +Calendar.getInstance().get(Calendar.MINUTE);
        saida +=
        ":" +Calendar.getInstance().get(Calendar.SECOND);
        saida += " - "+mensagem+"\n";

        Gdx.app.Log("teste",saida);

        FileHandle log =
        Gdx.files.local("log/mensagens.txt");
        log.writeString(saida, true);
    }

    private void mostrarMensagens(){
        if (tempoMensagem < TEMPO_MAX_MENSAGEM){
            tempoMensagem++;

            float xx = 0.2f;
            float yy = Conf.CAM_TAM_ALT - 0.3f;

            font.setScale(0.025f);
            font.setColor(1, 1, 1, 1);
            font.draw(batch, mensagem, xx, yy);
        }
    }
}

```

4. Mapa.java

Nesta classe são armazenados todos os dados referentes ao nível, como matriz inferior e superior, e os métodos de exportação e importação de níveis, e de verificação de eficácia.

Código:

```
package br.ufla.americatribal.editor;

import br.ufla.americatribal.editor.aestrela.AEstrela;
import br.ufla.americatribal.editor.aestrela.Lista;
import br.ufla.americatribal.editor.aestrela.Ponto;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.files.FileHandle;

public class Mapa {

    /*    valores
        chao:
            0: grama
            1: areia
            2: terra
            3: agua
            4: pedra
            5: floresta

        objeto:
            0 : vazio
            1 : observatorio
            2 : fazenda
            3 : floresta
            4 : mina
            5 : carne
    */
```

```

        6 : madeira
        7 : pedra
        8 : aldeia time 1
        9 : aldeia time 2
       10: explorador time 1
       11: explorador time 2
*/

public static final int MIN_X = 13;
public static final int MIN_Y = 7;
public static final int MAX_X = 30;
public static final int MAX_Y = 30;

public String nome = "mapa-01";

public int tam_x = MIN_X;
public int tam_y = MIN_Y;

public int x = 0;
public int y = 0;

public int [][] chao;
public int [][] objeto;

/* Mapa com uma camada apenas, para o A*
 *
 * valor das células:
 *
 * 0 : passável, sem nada em cima
 * 1 : bloqueio fixo
 * 2 : aldeia do jogador
 * 3 : aldeia inimiga
 * 4 : explorador do jogador
 * 5 : explorador inimigo
 * 6 : recursos coletáveis (carne)
 * 7 : recursos coletáveis (madeira)
 * 9 : recursos coletáveis (pedra)
 * 10 : consumíveis (carne)
 * 11 : consumíveis (madeira)
 * 12 : consumíveis (pedra)
 * 13 : observatório
 * 14 : unidades inimigas (para ambos os times) *não-
utilizado-ainda
 *
 */

```

```

//public static final int CEL_PASSAVEL = 0;
public int [][] passavel;
public int [][] geral;

public Mapa(){
    mudarTamanho(tam_x, tam_y, true);
}

public Mapa(int tx, int ty){
    mudarTamanho(tx, ty, true);
}

public void mudarTamanho(int tx, int ty, boolean alterar){
    if (tx < MIN_X)
        tx = MIN_X;
    if (ty < MIN_Y)
        ty = MIN_Y;

    if (tx > MAX_X)
        tx = MAX_X;
    if (ty > MAX_Y)
        ty = MAX_Y;

    if (tx < tam_x || ty < tam_y)
        alterar = true;

    int [][] chaoantigo = chao;
    int [][] objetoantigo = objeto;

    chao = new int[tx][ty];
    objeto = new int[tx][ty];

    if (alterar == false){
        for (int x = 0; x < tam_x; x++){
            for (int y = 0; y < tam_y; y++){
                chao[x][y] = chaoantigo[x][y];
                objeto[x][y] =
objetoantigo[x][y];
            }
        }
        for (int x = tam_x; x < tx; x++){
            for (int y = 0; y < ty; y++){
                chao[x][y] = 0;
                objeto[x][y] = 0;
            }
        }
    }
}

```

```

    }
}

tam_x = tx;
tam_y = ty;

if (alterar == true){
    for (int x = 0; x < tam_x; x++){
        for (int y = 0; y < tam_y; y++){
            chao[x][y] = 0;
            objeto[x][y] = 0;
        }
    }
    // time 1, aldeia e explorador
    objeto[1][1] = Conf.ALDEIA_A;
    objeto[2][1] = Conf.EXPLORADOR_A;
    // time 2, aldeia e explorador
    objeto[tam_x-2][tam_y-2] = Conf.ALDEIA_B;
    objeto[tam_x-3][tam_y-2] = Conf.EXPLORADOR_B;
}

}

public void mudarChao(int x, int y, int tile){
    if (x >= 0 && x < tam_x && y >= 0 && y < tam_y)
        chao[x][y] = tile;
}

public void mudarObjeto(int x, int y, int obj){
    if (x >= 0 && x < tam_x && y >= 0 && y < tam_y){
        if (objeto[x][y] != Conf.ALDEIA_A &&
            objeto[x][y] != Conf.ALDEIA_B &&
            objeto[x][y] != Conf.EXPLORADOR_A &&
            objeto[x][y] != Conf.EXPLORADOR_B)
        {
            if (obj == Conf.ALDEIA_A){
                remover(Conf.ALDEIA_A);
            }
            if (obj == Conf.ALDEIA_B){
                remover(Conf.ALDEIA_B);
            }
            if (obj == Conf.EXPLORADOR_A){
                remover(Conf.EXPLORADOR_A);
            }
            if (obj == Conf.EXPLORADOR_B){

```



```

        remover(Conf.EXPLORADOR_B);
    }

    objeto[x][y] = obj;
}
}

private void remover(int obj){
    for (int x = 0; x < tam_x; x++){
        for (int y = 0; y < tam_y; y++){
            if (objeto[x][y] == obj)
                objeto[x][y] = 0;
        }
    }
}

public void salvar(){
    FileHandle arquivo = Gdx.files.Local(Conf.CAMINHO +
nome + Conf.EXTENSAO);
    StringBuilder conteudo = new StringBuilder();

    conteudo.append("Nível America Tribal\n");
    conteudo.append("Dimensões:");
    conteudo.append(tam_x+" "+tam_y+"\n");
    conteudo.append("Blocos Chão:\n");
    for (int y = tam_y - 1; y >= 0; y--){
        for (int x = 0; x < tam_x; x++){
            conteudo.append(" "+chao[x][y]);
        }
        conteudo.append("\n");
    }
    conteudo.append("Blocos Objetos:\n");
    for (int y = tam_y - 1; y >= 0; y--){
        for (int x = 0; x < tam_x; x++){
            if (objeto[x][y] < 10)
                conteudo.append("
"+objeto[x][y]);
            else
                conteudo.append("
"+objeto[x][y]);
        }
        conteudo.append("\n");
    }
    conteudo.append("Fim");
}

```

```

        arquivo.writeString(conteudo.toString(), false);
    }

    public boolean carregar(String nomeNovo){
        FileHandle arquivo = Gdx.files.Local(Config.CAMINHO +
nomeNovo + Config.EXTENSAO);
        if (arquivo.exists()){
            String [] dados =
arquivo.readString().split("\n");

            nome = arquivo.nameWithoutExtension();
            TelaPrincipal.nomeMapa = nome;

            String [] dimensoes = dados[1].split(":");
            dimensoes = dimensoes[1].split(" ");
            int tx = Integer.valueOf(dimensoes[0]);
            int ty = Integer.valueOf(dimensoes[1]);

            mudarTamanho(tx,ty,true);

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

                for (int x = 0; x < tx; x++){
                    chao[x][ty-y-1] =
pegarInteiro(linhachao[x]);
                    objeto[x][ty-y-1] =
pegarInteiro(linhaobj[x]);
                }
            }
            return true;
        }else{
            return false;
        }
    }

    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 void criarMapaGeral(){
        for (int x = 0; x < tam_x; x++){
            for (int y = 0; y < tam_y; y++){

            }
        }
    }
    */

    public void criarMapaPassavel(){
        passavel = new int[tam_x][tam_y];

        for (int x = 0; x < tam_x; x++){
            for (int y = 0; y < tam_y; y++){
                passavel[x][y] = 0;

                if (objeto[x][y] == 8){
                    passavel[x][y] = 2;
                }else if (objeto[x][y] == 9){
                    passavel[x][y] = 3;
                }else if (objeto[x][y] > 0 &&
objeto[x][y] < 5){
                    passavel[x][y] = 1;
                }else if (objeto[x][y] == 0 &&
chao[x][y] > 2){
                    passavel[x][y] = 1;
                }
            }
        }
    }

    private boolean verificarCaminhoEntreTribos(){
        int origemx = 0, origemy = 0;
        int destinox = 1, destinoy = 1;
    }

```

```

        criarMapaPassavel();

        for (int x = 0; x < tam_x; x++){
            for (int y = 0; y < tam_y; y++){
                if (passavel[x][y] == 2){
                    origemx = y;
                    origemy = x;
                }
                if (passavel[x][y] == 3){
                    destinox = y;
                    destinoy = x;
                }
            }
        }

        Ponto origem = new Ponto(origemx, origemy);
        Ponto destino = new Ponto(destinox, destinoy);

        //origem.desenhar();
        //destino.desenhar();

        AEstrela aestrela = new
        AEstrela(passavel,origem,destino);

        //aestrela.desenhar();

        Lista caminho = Lista.aEstrela(aestrela);

        if (caminho == null){
            //Gdx.app.log("aEstrela", "Não tem
            caminho!");
            return false;
        }

        //caminho.desenhar();

        if (caminho.tem(origem) && caminho.tem(destino) ){
            //Gdx.app.log("aEstrela", "TEM caminho!");
            return true;
        }

        //Gdx.app.log("aEstrela", "Se pá deu erro!");
        return false;
    }

```

```

private boolean verificarJogadores(){
    int exploradorA = 0;
    int exploradorB = 0;
    int aldeiaA = 0;
    int aldeiaB = 0;

    /*
    2 : fazenda
    3 : floresta
    4 : mina
    5 : carne
    6 : madeira
    7 : pedra
    8 : aldeia time 1
    9 : aldeia time 2
    10: explorador time 1
    11: explorador time 2
    */

    for (int x = 0; x < tam_x; x++){
        for (int y = 0; y < tam_y; y++){
            switch(objeto[x][y]){
                case 8:      aldeiaA++; break;
                case 9:      aldeiaB++; break;
                case 10:exploradorA++; break;
                case 11:exploradorB++; break;
            }
        }
    }

    if (aldeiaA != 1 ||
        aldeiaB != 1 ||
        exploradorA != 1 ||
        exploradorB != 1)
    {
        return false;
    }

    return true;
}

private boolean verificarRecursos(){
    int col_comida      = 0;
    int col_madeira = 0;

```

```

int col_pedra      = 0;

int fix_comida     = 0;
int fix_madeira    = 0;
int fix_pedra      = 0;

/*
2 : fazenda
3 : floresta
4 : mina
5 : carne
6 : madeira
7 : pedra
8 : aldeia time 1
9 : aldeia time 2
10: explorador time 1
11: explorador time 2
*/

for (int x = 0; x < tam_x; x++){
    for (int y = 0; y < tam_y; y++){
        switch(objeto[x][y]){
            case 2:      fix_comida++; break;
            case 3:      fix_madeira++; break;
            case 4:      fix_pedra++; break;
            case 5:      col_comida++; break;
            case 6:      col_madeira++; break;
            case 7:      col_pedra++; break;

        }
    }
}

if (fix_comida % 2 != 0 ||
    fix_madeira % 2 != 0 ||
    fix_pedra % 2 != 0 ||
    col_comida % 2 != 0 ||
    col_madeira % 2 != 0 ||
    col_pedra % 2 != 0)
{
    return false;
}

return true;

```

```

    }

    public boolean verificarEficacia(){
        if (verificarJogadores() == false)
            return false;

        if (verificarRecursos() == false)
            return false;

        if (verificarCaminhoEntreTribos() == false)
            return false;

        return true;
    }
}

```

5. Conf.java

Esta classe simplesmente armazena os dados globais utilizados pelo editor.

Código:

```

package br.ufla.americatribal.editor;

import com.badlogic.gdx.Gdx;

public class Conf {
    public static final int TELA_RES_LAR = 1280;
    public static final int TELA_RES_ALT = 720;

    public static final float CAM_TAM_LAR = 24.0f;
    public static final float CAM_TAM_ALT = 13.5f;

    public static float unidades_x = CAM_TAM_LAR /
TELA_RES_LAR;
    public static float unidades_y = CAM_TAM_ALT /
TELA_RES_ALT;

    public static final float TAM_BLOCO = 1f;

```

```

        public static float tam_bloco_atual = TAM_BLOCO;

        public static final String PACOTE_IMAGENS =
            "data/imagens/america-tribal-editor-
imagens.pack";

        public static final int MAX_ITENS_CHAO = 6;
        public static final int MAX_ITENS_OBJ = 12;

        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;

        public static void alterarUnidades(){
            unidades_x = CAM_TAM_LAR / Gdx.graphics.getWidth();
            unidades_y = CAM_TAM_ALT / Gdx.graphics.getHeight();
        }

        public static final String CAMINHO = "niveis/";
        public static final String EXTENSAO = ".atm";
    }

```


9 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 específico 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 primária 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 Secundária

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:** os 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 início de cada turno um número aleatório entre 0 e o número 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 número 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 crítico ou um dano comum. Ela é definida por uma porcentagem. Toda vez que é confirmado um ataque é gerado um número aleatório entre 0 e 100 para verificar o crítico. 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: <i>Skin Walker</i> (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