UNIVERSIDADE FEDERAL DE SANTA CATARINA

JTiled2D: uma biblioteca para desenvolvimento de jogos 2D em perspectiva *top-down* na linguagem Java.

Diego Feijó

Florianópolis – SC

2020/1

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

JTiled2D: uma biblioteca para desenvolvimento de jogos 2D em perspectiva *top-down* na linguagem Java.

Diego Feijó

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de bacharel em Sistemas de Informação

Florianópolis – SC
2019/1

Diego Feijó

JTiled2D: uma biblioteca para desenvolvimento de jogos 2D em perspectiva *top-down* na linguagem Java.

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de bacharel em Sistemas de Informação

Orientador

_____

José Eduardo De Lucca

Banca Examinadora:

_____

Ricardo Pereira e Silva

_____

Jean Carlo Rossa Hauck

Sumário

Lista de Figuras

Lista de Reduções

2D	Duas dimensões
3D	Três dimensões
API	Application Programming Interface
HUD	Heads Up Display
IDE	Integrated Development Environment
NPC	Non Playable Character ou Personagem Não Jogável
PDJ	Personagem do Jogador
RPG	Role-playing Game
UML	Unified Modeling Language
XML	Extensible Markup Language

Resumo

O desenvolvimento de jogos em Java exige, além do conhecimento avançado na linguagem, o uso de ferramentas que garantam desempenho em tempo de execução, estas que por sua vez tem sua própria complexidade de implementação. Para iniciantes no desenvolvimento de de jogos digitais esses são dois grandes obstáculos. Este presente trabalho tem por objetivo desenvolver uma biblioteca que facilite o uso das principais APIs da linguagem Java utilizadas no desenvolvimento de games em 2D, abstraindo sua implementação para o desenvolvedor iniciante. Para fins didáticos, a biblioteca dará suporte a jogos baseados em tiles – uma divisão da tela em uma grade composta por células quadradas - com perspectiva top-down – onde o jogador visualiza o cenário de cima do ator. No decorrer deste documento são discutidos os principais elementos de um game no formato descrito, a criação da biblioteca baseada nesses elementos e uma documentação de uso desta. O objetivo da biblioteca desenvolvida é de ser uma porta de entrada para desenvolvedores iniciantes em Java e na área de jogos digitais. Além disto, espera-se do desenvolvedor que for utilizar esta ferramenta o conhecimento básico em Programação Orientada a Objetos. Com o uso da ferramenta jTiled2D foi desenvolvido um jogo de exemplo que utiliza as suas principais funcionalidades, exibindo a capacidade da biblioteca e servindo também de apoio aos seus futuros usuários.

Palavras-chave: jogos digitais. jogos 2D em java. desenvolvimento de jogos em java.

1 Introdução

Jogos virtuais - ou simplesmente games - são uma grande fonte de entretenimento e aprendizado com um número de usuários crescente a cada ano. Uma pesquisa pelo grupo The Npd Group Inc (2015) informa que aproximadamente 82% dos brasileiros são *gamers*. Em uma outra pesquisa, realizada pela Newzoo (2018), o número cresce para 83% de brasileiros que compraram jogos ou gastaram dinheiro em itens ou produtos virtuais em jogos e posiciona o Brasil como o 13º mercado de games mundial.

1.1 Motivação

Para a aprovação na disciplina de Análise e Projeto de Sistemas, do curso de Sistemas de Informação do CTC/UFSC, ministrada pelo professor Ricardo Pereira e Silva, os alunos têm como desafio o desenvolvimento de um game multijogador que utilize o *framework NetGamesNRT* para comunicação em rede, este por sua vez na linguagem Java[1]. A disciplina tem por objetivo apresentar o processo de Engenharia de Software de forma genérica, sem foco para as diretivas de um processo de desenvolvimento de um game. Logo, o aluno que não conheça esse universo, tende a encontrar uma grande barreira ao iniciar o projeto do game em Java.

É evidente que a existência de ferramentas - bibliotecas, *frameworks*, *game engines* - para auxiliar no desenvolvimento de games em Java é vasta

[1] JAVA Oracle. Disponível em: https://www.java.com. Acesso em: 28 mar. 2020.

e alguns exemplos serão vistos no decorrer deste trabalho. Em adição a estas ferramentas, o presente trabalho propõe o desenvolvimento da biblioteca jTiled2D, com o objetivo de simplificar o processo de aprendizado para o aspirante em desenvolvimento de games e aproximar os estudantes de Sistemas de Informação desta área.

1.2 Objetivos

1.2.1 Objetivo geral

Desenvolver uma biblioteca que abstraia a complexidade de desenvolvimento de games em Java para programadores iniciantes na linguagem e facilite a construção de um game.

Considerando a definição de jogo adotada, ao fim deste projeto deve-se portanto ser possível emular um sistema que sugira um conflito artificial, permeie as regras deste e permita a interação do jogador, com resultados quantificáveis.

Como escopo do projeto, será utilizada a estrutura de um game em duas dimensões e de visão *top-down* onde o jogador visualiza o cenário por cima, como visto em grande parte dos jogos para dispositivos móveis - ou os famosos RPGs dos anos 90.

1.2.2 Objetivos específicos

1. Desenvolver a revisão bibliográfica do estado da arte para desenvolvimento de jogos virtuais.

2. Apresentar as principais características de jogos em 2D com visão *top-down*.
3. Implementar a ferramenta que abstraia o desenvolvimento das características apresentadas acima para programadores novatos no universo de desenvolvimento de games.

1.3 Metodologia

O trabalho será desenvolvido com base em uma pesquisa exploratória por meio do desenvolvimento de uma biblioteca que abstraia o desenvolvimento das características básicas de games em 2D com visão *top-down* em Java. A metodologia de desenvolvimento deste trabalho será dividida nas seguintes etapas:

1. Análise da literatura, focando em desenvolvimento de jogos 2D.
2. Levantamento de requisitos da biblioteca objetivo.
3. Construção do projeto de software em notação UML, utilizando a ferramenta Visual Paradigm[2].
4. Desenvolvimento da biblioteca em linguagem Java, utilizando a IDE IntelliJ[3]. Durante esta etapa, será realizado em paralelo:
   a) Revisão dos requisitos levantados e desenvolvidos.
   b) Criação da documentação de uso da biblioteca.
   c) Teste da biblioteca através do desenvolvimento de um game.

---

[2] VISUAL Paradigm. Disponível em: https://www.visual-paradigm.com/. Acesso em: 28 mar. 2020.

[3] INTELLIJ IDEA. Disponível em: https://www.jetbrains.com/idea/. Acesso em: 28 mar. 2020.

2 Conceitos básicos

Neste capítulo serão tratados os conceitos básicos que servirão de insumos para a elicitação de requisitos da biblioteca jTiled2D. Serão definidos os conceitos de *jogo, perspectiva de visão de jogo, tiles* e *tilesets.*

2.1 Definição de jogo

Quando se pergunta o que é um jogo, a resposta geralmente vem em forma de exemplos: “isto é um jogo, isto não”. Mas quando se pensa em uma definição formal, o real desafio surge. O que diferencia um jogo de qualquer outra atividade?

Uma brincadeira de “pegar” entre cachorros pode ser comparada com a mesma entre humanos, mesmo quando o primeiro nunca teve interferência dos homens para ensiná-lo. Mas o que diferencia o jogar dos animais do dos homens? Em Homo Ludens, (HUIZINGA, 2017) o autor comenta que “em toda parte encontramos presente o jogo, como uma qualidade de ação bem determinada e distinta da vida comum” e continua com “o conceito de jogo deve permanecer distinto de todas as outras formas de pensamento através das quais exprimimos a estrutura da vida espiritual e social”. O jogo, portanto, se distingue do comum, tem seu próprio escopo de pensamento. Mas como definir isto? Em seu ensaio sobre a interação lúdica na vida social humana, Huizinga propõe a seguinte definição de jogo:

> O Jogo é uma atividade livre, ficando conscientemente tomada como “não séria” e exterior à vida habitual, mas ao mesmo tempo, capaz de absorver o jogador de maneira intensa e total. É uma atividade desligada de todo e qualquer interesse material, com a qual não se pode obter qualquer lucro. Ela é praticada dentro de seus próprios limites de tempo e espaço de acordo com regras fixas e de uma maneira ordenada. Promove a formação de agrupamentos sociais, que tendem a se cercar de sigilo e sublinha a sua diferença em relação ao mundo comum, por disfarce ou outros meios (HUIZINGA, 2017, p.13).

Com isso pode-se dizer que o jogo é uma atividade:

- Livre: que deve ser realizada por vontade dos participantes, sem obrigação;
- Fora da vida ordinária: absorve os participantes para sua própria atmosfera;
- Improdutiva: nada se espera criar;
- Finita: em seu próprio espaço e tempo. O ato de jogar tem início e fim; e

- Social: onde participantes do mesmo jogo se reúnem com os mesmos interesses e objetivos.

Estão definidas as características do objeto de estudo, mas ainda assim não é possível distingui-lo de outras atividades parecidas. O exemplo de cachorros brincando de pegar pode se encaixar perfeitamente na descrição de Huizinga. Sabemos que a definição de jogo que se busca vai além disso.

Em Os jogos e os homens (CAILLOIS, 2017) segue o estudo de Huizinga com uma definição inspirada porém, de certa forma, expandida. Em adição às características de Huizinga, Caillois apresenta as seguintes:

- Incerta - não se pode determinar o curso da atividade de jogo nem seu resultado;
- Regida por regras - jogo tem seu próprio universo de limites e circunstâncias pré determinadas; e
- Faz de conta - separado da vida real, altera a realidade de sua atmosfera.

Ainda com essas adições, não é possível determinar um conceito para o objeto deste estudo. A brincadeira de pegar dos cachorros pode, novamente, ser descrita pelo conjunto de características obtido até então.

Tacitamente é regrado que não se pode morder “para valer”, faz-se de conta que é uma briga e o curso da atividade é claramente incerto.

Já em Regras do jogo (SALEN; ZIMMERMAN, 2012) são comparadas várias definições de jogos - incluindo as já citadas - e seu estudo vai ao encontro do que se busca neste. Através do compilado de oito diferentes definições, chegam ao seguinte resultado: “Um jogo é um sistema no qual os jogadores se envolvem em um conflito artificial, definido por regras, que implica um resultado quantificável.” (SALEN; ZIMMERMAN, 2012) O importante desta definição é que adiciona o conceito de sistema que, de acordo com os autores, pode ser definido como “um conjunto de peças que se inter-relacionam para formar um todo complexo”. Além disso os jogadores são incluídos assim como a ideia de conflito - uma cooperação ou competição entre os jogadores.

Neste estudo, será considerada a definição de Salen e Zimmerman a fim de determinar o que pode ser considerado jogo.

2.2 Perspectivas de visão de jogo

Taylor (2002) define perspectivas de visão de jogo da seguinte maneira (retirado do texto em inglês):

> “A perspectiva ótica de como o jogo é jogado, tal como é representado no envolvimento com a interface, é também fundamental para a criação do espaço de jogo. A perspectiva para o jogo de videogame define o ponto de visão pelo qual o

> jogador interage com o espaço geral do jogo e o seu ambiente interno. As perspectivas de ponto de vista mais utilizadas para a jogabilidade são:
>
> - *first person*;
> - *third person*;
> - *overhead, top-down* ou *god view*; e
> - *three-fourths isometric*.” (TAYLOR, 2002, p.5)

2.2.1 First person

O espaço de jogo é apresentado como a visão do protagonista, este é sempre a primeira pessoa da cena apesar de não ser exibido por completo. É o caso de jogos de tiro como *Doom* da produtora *id Software* ou jogos de simulação de corrida (vide Figura 1).

Figura 1 - Doom e sua perspectiva *first person*

Fonte: Doom Fandom[4]

2.2.2 Third Person

Ao invés de apresentar a visão do protagonista, esta perspectiva se preocupa em apresentar o ambiente de jogo em volta dele. Aqui o jogador acompanha o personagem, seja em uma visão lateral como em *Super Mario Bros*[5]. ou, em jogos 3D por trás do personagem.

[4] Disponível em: https://doom.fandom.com/wiki/Doom. Acesso em: 28 mar. 2020.

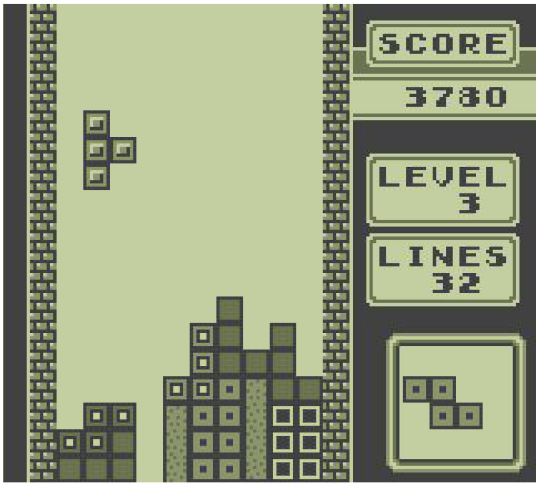
[5] SUPER Mario Bros. Disponível em: https://mario.nintendo.com/. Acesso em: 28 mar. 2020.

2.2.3 Overhead, Top-down ou God View

Conforme Taylor (2002) “a jogabilidade é baseada no conceito de que o jogador é uma força que age sobre o mundo do jogo ao invés de uma entidade dentro do jogo que então interage com objetos e atores.” É uma perspectiva “de cima” do mundo de jogo, que representa uma visão geral, um ponto de visão distante, abstrato, o fim lógico do paradigma “cone de visão” da perspectiva linear onde o fim do cone está fora do espaço de jogo.

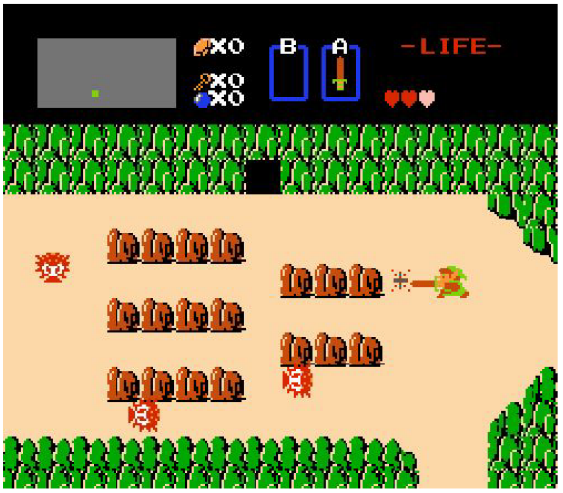
Esta perspectiva pode ser observada em jogos puzzle como *Tetris* (vide Figura 2), simuladores de jogos de tabuleiro – xadrez, damas, cartas – , jogos de RPG e ação como *Final Fantasy*[6] e *The Legend of Zelda*[7] (vide Figura 3) entre outros.

Figura 2 - Tetris



Fonte: Tetris[8]

Figura 3 - The Legend of Zelda



Fonte: The Legend of Zelda[7]

[6] FINAL Fantasy. Disponível em: https://na.finalfantasy.com/. Acesso em: 28 mar. 2020.

[7] THE Legend of Zelda. Disponível em: https://www.zelda.com/. Acesso em: 28 mar. 2020.

[8] TETRIS. Disponível em: https://tetris.com/. Acesso em: 28 mar. 2020.

2.2.4 Three-fourths Isometric

Esta visão coloca o personagem do jogador como parte da estrutura global e, por ter uma visão de mundo mais simplificada, permite que mais informações estejam presentes na tela. Jogos como *Diablo I* e *II*[9] contam com esta perspectiva. A Figura 4 exibe um exemplo de visão isométrica.

Figura 4 - Diablo I com visão isométrica

Fonte: Blizzard[9]

[9] BLIZZARD. Disponível em: https://www.blizzard.com/en-us/games/legacy/. Acesso em: 28 mar. 2020.

2.3 Tileset e tiles

Um jogo baseado em tiles é um tipo de jogo onde a área jogável consiste de uma grade onde suas células são imagens em pequenos quadrados chamados de tiles. Estas imagens unidas montam o cenário de jogo e toda sua mecânica – como colisão e sobreposição – são baseadas nestes tiles.

O conjunto de tiles disponíveis para a montagem do mapa é conhecido como tileset e funciona como uma paleta para o designer do mapa, exemplificado na Figura 5.

Figura 5 - Exemplo de tilesets

Fonte: OpenGameArt.Org[10]

[10] OPENGAMEART.ORG. Disponível em: https://opengameart.org/. Acesso em: 28 mar. 2020

3 Ferramentas existentes

Nesta seção são discutidas as ferramentas existentes para auxílio ao desenvolvimento de jogos na linguagem Java.

Foi utilizada a ferramenta de pesquisa Google com as palavras chaves java; 2d; tile; game; top down; library; game engine; game dev; e encontrados aproximadamente 990 mil resultados. Foram lidas as 10 primeiras páginas com resultados de pesquisa, sendo os mais relevantes apresentados aqui.

3.1 LITIengine

O game engine open source LITIengine[11] provê a infraestrutura para desenvolvimento de jogos 2D baseados em tiles na linguagem java. Originalmente escrito pelos irmãos Steffen e Matthias Wilke, hoje conta com o suporte de uma grande comunidade de desenvolvedores.

Suas principais funcionalidades incluem um mecanismo de física em duas dimensões, um mecanismo de renderização em duas dimensões, reprodução de sons em duas dimensões, um sistema de partículas, suporte ao formato .tmx de mapas baseados em tiles e uma API para estrutura básica de jogo.

Além disso, é totalmente baseada na biblioteca Java AWT Graphics.

A documentação é disponível online e uma comunidade de game devs engaja em discussões no blog do projeto.

[11] LITIENGINE. Disponível em: https://litiengine.com. Acesso em: 04 out. 2019.

3.2 jMonkeyEngine

Esta engine[12] tem uma proposta para jogos 3D e crossplatform – desenvolvimento para várias plataformas como Windows, Mac OS, Linux, Android e iOS. É uma ferramenta gratuita, de código aberto, desenvolvida para quem deseja criar jogos em java usando a mais moderna tecnologia. Possui ampla documentação e, assim como a LITIengine, sugere que o desenvolvedor tenha uma base de conhecimento em programação para que possa desfrutar de seus benefícios.

É uma proposta diferente do que este trabalho apresenta, visto o desenvolvimento voltado para três dimensões, porém vale ser mencionado visto o grande número de usuários em sua comunidade de desenvolvimento. Em outubro de 2019 contava com mais de 100 contribuintes no repositório do GitHub.

3.3 libGDX

Ao contrário dos anteriores, libGDX[13] não é uma engine mas sim um framework para desenvolvimento de jogos em Java. Fornece uma API para as plataformas suportadas e permite testar o jogo em ambiente desktop antes de publicar para plataformas móveis como Android e iOS.

---

[12] JMONKEYENGINE. Disponível em: https://jmonkeyengine.org/. Acesso em: 07 out. 2019.

[13] LIBGDX. Disponível em: https://libgdx.badlogicgames.com/. Acesso em: 04 out. 2019.

Como especificado em sua documentação, libGDX não pretende ser uma ferramenta única de trabalho mas sim fornecer um grande auxílio para controles de gráficos, inserção de comandos, controle de áudio, ferramentas de auxílio para cálculos matemáticos e físicos, gestão de escrita e leitura de arquivos, entre outras funcionalidades relacionadas.

Conta com uma grande comunidade de usuários que, além dos fóruns comuns às demais ferramentas apresentadas, se reúnem em um canal na ferramenta de chat online Discord. No momento em que este texto foi escrito haviam 270 usuários online – de um total de 2388 membros – discutindo sobre desenvolvimento de jogos com o libGDX.

3.4 Ferramentas de apoio

Em conjunto ao desenvolvimento das regras do jogo, deve-se considerar também a ambientação do jogador. Para os jogos 2D baseados em tiles, grande parte desta responsabilidade cai sobre os cenários apresentados em formatos de mapas de tiles. Além disso, os elementos de interação – como o avatar do jogador, personagens não jogáveis, itens coletáveis, inimigos e etc. - devem ser apresentados neste mapa.

Nesta seção é apresentada uma ferramenta de apoio ao desenvolvimento de jogos 2D baseados em tiles que auxilia o desenvolvedor a criar e editar os recursos de ambiente, mais precisamente para desenvolvimento de mapas.

Utilizou-se a ferramenta de pesquisa Google com as palavras chave tiled map 2d editor e a ferramenta Tiled Map Editor foi o resultados mais relevante entre os 4.170.000 encontrados.

3.4.1 Tiled Map Editor

O Tiled Map Editor[14] é um editor de nível 2D que ajuda a desenvolver o conteúdo de jogo. Sua principal característica é editar mapas baseados em tiles de várias formas, mas também suporta o posicionamento livre de imagens, bem como maneiras poderosas de anotar seu mapa com informações extras usadas pelo jogo.

Suporta o uso tanto de apenas um como vários tileset de forma dinâmica, sem que seja necessário um único arquivo com todos os tiles. Tilesets são conjuntos de tiles utilizado como peças de um quebra cabeças dinâmico para montagem de mapas. Permite desenhar o mapa em diferentes camadas, tornando simples a noção de profundidade e sobreposição de imagens. Conta também com modelos de terrenos, onde é possível indicar os tiles que indicam as bordas de um terreno que se deseja desenhar de forma livre – como um rio, por exemplo – e quando selecionado o tile para preenchimento, as bordas são adicionadas automaticamente.

Os mapas desenvolvidos podem ser exportados em formato TMX, que nada mais é do que um XML com tags próprios da ferramenta, o que facilita a leitura e renderização por bibliotecas e engines.

---

[14] TILED Map Editor. Disponível em: https://www.mapeditor.org/. Acesso em: 15 out. 2019.

É uma ferramenta utilizada por vários desenvolvedores independentes e mantida por uma grande comunidade de usuários. Seu repositório Git conta com 251 contribuintes no momento em que este texto foi redigido. Possui documentação online, fórum de dúvidas e tutoriais e ainda um canal na ferramenta Discord.

3.5 Análise das ferramentas apresentadas

De acordo com as documentações das ferramentas apresentadas, o requisito mínimo de conhecimento da linguagem Java é um pouco além do básico. É exigido do desenvolvedor o uso de estruturas um pouco mais complexas do que *if* e *for* para o desenvolvimento de um jogo simples.

A ferramenta desenvolvida neste trabalho não tem a intenção de competir ou atender todas as funcionalidades acobertadas pelas ferramentas apresentadas. Ao invés disso, almeja apresentar um fluxo simples de desenvolvimento para jogos simples. Se o desenvolvedor possui conhecimento moderado ou avançado na linguagem ou objetiva desenvolver um jogo complexo, é altamente recomendado que considere o uso das ferramentas listadas neste capítulo. Agora se pretende saborear o desenvolvimento de um jogo simples e está iniciando sua aventura com Java, JTiled2D será seu guia.

4 Desenvolvimento da biblioteca jTiled2D

Neste capítulo é discutido o desenvolvimento da biblioteca objetivo deste estudo. Para melhor organização, o capítulo está subdividido entre os requisitos funcionais e não funcionais elicitados, os principais módulos desenvolvidos e as técnicas utilizadas.

Para elicitação dos requisitos foram consideradas as definições de jogo apresentadas neste trabalho.

4.1 Requisitos

4.1.1 Requisitos Não Funcionais

1) **Especificação de projeto**: deve ser desenvolvido em linguagem Java na versão 1.8.1, tendo como resultado o código fonte, especificação das classes em linguagem UML 2.5 e documentação de uso.
2) **Divisão de módulos:** a biblioteca deve ser organizada em módulos, conforme especificado:
   - **a)** Entidades: onde serão gerenciados os atores do jogo, representando inimigos, personagens não jogáveis e o avatar do jogador;
   - **b)** Estados: gerencia o estado atual do jogo;
   - **c)** Gráficos (gfx): controle de renderização de imagens e câmera do jogo;

d) Input: interface entre jogador e jogo, responsável por gerenciar os comandos de teclado e mouse;

e) Janelas: alguns componentes prontos para renderização de janelas de diálogo, menus e monitores de jogo (HUD);

f) Mapas: controle de renderização de mapas e importador de arquivos de extensão TMX da ferramenta Tiled; e

g) Tiles: gestão dos tiles do jogo.

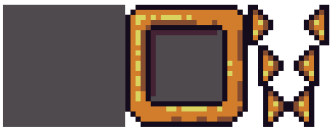
3) **Imagens de entidades:** as imagens de entidades devem ser imagens em grades com 4 (quatro) colunas e 2 (duas) linhas. Cada célula da grade deve ser subdividida em uma nova grade com 3 (três) colunas e 4 (quatro) linhas. A Figura 6 apresenta um exemplo de imagens de entidades.

Figura 6 - Exemplo de imagens de entidade

Fonte: OpenGameArt.Org

4) **Imagens de janelas:** as imagens de janelas devem ser divididas em grades com 3 (três) colunas sendo a primeira coluna a representação da imagem de fundo da janela, a segunda coluna a moldura e a terceira coluna subdividida em uma grade de 2 (duas) colunas e 3 (três) linhas para representar as animações do cursor. A figura 7 demonstra um arquivo de imagem neste formato.

Figura 7 - Exemplo de imagens de janela



Fonte: OpenGameArt.Org, editado pelo autor

4.1.2 Requisitos Funcionais

Para melhor leitura, os requisitos funcionais estão divididos por módulo.

Entidades	
ENT1	Deve ser possível controlar o movimento das entidades do jogo via comando de teclado.
ENT2	As entidades devem respeitar o cenário de jogo, deve-se portanto conferir colisões entre entidades e *tiles*.
ENT3	Deve ser possível renderizar gráficos animados de entidades.
Estados	
EST1	Deve ser possível apresentar os diversos estados do jogo como menus e mapas.
EST2	Deve ser possível guardar o estado atual do jogo para renderização.
Gráficos	
GRA1	O desenvolvedor deve ter controle sobre a câmera de jogo, permitindo centralizá-la em uma entidade.

GRA2	Deve ser possível controlar a movimentação da câmera de jogo livremente.
GRA3	Deve ser possível predeterminar as imagens das entidades em quatro dimensões e animadas.
GRA4	A velocidade da animação das imagens das entidades deve ser configurável.
GRA5	A biblioteca deve contar com o suporte básico a apresentação de um HUD (*heads up display*, ou indicadores de jogo em uma tradução livre).
Input	
INP1	A biblioteca deve dar suporte a inserção de comandos através do teclado, com as seguintes teclas mapeadas: setas direcionais, Enter (confirmação) e Esc (cancelamento).
INP2	A biblioteca deve dar suporte a inserção de comandos através do mouse, sendo possível movimentar um cursor e capturar os botões direito e esquerdo.
Janelas	
JAN1	A biblioteca deve dar suporte a criação de janelas de diálogo, permitindo indicar sua posição em tela, com título, texto e tema gráfico.
JAN2	Deve dar suporte a inserção de textos através de janelas de diálogos.
JAN3	Deve dar suporte a janelas de seleção, onde várias escolhas serão exibidas ao jogador e lhe será permitido selecionar uma destas.
Mapas	
MAP1	Deve dar suporte a renderização de mapas compostos por *tiles*, com camadas sobrepostas e entidades.
MAP2	Deve dar suporte nativo a importação de mapas em formato TMX, da ferramenta Tiled.
Tiles	
TIL1	Deve permitir criar tiles individuais com propriedades específicas, como se uma entidade deve atravessar ou não por ele (colisão).
TIL2	Deve permitir carregar tilesets: imagens com coleções de tiles em grade.
TIL3	Ao carregar um mapa, deve ler as propriedades de cada tile indicado no mapa e armazenar no tile correspondente.

4.2 Desenvolvimento da Ferramenta

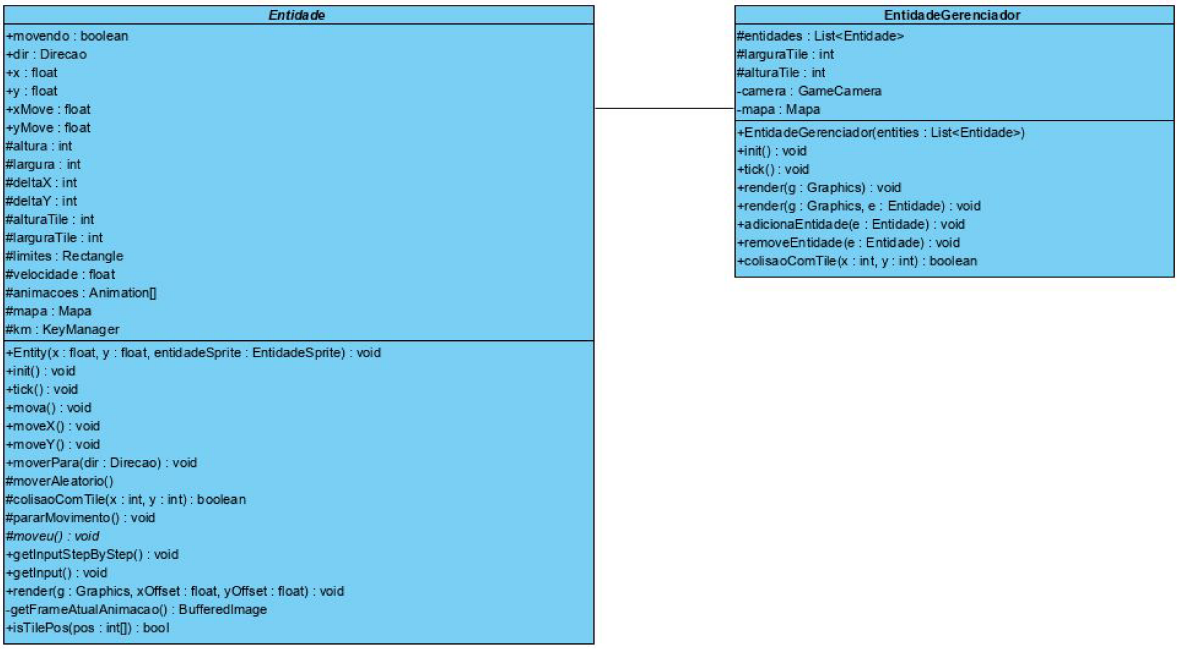
Com base nos requisitos levantados, é apresentado nesta seção, por módulo, o desenvolvimento da ferramenta e o respectivo diagrama de classes.

4.2.1 Desenvolvimento do módulo Entidades

As entidades representam as interações do jogador com o mundo de jogo. Seu próprio avatar no mundo de jogo é uma entidade, assim como inimigos, itens, personagens não jogáveis, portas, baús, etc.

A Figura 8 representa o diagrama de classes do módulo Entidades.

Figura 8 - Diagrama de classes do módulo Entidades



Para o requisito ENT1 foram consideradas duas possibilidades de movimento: *moverPara* onde a entidade será movida enquanto a tecla direcional for pressionada e *mova* onde a direção do movimento será fornecida previamente e a entidade será movida apenas um *tile* para a direção. Ambos consideram colisão com o cenário para completar o movimento, respeitando o requisito ENT2.

A classe *EntidadeGerenciador*, como o próprio nome sugere, gerencia a coleção de entidades em um mesmo *Estado* de jogo. Será responsável por renderizar as animações - conforme requisito ENT2 -, adicionar e remover entidades da coleção e conferir colisões.

4.2.2 Desenvolvimento do módulo Estados

O estado inicial do jogo é determinado pela classe *Launcher* que, ao ser criada, recebe o estado entre outros parâmetros. Este estado deve ser uma implementação da classe abstrata *Estado*, dando a liberdade ao desenvolvedor de criar seu próprio estado de jogo. É no estado que poderão ser carregados o mapa, as entidades e as janelas.

A classe *Launcher* é responsável por inicializar o jogo através da ferramenta jTiled2D. O método estático *launchGame* prepara a classe *Config* com os parâmetros passados para que estejam acessíveis a qualquer momento do jogo. Para isso, foi utilizado o padrão de projeto *Singleton*[15].

A classe *Game* armazena o estado atual do jogo que pode ser alterado a qualquer momento (como ao acessar um menu, ou alterar o cenário), atendendo ao requisito EST1. Além disso, é responsável por disparar os métodos de atualização de regras *tick* e de renderização de gráficos *render*. Estes dois métodos são disparados a cada ciclo de jogo, gerenciado pelo método *run*. Um ciclo de jogo é

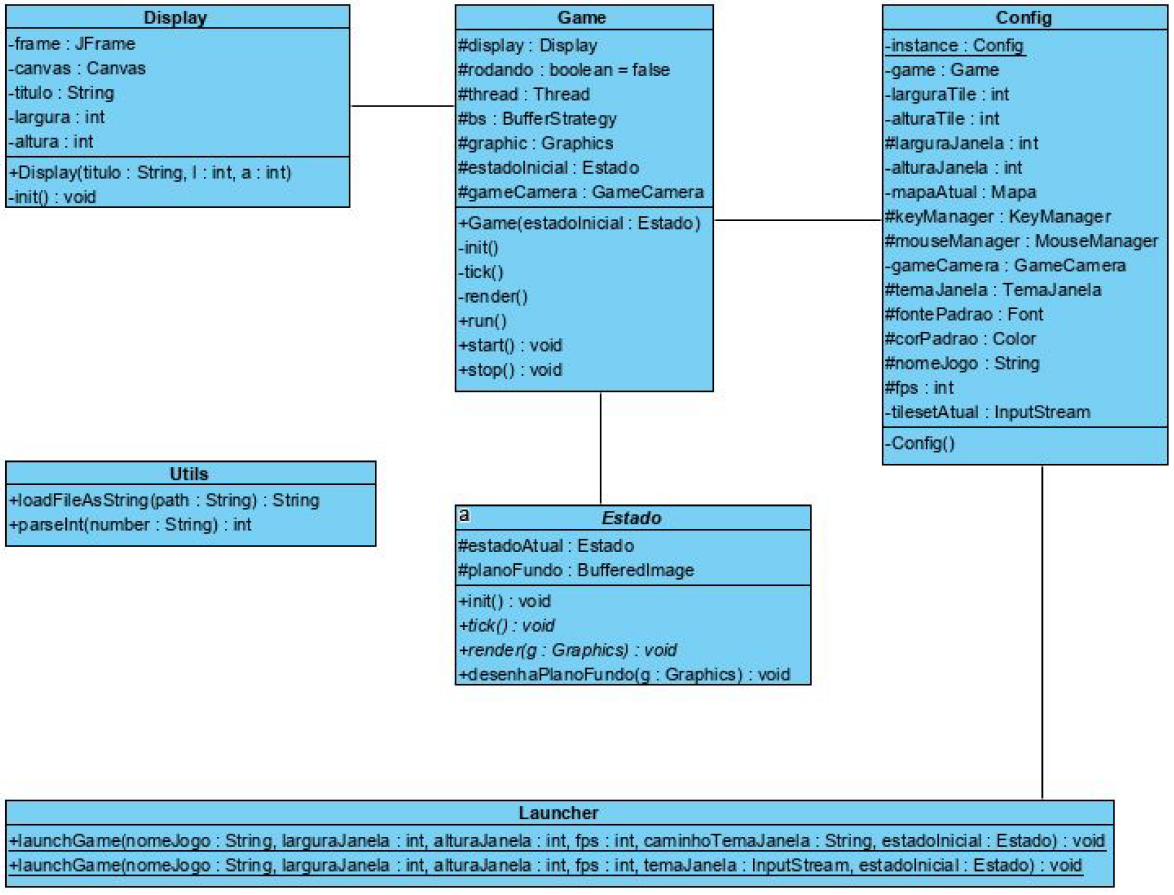
[15] SINGLETON Design Pattern. Disponível em: https://sourcemaking.com/design_patterns/singleton. Acesso em: 28 nov. 2020.

determinado pelo atributo *fps* em *Config*. Um ciclo de 60 fps significa 60 ciclos por segundo. Isto atende o requisito EST2.

A classe *Game* também controla a câmera do jogo, que por padrão centraliza a entidade do jogador no mapa. O método *render* tem a responsabilidade de atualizar a posição da câmera. O controle de câmera será visto no próximo tópico.

A Figura 9 representa o diagrama de classes do módulo Estados.

Figura 9 - Diagrama de classes do módulo Estados

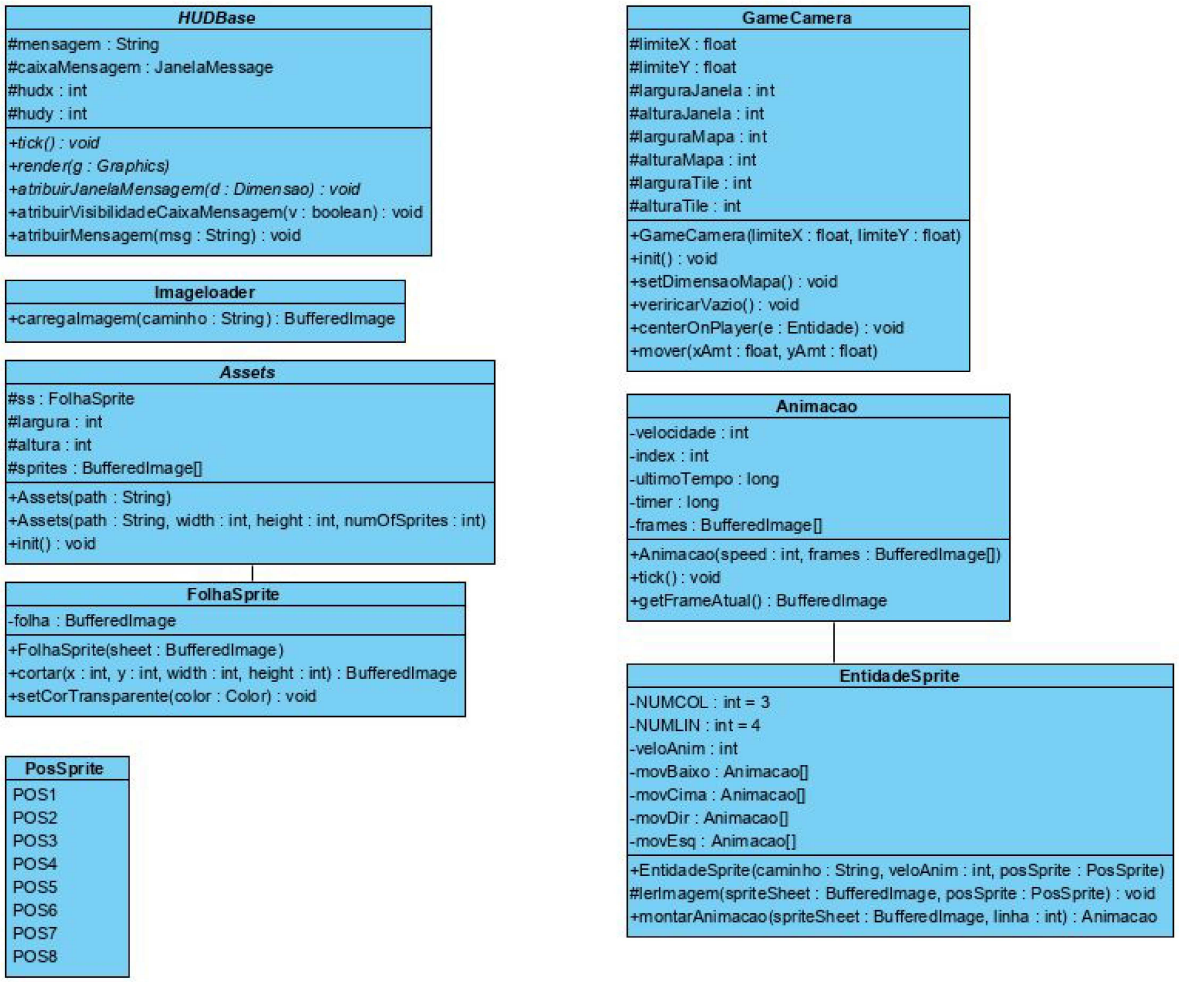


4.2.3 Desenvolvimento do módulo Gráficos

O módulo Gráficos e responsável pela gestão de imagens e animações assim como o controle da câmera de jogo.

A Figura 10 representa o diagrama de classes deste módulo.

Figura 10 - Diagrama de classes do módulo Gráficos



A classe *Assets* facilita o acesso aos recursos visuais. Deve ser implementada pelo desenvolvedor para centralizar o acesso às imagens que serão utilizadas durante a criação do jogo.

A classe *FolhaSprite* tem a função de atender o requisito GRA3. A imagem passada na construção desta classe deve ter os quadros de animação da entidade em uma grade 4x2, ou seja com 8 imagens igualmente centralizadas. Estas imagens serão utilizadas para as animações da entidade.

A classe *GameCamera* é responsável por atender os requisitos GRA1 e GRA2. Sendo assim os métodos centerOnPlayer e mover permitem ter o controle da visão de jogo, centralizando a câmera em uma entidade e movendo livremente, respectivamente.

As classes *Animacao* e *EntidadeSprite* são responsáveis por atender o requisito GRA3 em apoio a *FolhaSprite* e também o requisito GRA4. *EntidadeSprite* armazena as animações da entidade e se responsabiliza em alterá-las conforme o estado da entidade. Uma entidade que está virada para a posição esquerda terá a animação *movEsq* ativa, por exemplo. A velocidade da animação será determinada na construção da *Animacao*.

Para manter os indicadores de jogo disponíveis no estado de jogo, foi criada a classe *HUDBase* que nada mais é que uma classe que cria janelas especializadas, atendendo assim o requisito GRA5. Caso o desenvolvedor queira utilizar um *HUD*, deve implementar esta classe.

4.2.4 Desenvolvimento do módulo Input

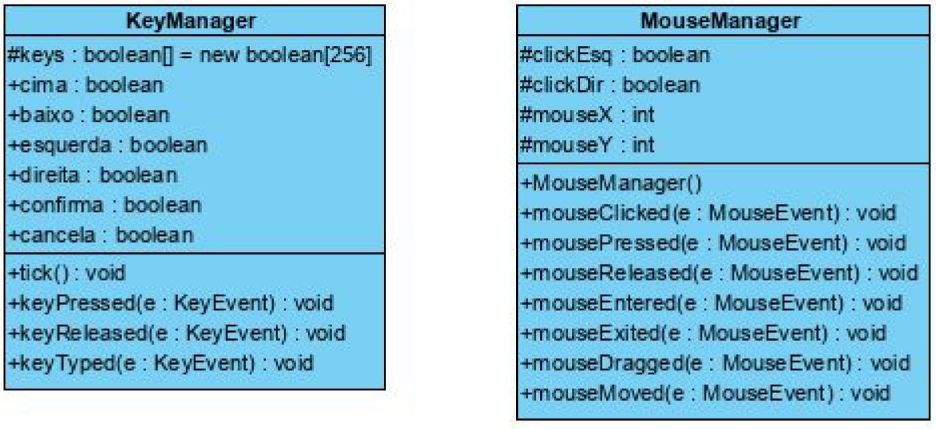
O módulo Input é responsável por capturar as entradas do jogador, interpretá-las e atualizá-las em mundo de jogo, sendo assim a interface entre o jogador e o jogo.

A classe *KeyManager* é responsável pelas entradas de teclado, tendo mapeadas as teclas: Enter para confirmação, Esc para cancelamento e as setas direcionais para movimento, atendendo assim o requisito INP1. O desenvolvedor pode implementar seu próprio *KeyManager* para utilizar teclas diferentes, desde que herde esta classe.

A classe *MouseManager* é responsável pelas entradas de mouse, tendo mapeados os botões esquerdo e direito. Além disso, controla a área de movimento do mouse e os eventos de clique, clicado e liberado o clique e pressionado e arrastado o clique.

A Figura 11 representa o diagrama de classes do módulo Input.

Figura 11 - Diagrama de classes do módulo Input



4.2.5 Desenvolvimento do módulo Janelas

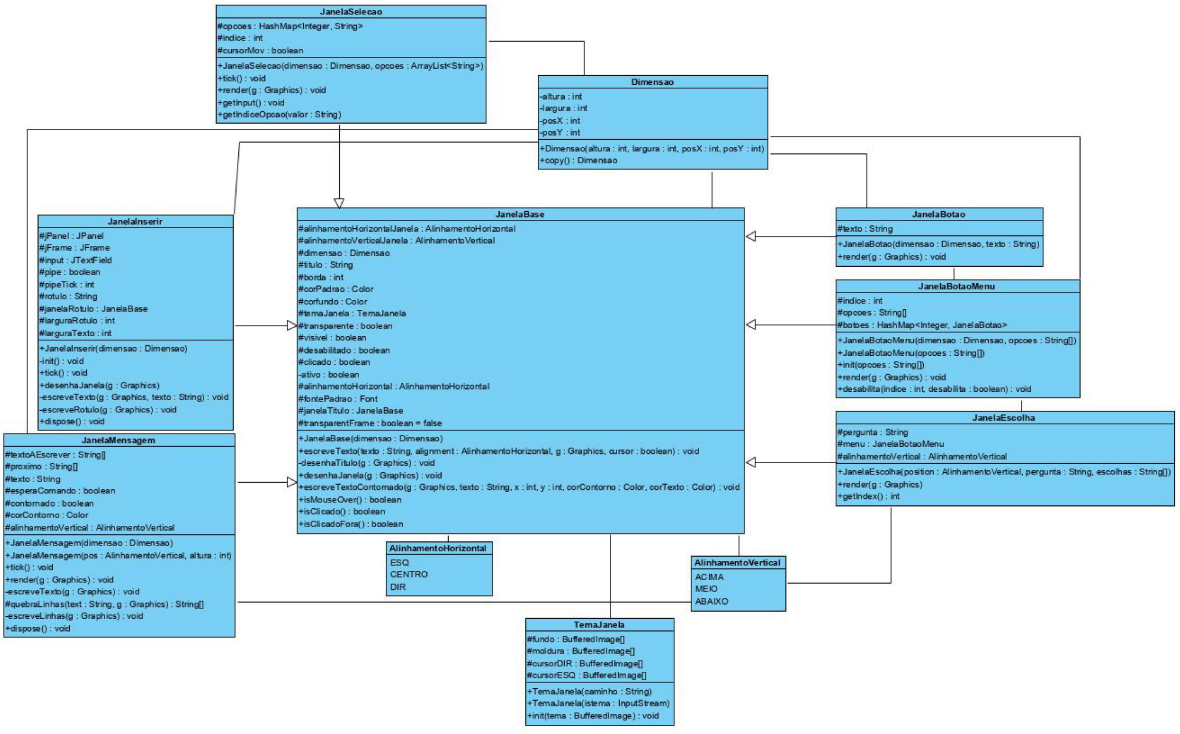
O módulo Janelas manipula a exibição de janelas de diálogo no ambiente de jogo. Menus, textos e indicadores de jogo são exemplos de uso das janelas.

A classe *JanelaBase* fornece os atributos e métodos básicos para implementação de janelas na ferramenta jTiled2D o que por si já atende os requisitos JAN1 e JAN2.

Para atender o requisito JAN3, foram criadas as classes *JanelaSelecao*, *JanelaBotao*, JanelaBotaoMenu e *JanelaEscolha*. Estas são responsáveis por exibir um menu de opções selecionáveis para o jogador.

A Figura 12 representa o diagrama de classes do módulo Janelas.

Figura 12 - Diagrama de classes do módulo Janelas



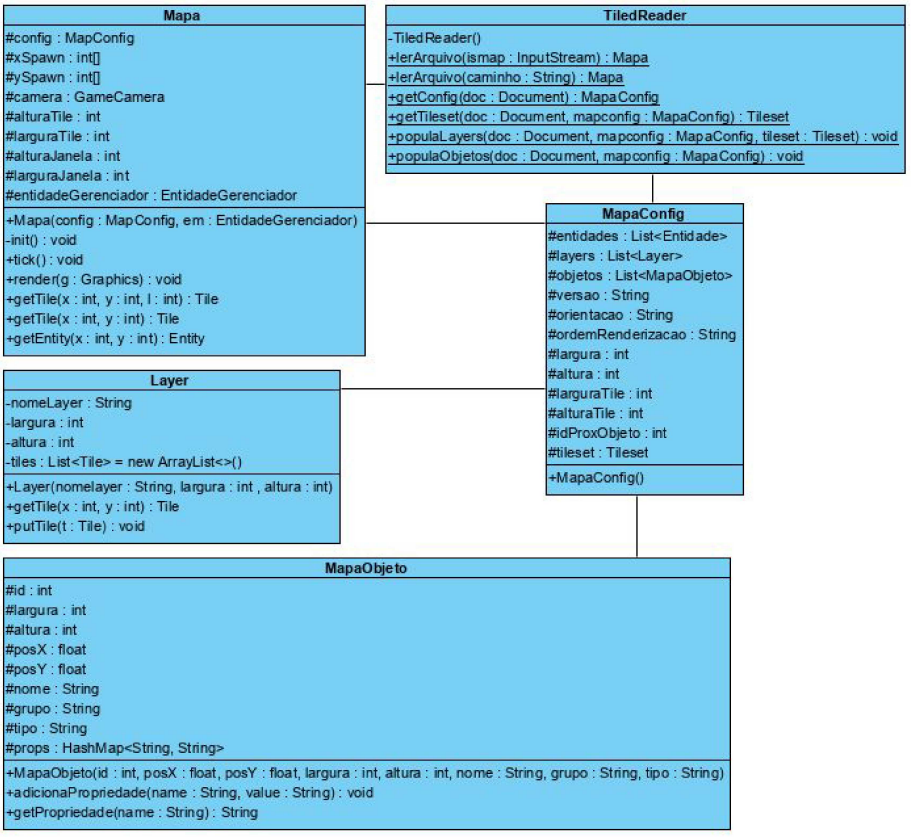
As janelas são renderizadas e atualizadas através dos métodos *render* e *tick* disparados através da classe *Game*.

4.2.6 Desenvolvimento do módulo Mapas

O módulo Mapas é responsável por gerenciar a importação de mapas, renderização e atualização dos tiles. Os mapas são compostos por camadas (Layers) permitindo aos tiles serem sobrepostos para dar noção de profundidade, atendendo assim o requisito MAP1. Os métodos *getTile* e *getEntity* são utilizados em apoio ao módulo de entidades para conferir colisão.

A Figura 13 representa o diagrama de classes do módulo Mapas.

Figura 13 - Diagrama de classes do módulo Mapas



A classe *TiledReader* foi desenvolvida especificamente para suporte nativo a importação de mapas de formato TMX, gerados na ferramenta Tiled, atendendo assim o requisito MAP2. Se o desenvolvedor preferir utilizar outra ferramenta para criação de mapas, pode criar seu próprio importador que retorne um objeto Mapa.

A classe *MapaObjeto* foi criada para leitura de objetos inseridos no mapa que eventualmente deverão ser tratados como entidades.

4.2.5 Desenvolvimento do módulo Tiles

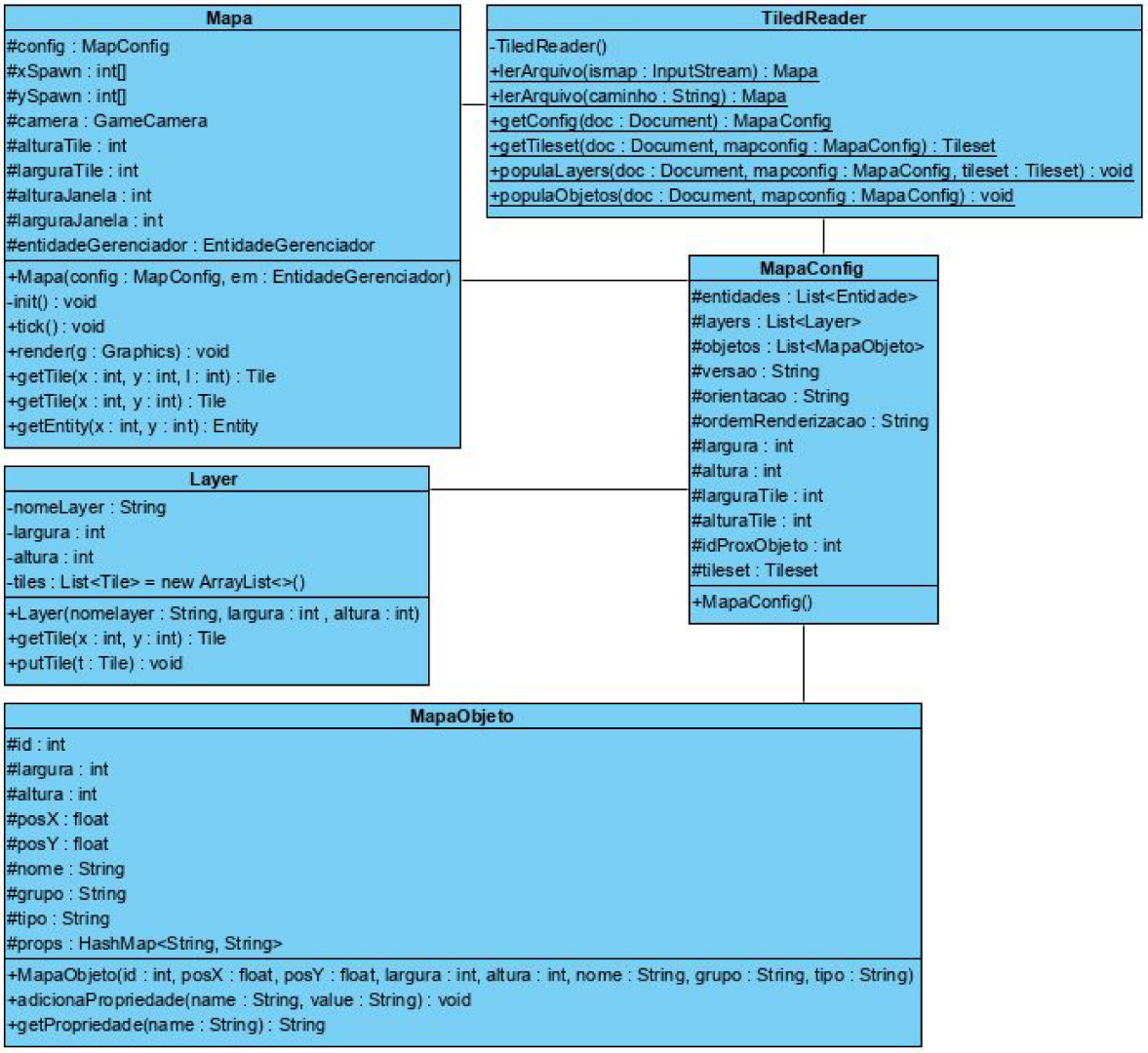
O módulo Tiles gerencia a criação dos tiles através de uma coleção de tiles chamada Tileset. Assim como a *FolhaSprite* atua para as entidades, o Tileset atua para os tiles, ou seja, uma imagem em grade que terá em suas células a imagem do tile, atendendo assim o requisito TIL2.

Cada camada de mapa tem uma lista de tiles para serem exibidos e cada tile armazena sua propriedade *solido* que indica se colide ou não com entidades, atendendo assim o requisito TIL1. Um tile sólido não permite passagem.

O requisito TIL3 é atendido no módulo Mapa, no importador *TiledReader*. As propriedades dadas aos tiles na ferramenta Tiled são repassadas ao Tile em jTiled2D.

A Figura 14 representa o diagrama de classes do módulo Tiles.

Figura 14 - Diagrama de classes do módulo Tiles



5 Desenvolvimento do Jogo Exemplo

Para demonstrar as funcionalidades da biblioteca jTiled2D é desenvolvido o jogo de exemplo chamado Sample Game.

Este jogo terá dois estados sendo estes:

- **MenuState**: onde o jogador será submetido a uma tela com os botões iniciar e sair; e
- **MapState**: onde ocorrerá a ação e imersividade do mundo de jogo como conversar com personagens, interagir com placas e atacar inimigos;

A classe inicial será chamada *StartGame* e, além do método main, terá as configurações básicas para uso da biblioteca.

Além disso será criada a classe *Resources* para fácil acesso aos recursos gráficos do jogo.

5.1 Recursos

O primeiro passo é escolher as imagens de entidade, os tilesets e imagens de janela que serão utilizados. Para fácil acesso, estes arquivos são armazenados na pasta *src/gfx* do projeto.

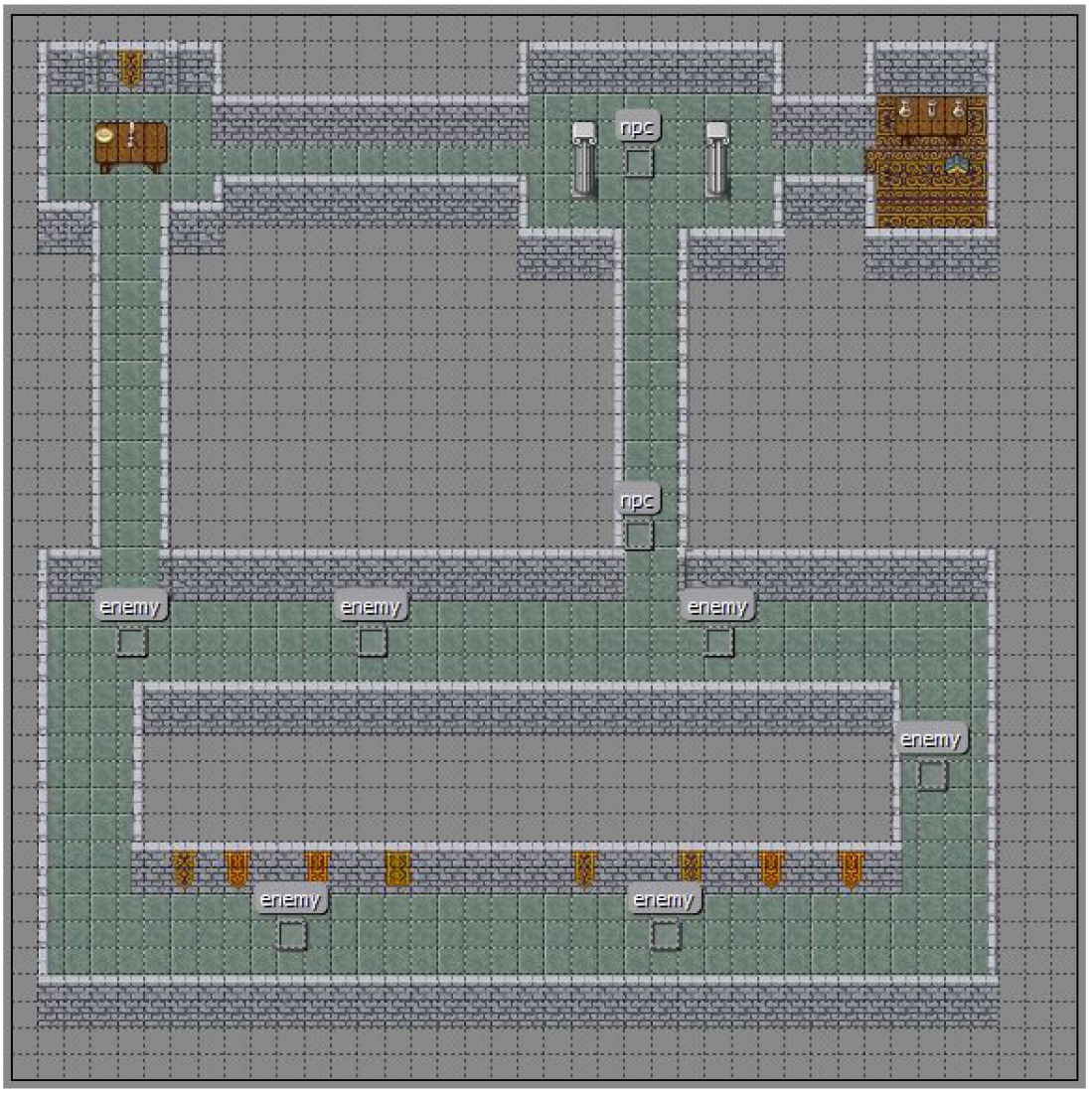
Em seguida, foi utilizada a ferramenta Tiled para desenhar o mapa de jogo, utilizando o tileset escolhido no passo anterior.

No mapa desenvolvido, como apresenta a Figura 16, foram adicionados dois NPCs com diálogos e movimentações distintas e seis inimigos. Foi convencionado para os NPCs o uso das seguintes propriedades:

- **dialogo**: representa o texto que será exibido ao interagir com este NPC; e
- **tipoMovimento**: que determina a movimentação do NPC no mundo de jogo.

Para os inimigos, apenas tipoMovimento será utilizada e com a mesma intenção.

Figura 15 - Mapa desenvolvido na ferramenta Tiled



O mapa, exportado em formato .tmx, foi armazenado na pasta *src/gfx*.

Foi criada então, no projeto, uma nova pasta em *src* com o nome de *gfx*, onde são armazenados os elementos gráficos do jogo. para organização dos atores chamada *actors.*

A indicação de solidez do tile foi informada na edição do tileset através da propriedade *solid* com o valor *true*.

Ainda dentro da pasta *src/gfx* foi criada a classe *Resources* com o fim de ter fácil acesso aos recursos gráficos selecionados.

Os recursos são tratados com o seu caminho em formato de texto com exceção dos *Tilesets* que devem ser tratados como um *InputStream*.

A figura 16 apresenta o código final da classe Resources.

Figura 16 - A classe Resources

```
package src.gfx;

import java.io.InputStream;

public class Resources {

    public final String windowskin = getClass().getResource( name: "windowskin.png").getPath();
    public final String mapa1 = getClass().getResource( name: "testMap.tmx").getPath();
    public final String characters = getClass().getResource( name: "characters.png").getPath();

    public final InputStream tileset = getClass().getResourceAsStream( name: "indoor.png");

}
```

5.2 Entidades

O *Sample Game* tem três entidades bem definidas: o jogador, os inimigos e outros personagens não jogáveis (NPCs, da sigla em inglês). Para dar vida a estas entidades, é utilizada a classe *Entidade* da biblioteca jTiled2D.

Na estrutura do projeto foi criada a pasta *src/actors* para armazenar as classes de entidades. Em seguida as classes *Player, NPC e Enemy* foram criadas na mesma pasta.

A classe Player é distinta por representar o avatar do jogador e, no Sample Game, recebe inputs de teclado para se mover pelo mapa. Para isso a classe implementa a classe Entidade da biblioteca jTiled2D e, no ciclo de jogo - ou *tick* - chama o método *getInput* que é responsável por habilitar o movimento via teclado para qualquer instância desta classe. Por padrão as teclas mapeadas são as setas direcionais e o movimento acontece tile à tile.

Figura 17 - Construtor e método tick da classe Player

```
public class Player extends Entidade {

    public Player(int x, int y, EntidadeSprite es) throws Exception {
        super( x: x * Config.getInstance().getLarguraTile(),
                y: y * Config.getInstance().getAlturaTile(),
               es);
    }

    @Override
    public void tick() {
        getInput();
        super.tick();
    }
```

A Figura 17 apresenta o construtor e o método *tick* da classe Player.

A classe NPC tem duas funcionalidades: apresentar um diálogo ao jogador quando a tecla de ação for disparada em sua direção e ter seu movimento no mapa pré-determinado por uma propriedade do objeto do mapa desenhado na ferramenta

Tiled. Diferente do jogador, não há inputs para essa classe, toda ação será feita através do Player contra o NPC.

Iniciando pelo movimento, é utilizada a propriedade *tipoMovimento* na criação do objeto no mapa e a palavra chave *random* é o ponto de entrada para definir se o NPC se movimenta.

A Figura 18 apresenta o código da classe NPC com seu construtor, o método *tick* e o controle de movimentação.

Figura 18 - A classe NPC

```
public class NPC extends Entidade {
    private String tipoMovimento;
    private String dialogo;
    private int ticksParaMover;

    public NPC(float x, float y, EntidadeSprite es, String tipoMovimento) throws Exception {
        super(x, y, es);
        this.tipoMovimento = tipoMovimento;
    }

    @Override
    public void tick() {
        if (!movendo)
            MovimentoNPC();
        super.tick();
    }

    private void MovimentoNPC() {
        //Aguardar 30 ticks ate mover novamente
        if(ticksParaMover == 0) {
            ticksParaMover = 30;
            movendo = false;
            if (tipoMovimento.equals("random")) {
                moverAleatorio();
            }
        } else{
            ticksParaMover --;
        }
    }
```

No controle de movimentação foi construído um delay para que o movimento seja suave. Como foi utilizada a configuração padrão de 60 frames por segundo, a cada meio segundo (ou seja, 30 ticks) é decidida a movimentação do NPC.

Para o controle de diálogo, a propriedade *dialogo* com seus acessos é suficiente. A classe MapState - que será desenvolvida no decorrer deste capítulo - é responsável por popular e exibir o seu valor em uma janela de texto.

Os inimigos também podem ser considerados NPCs, pois não são personagens jogáveis, e para o Sample Game eles tem apenas uma especialização: causar dano no jogador. O controle de dano será feito pela classe MapState, portanto, a classe Enemy pode herdar a implementação de NPC perfeitamente.

A Figura 19 apresenta o código da classe Enemy.

Figura 19 - A classe Enemy

```
public class Enemy extends NPC {

    public Enemy(float x, float y, EntidadeSprite es, String tipoMovimento) throws Exception {
        super(x, y, es, tipoMovimento);
    }
}
```

Com isso as entidades estão implementadas. O próximo passo agora é desenvolver os estados de jogo.

5.3 MenuState

O primeiro estado do jogo é o MenuState, onde são exibidas duas opções ao jogador: iniciar e sair.

Na estrutura do projeto, foi criada a pasta *src/states* para armazenar os estados de jogo. A classe *MenuState* foi criada nesta pasta.

O *MenuState* é um estado de jogo, logo deve herdar a classe *Estado* e, como exigido, deve implementar os métodos *init, tick* e *render*.

Este estado deve apresentar ao usuário um menu clicável com duas opções: iniciar e sair; sendo que a primeira deve mudar o estado para o mapa e a última fecha o programa. A propriedade *menu* do tipo *JanelaBotaoMenu* representa este menu.

Para controle de inicialização do estado, a propriedade *init* será utilizada. Ela indica se os botões já foram instanciados para serem renderizados e atualizados a cada *tick*.

A propriedade *resources* dará acesso ao mapa a ser carregado no próximo estado.

Figura 20 - Init do MenuState

```
public class MenuState extends Estado {

    private boolean init;
    private JanelaBotaoMenu menu;
    private Resources resources;

    public MenuState() { resources = new Resources(); }

    @Override
    public void init() {
        menu = new JanelaBotaoMenu(new String[]{"Iniciar", "Sair"});
        menu.setAlinhamentoVertical(JanelaBase.AlinhamentoVertical.MEIO);
        menu.setAlinhamentoHorizontal(JanelaBase.AlinhamentoHorizontal.CENTRO);
        init = true;
    }
```

Na Figura 20 é apresentado que o método *init* descreve o algoritmo de inicialização pós construção do estado e deve ser implementado. Neste caso o menu será inicializado e posicionado na tela.

O método *tick* representa o ciclo deste estado a cada tick de jogo. Neste caso, será conferido o clique nas opções do menu. A classe JanelaBotaoMenu é responsável por informar se houve alguma ação com o menu e retorna o índice que sofreu a interação através do método *getIndice*. Na Figura 21 é possível observar a implementação do método tick. O índice 0 (zero) representa a ação *Iniciar*, logo um novo estado será criado.

Figura 21 - Tick do MenuState

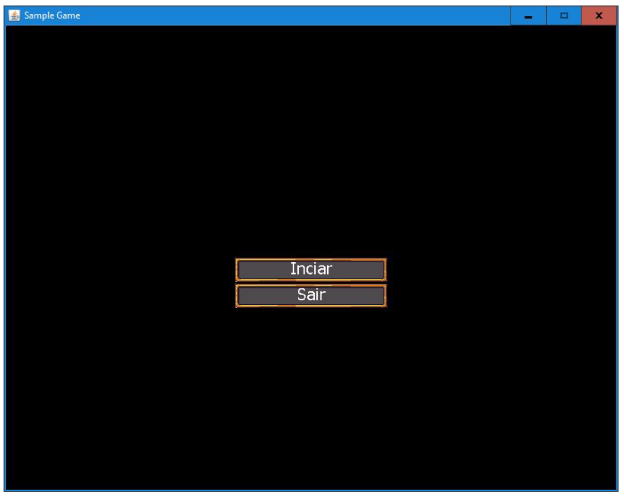
```
@Override
public void tick() {
    if (init) {
        switch (menu.getIndice()) {
            case 0: //iniciar
                setEstado(new MapState(resources.mapa1));
                break;
            case 1: //sair
                System.exit( status: 0);
                break;
        }
    }
}

@Override
public void render(Graphics g) {
    if (!init) init();
    menu.render(g);
}
```

O método *render* é responsável por desenhar as imagens na tela de jogo e é invocado a cada tick de jogo. No caso do MenuState, é preciso garantir que a janela seja desenhada de acordo e somente após o estado ter sido completamente inicializado.

A Figura 22 demonstra o MenuState em tempo de execução.

Figura 22 - MenuState em tempo de execução



5.3 MapState

A representação do mundo de jogo se dará por este estado. Esta classe implementa Estado, logo deve implementar também os métodos *init, tick* e *render* como todo estado de jogo. Como um mapa será desenhado e controlado nesse estado, é exigido em seu construtor o caminho do arquivo de extensão TMX através da propriedade *path*. A Figura 23 apresenta as propriedades e o construtor do MapState.

Figura 23 - Propriedades e construtor do MapState

```
public class MapState extends Estado {

    private Mapa mapa;
    private Player player;
    private JanelaMessage janelaMessage;
    private List<NPC> npcs;
    private List<Enemy> enemies;

    //Controla a exibição de diálogos
    private boolean emDialogo;

    MapState(String path) {
        Config.getInstance().setTilesetAtual(new Resources().tileset);
        this.mapa = TiledReader.lerArquivo(path);
        init();
    }
```

A propriedade *mapa* representa o mapa carregado pelo caminho *path* através do método *init*, assim como *player* para o avatar do jogador e as listas de NPCs e Inimigos para estas entidades. Já a propriedade *janelaMessage* representa a janela de diálogo ao interagir com um NPC.

O método *init* - apresentado na Figura 24 - ganha uma importância neste estado por ser responsável por carregar o arquivo de mapa, seus tiles, objetos e criar as entidades de acordo, populando suas propriedades. As entidades do EntidadeGerenciador do mapa são as que serão afetadas pelos métodos *tick* e *render* após a inicialização completa do estado.

Figura 24 - Init do MapState

```
@Override
public void init() {
    EntidadeSprite es = new EntidadeSprite(new Resources().characters,  veloAnim: 100, FolhaSprite.PosSprite.POS7);
    try {
        player = new Player( x: 12,  y: 5, es);
        Config.getInstance().getGameCamera().centralizaEm(player);
        mapa.getEntidadeGerenciador().adicionaEntidade(player);
    } catch (Exception e) {
        e.printStackTrace();
    }
    //Inicializa a janela de mensagem
    janelaMessage = new JanelaMessage(JanelaBase.AlinhamentoVertical.ABAIXO,  altura: 100);

    populaNPCs();
    populaInimigos();
}
```

Figura 25 - Tick do MapState

```
@Override
public void tick() {
    this.mapa.tick();
    if (emDialogo) {
        this.mapa.getEntidadeGerenciador().desativarEntidades();
        this.mapa.getEntidadeGerenciador().desativarAcaoEntidades();
        if (janelaMessage.isVisivel()) {
            janelaMessage.tick();
        } else {
            emDialogo = false;
        }
    } else {
        this.mapa.getEntidadeGerenciador().ativarEntidades();
        if (player != null) {
            Config.getInstance().getGameCamera().centralizaEm(player);
        }
        //Confere se algum NPC tem algo a dizer
        confereAcaoNPC();
        confereAcaoInimigo();
    }
}
```

O método *tick* (Figura 25) nesta classe representa o fluxo no mundo de jogo, determinando quando ou não o jogador pode agir. Para tal foi determinado um estado *emDialogo* que controla se um diálogo de NPC é exibido e, portanto, se o jogador pode mover seu avatar. Além disso, é conferido neste método também se alguma ação de NPC ou Inimigo foi disparada.

Se algum NPC ou Inimigo agiu (ou sofreu uma ação do jogador) então os métodos *confereAcaoNPC* e *confereAcaoInimigo* executarão essas ações. São similares em sua estrutura sendo a diferença na ação que será executada. Para NPCs o diálogo é exibido e para Inimigos a entidade será removida da lista de entidades do mapa, logo não será mais elencada para os métodos *tick* e *render*.

A Figura 26 exibe um NPC e alguns inimigos do MapState em tempo de execução.

Figura 26 - MapState em tempo de execução.

Na classe Player, foi implementado o método *disparaAcao* que determina o que deve acontecer ao pressionar a tecla Enter. Se alguma outra entidade estiver próxima ao jogador e na sua linha de visão, dispara a ação pré-determinada daquela entidade. No Sample Game, isto pode ser o diálogo de um NPC ou um ataque em um Inimigo.

Figura 27 - Método disparaAcao em Player

```
@Override
protected void disparaAcao() {
    //Confere se tem alguma entidade adjacente para interagir
    int eX = this.getTileX();
    int eY = this.getTileY();
    switch (dir) {
        case CIMA:
            eY -= 1;
            break;
        case BAIXO:
            eY += 1;
            break;
        case DIREITA:
            eX += 1;
            break;
        case ESQUERDA:
            eX -= 1;
            break;
    }
    Entidade e = mapa.getEntity(eX, eY);
    if (e != null)
        e.acaoDisparada = true;
}
```

5.5 Start Game

A classe mais importante e mais simples de todas é a StartGame. Esta implementa o método *main* (iniciando um programa Java) e inicializa a biblioteca jTiled2D.

A Figura 28 apresenta a implementação da classe StartGame.

Figura 28 - StartGame

```
public class StartGame {
    public static void main(String[] args) {
        try {
            Resources resources = new Resources();
            Config.getInstance().setFontePadrao(new Font( name: "Tahoma", Font.PLAIN,  size: 22));
            Config.getInstance().setCorPadrao(Color.WHITE);
            Launcher.launchGame( nomeJogo: "Sample Game",  larguraJanela: 800,  alturaJanela: 608,
                    fps: 30, resources.windowskin, new MenuState());
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

O código completo do SampleGame está disponibilizado no seguinte repositório git:

https://gitlab.com/diego.feijo/jtiled2d-sample-game

O código da biblioteca jTiled2D e o arquivo .jar são encontrados neste repositório:

https://gitlab.com/diego.feijo/TiledGame_Framework

5.4 Considerações

Para o desenvolvimento do SampleGame foram utilizados os seguintes conceitos de Programação Orientada a Objetos:

- Estruturas de controle de código, como *if* e *switch;*
- Estruturas de loop, como *for*;

- Implementação de classes abstratas;
- Herança de classe; e
- Controle de listas.

Este conhecimento é o mínimo necessário para desenvolver jogos simples com a ferramenta jTiled2D. Sem o uso da biblioteca, os seguintes conhecimentos são necessários:

- Programação paralela e controle de concorrência, para controle do ciclo de jogo e renderização;
- Carga de arquivos na leitura de mapa e imagens de tiles, entidades e janelas;
- Manipulação de imagens, com estratégias de ganho de desempenho (como o corte nas imagens de entidades e janelas);
- Uso de padrões de desenvolvimento como *Singleton* e *State*;
- Física básica de jogos, como o cálculo de colisão e distância entre entidades e tiles;
- Conceito de Interfaces; e
- Listeners (para inputs de teclado e mouse).

Para os alunos que pretendem utilizar o jTiled2D durante a disciplina de Análise e Projeto de Sistemas - motivadora deste projeto - provavelmente ainda não terão aprendido durante o curso grande parte do conhecimento que a biblioteca abstrai. Programação paralela, por exemplo, será ministrada em fases posteriores. Logo o uso da ferramenta serve como um suporte até que o desenvolvedor adquira o conhecimento desejado e possa utilizar engines e frameworks comerciais com a

robustez desejada para desenvolver um jogo profissional. As ferramentas apresentadas no capítulo 3 são uma excelente alternativa para o desenvolvedor que já esteja seguro de seu conhecimento e queira criar seu jogo em Java.

6 Conclusões

Com o desenvolvimento do jogo de exemplo observa-se que o principal objetivo deste trabalho foi atingido. Com o uso da biblioteca JTiled2D foi possível desenvolver um jogo virtual com imersão no mundo de jogo através de um cenário, controle do avatar do jogador, interação com personagens no mundo de jogo e uma pequena dose de ação com confronto de inimigos o que permite quantificar a experiência do jogador. Tudo isso com o uso de estruturas básicas da linguagem Java como *if* e *for*, abstraindo assim o acesso a outras estruturas complexas de programação - como as *threads* usadas para o *tick* e *render* do jogo - e o acesso aos recursos gráficos assim como sua exibição.

A biblioteca desenvolvida permite, portanto, ao aspirante desenvolvedor de jogos virtuais dar os primeiros passos na criação de jogos utilizando conhecimento básico da linguagem Java.

Além disso, um objetivo não listado - porém inerente ao processo de desenvolvimento deste documento - foi atingido. Trata-se do aprendizado do autor quanto ao que se entende por jogo e desenvolvimento de jogos digitais. Ao pesquisar o significado de jogo com a visão de outras áreas que não a de tecnologia tornou-se claro a importância dessa interação lúdica na sociedade. Esse conhecimento abriu um novo caminho permitindo vislumbrar uma possível carreira no que hoje é um hobby.

6.1 Sugestões para Trabalhos Futuros

Trabalhos futuros podem adicionar a funcionalidade de acesso ao tratamento de efeitos sonoros em camadas, como música de fundo ou som ambiente e sons de ação, entidades, objetos de mapa e afins.

Outra sugestão seria a criação de um sistema de persistência, onde o jogador possa salvar o seu progresso e continuar o jogo de onde parou.

Para tornar ainda mais usual a experiência com a ferramenta, pode-se criar uma interface gráfica para interação com os métodos existentes e criação automática de código, criando assim uma camada interativa para o desenvolvedor, sem a necessidade deste trabalhar diretamente no código.

7 Referências

CAILLOIS, R. **Os jogos e os homens**: a máscara e a vertigem. [S.l.]: Editora Vozes Limitada, 2017.

DESIGN Pattern. Disponível em: https://sourcemaking.com/design_patterns/singleton. Acesso em: 12 abr. 2020.

HUIZINGA, J. **Homo Ludens:** O jogo como elemento da cultura. São Paulo: Perspectiva, 2017. v. 8.

LINDEIJER, Thorbjorn. **Tiled Map Editor.** Disponível em: <https://www.mapeditor.org/>. Acesso em: 15 out. 2019.

MONKEYENGINE. **JMonkeyEngine.** Disponível em: <http://jmonkeyengine.org/>. Acesso em: 07 out. 2019.

NEWZOO. **Brazil Games Market 2018.** 2018. Disponível em: https://newzoo.com/insights/infographics/brazil-games-market-2018/. Acesso em: 28 mar. 2020.

OPENGAMEART.ORG. 2020. Disponível em: https://opengameart.org/. Acesso em: 28 mar. 2020.

SALEN, K.; ZIMMERMAN, E. **Regras do jogo:** fundamentos do design de jogos. São Paulo: Blucher: [s.n.], 2012. v. 3.

TAYLOR, Laurie N.. **Video Games:** Perspective, Point-of-view, and Immersion. 2002. Disponível em: <https://www.researchgate.net/profile/Laurie_Taylor2/publication/35486642_Video_games_electronic_resource_perspective_point-of-view_and_immersion/links/551e9f450cf2a2d9e13c6f53/Video-games-electronic-resource-perspective-point-of-view-and-immersion.pdf>. Acesso em: 29 jun. 2019.

THE NPD GROUP INC. **New Report from The NPD Group Provides In-Depth View of Brazil's Gaming Population.** 2015. Disponível em: https://www.npd.com/wps/portal/npd/us/news/press-releases/2015/new-report-from-the-npd-group-provides-in-depth-view-of-brazils-gaming-population/. Acesso em: 28 mar. 2020.

WILKE, Steffen; WILKE, Matthias. **LITIengine.** Disponível em: <https://litiengine.com/>. Acesso em: 04 out. 2019.

ZECHNER, Mario. **LibGDX.** Disponível em: <https://libgdx.badlogicgames.com/>. Acesso em: 07 out. 2019.

Apêndice A - Relatório Sobre o Trabalho de Conclusão de Curso JTiled 2D

Diego Feijó

Departamento de Informática e Estatística - Universidade Federal de Santa Catarina - Florianópolis, SC - Brasil

diego.feijo@hotmail.com

Abstract. *This report describes the development of the final paper JTiled2D: a library for developing 2D games in a top-down perspective in the Java language. The methodology used to develop the library and the example game created will be presented.*

Resumo. *Este relatório descreve o desenvolvimento do trabalho de conclusão de curso JTiled2D: uma biblioteca para desenvolvimento de jogos 2D em perspectiva top-down na linguagem Java. Será apresentada a metodologia utilizada para desenvolvimento da biblioteca e o jogo de exemplo criado.*

1. Introdução

O trabalho tem por objetivo apresentar o desenvolvimento de uma biblioteca que auxilie na construção de jogos digitais (*games*) em duas dimensões (2D) baseados em uma matriz de gráficos de tamanho uniforme.

A principal motivação do autor foi sua experiência acadêmica na disciplina Análise e Projeto de Sistemas onde foi proposto a criação de um jogo para exemplificar o processo de desenvolvimento de software de ponta a ponta, desde a análise de requisitos até a entrega. A inexperiência no campo de desenvolvimento de games foi vista como um desafio e, depois de buscar por várias bibliotecas que não atenderam a necessidade imediata e pensando ser esta uma dificuldade compartilhada entre os colegas, surgiu a ideia de criar sua própria, para que sirva como primeiro passo para os aspirantes a desenvolvimento de jogos na linguagem Java - linguagem escolhida por ser um dos requisitos não funcionais exigidos na disciplina.

Para a elicitação de requisitos, o autor escolheu destrinchar conceitos básicos de jogos, partindo de um ensaio sobre a representação dos jogos na humanidade comparando com estudos de mesmo viés e caminhando até a representação de jogos digitais e suas diferentes perspectivas de visão de jogo.

Após elicitar os requisitos, foi apresentada a estrutura de classes da biblioteca desenvolvida com uma explicação de sua estrutura funcional e como atende os requisitos levantados.

Por fim foi apresentado o desenvolvimento de um jogo exemplo utilizando a biblioteca com suas principais funcionalidades.

2. A biblioteca JTiled2D

A base para cálculo de posicionamento, colisão, sobreposição e renderização de gráficos da biblioteca são os *tiles*. Tiles são células de uma grade de tamanho uniforme e representam pedaços de um mundo de jogo, como peças de um quebra-cabeça.

Uma paleta de tiles é conhecida como *tileset*. Um mapa de jogo desenhado em tela sendo gerenciado pela biblioteca JTiled2D terá ao menos um arquivo de imagem que representa o *tileset*. Este arquivo será subdividido em *tiles* e guardado em memória. O desenvolvedor pode informar à biblioteca qual o tamanho em pixels de cada tile. A Figura 1 apresenta parte de um tileset que pode ser utilizado para desenhar um mapa de jogo.

A biblioteca foi dividida em sete módulos interdependentes sendo estes: *entidades*, *estados, gráficos, input, janelas, mapas* e *tiles*. As principais funcionalidades destes módulos serão explicadas nas seções seguintes.

Figura 1 - Exemplo de tilesets
Fonte: OpenGameArt.Org[16]

- **Entidades:** responsável pela representação das entidades de jogo. São entendidos como entidades os atores do jogo como inimigos, personagens não jogáveis e o próprio jogador.
- **Estados:** os jogos desenvolvidos pela biblioteca são divididos em estados e cada um destes estados tem um conjunto de regras únicas para controle do ambiente de jogo. Um menu inicial, onde o jogador escolhe entre iniciar um novo jogo ou configurar opções, é um estado, enquanto o mapa de jogo é outro.
- **Gráficos:** todo controle de renderização de imagens é gerido por este módulo. Quando um *tilesets* é carregado para ser utilizado em um mapa, este módulo calcula sua divisão e entrega uma coleção de *tiles* ao módulo *mapas*. O mesmo acontece com as imagens do avatar do jogador, janelas e qualquer outro elemento gráfico a ser exibido em tela.
- **Input:** a interface entre o jogador e o jogo. Permite o mapeamento de teclas do teclado e botões do mouse para interpretação de comandos. Comandos de confirmação, cancelamento e setas direcionais para movimento são mapeados por padrão.
- **Janelas:** uma coleção de elementos gráficos para exibição de janelas de menu, diálogo e monitores de jogo. As principais janelas estão prontas para uso de forma genérica, possibilitando criação de elementos de interação com o mínimo esforço.
- **Mapas:** controla a renderização de mapas de jogo através de importação de arquivos. Possui suporte nativo a arquivos de extensão TMX, gerados pela ferramenta Tiled. É possível criar o próprio importador de mapas para qualquer formato que possa ser traduzido para uma coleção de tiles.
- **Tiles:** gestão dos *tiles* em tempo de jogo. Tem papel similar ao módulo *entidades* porém voltado aos elementos de mapa.

3. Jogo Exemplo

Para teste e apresentação das funcionalidades da biblioteca JTiled2D foi desenvolvido um jogo de exemplo. Foi criado um menu inicial e um mapa simples onde o jogador

[16] OPENGAMEART.ORG. Disponível em: https://opengameart.org/. Acesso em: 28 mar. 2020

pode controlar o seu avatar entre tiles sólidos - que impedem a passagem - e livres - que permitem a passagem. As figuras 2 e 3 apresentam exemplos do menu de jogo e mapa, respectivamente.

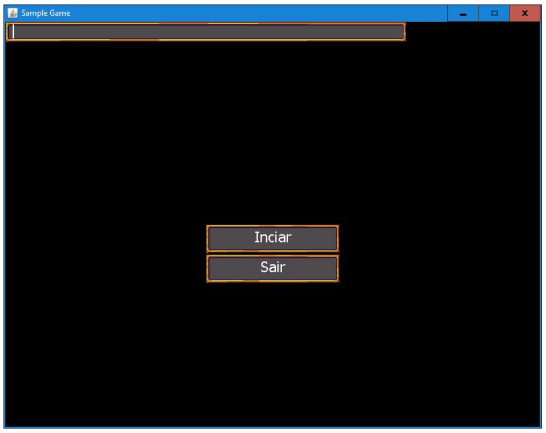


Figura 2 - Menu inicial do jogo exemplo

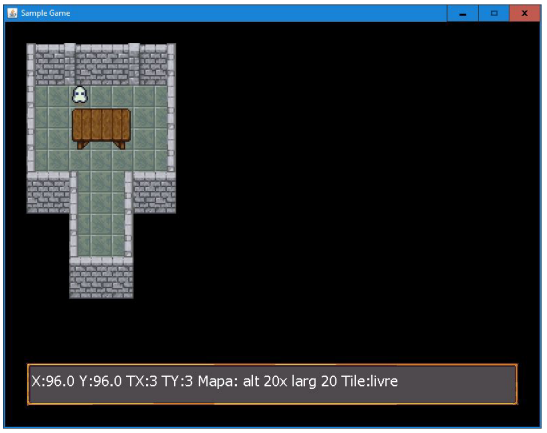


Figura 3 - Mapa do jogo exemplo

4. Conclusão

Com o desenvolvimento do jogo exemplo ficou evidente que a biblioteca atingiu seu objetivo de servir como auxiliar no desenvolvimento de jogos 2D baseados em tiles. O jogo desenvolvido conta com renderização de um mapa desenvolvido na ferramenta Tiled, um avatar de personagem de jogador controlável, controle de colisões e janelas de menu e mensagens. Para os trabalhos futuros foi sugerido a adição de controle de sons e armazenamento de progresso de jogo.

5. Referências

FEIJÓ, Diego. JTiled2D: uma biblioteca para desenvolvimento de jogos 2d em perspectiva top-down na linguagem java. 2020. 64 f. TCC (Graduação) - Curso de Sistemas de Informação, Universidade Federal de Santa Catarina, Florianópolis, 2020.

OPENGAMEART.ORG. 2020. Disponível em: https://opengameart.org/. Acesso em: 28 mar. 2020.