Alfredo José de Paula Barbosa - 16890
Michell Stuttgart Faria - 16930
Paulo Vicente Gomes dos Santos - 15993
Orientador: Prof. Dr. Enzo Seraphim

SAGA Game Library
Biblioteca para desenvolvimento de jogos eletrônicos 2D

Itajubá - MG

Alfredo José de Paula Barbosa - 16890
Michell Stuttgart Faria - 16930
Paulo Vicente Gomes dos Santos - 15993
Orientador: Prof. Dr. Enzo Seraphim

SAGA Game Library
Biblioteca para desenvolvimento de jogos eletrônicos 2D

Monografia apresentada para o Trabalho de Diploma do curso de Engenharia da Computação da Universidade Federal de Itajubá.

Universidade Federal de Itajubá - UNIFEI
Instituto de Engenharia de Sistemas e Tecnologias da Informação
Engenharia da Computação

Itajubá - MG

Sumário

Lista de Figuras

Resumo

1 Introdução p. 6

1.1 Motivação . . . . . p. 7

1.2 Objetivos . . . . . p. 7

2 Motivação p. 9

3 Metodologia p. 10

3.1 O software Tiled . . . . . p. 10

4 Desenvolvimento p. 15

4.1 Especificação dos requisitos do projeto . . . . . p. 16

4.1.1 Baixa curva de aprendizado . . . . . p. 16

4.1.2 Gerenciamento automático de recursos . . . . . p. 17

4.1.3 Arquitetura multi-plataforma . . . . . p. 17

4.1.4 Suporte a diversos formatos de arquivos . . . . . p. 17

4.1.5 Renderização acelerada por *hardware* e gráficos 2D . . . . p. 17

4.1.6 Suporte a eventos de sistemas e a dispositivos de entrada . p. 18
4.1.7 Suporte ao editor de níveis Tiled . . . . . p. 18
4.2 Projeto do sistema . . . . . p. 19
4.2.1 Linguagem de Programação . . . . . p. 20
4.2.2 Allegro . . . . . p. 22
4.2.3 Suporte ao Tiled . . . . . p. 26
4.3 Implementação da biblioteca . . . . . p. 28
4.3.1 SGL . . . . . p. 29

Referências Bibliográficas p. 35

Lista de Figuras

3.1 *Exemplo de tileset.* . . . . . p. 11

3.2 *Exemplo de um cenário construído com tileset.* . . . . . p. 12

3.3 *Interface do software Tiled.* . . . . . p. 13

4.1 *Interface do software Tiled.* . . . . . p. 19

4.2 Esquema estrutural da SGL. . . . . p. 29

4.3 UML da classe Video. . . . . p. 32

4.4 Display criado com o uso da classe Video. . . . . p. 33

4.5 Uso de vetores na simulação de gravidade. . . . . p. 34

Resumo

Uma biblioteca de jogos ou *game engine* pode ser vista como um conjunto de recursos e ferramentas para a construção de um jogo. Você pode criar um jogo sem uma biblioteca básica, assim como você pode criar uma mesa de madeira sem pregos, martelos, parafusos, chaves de fenda e serras, mas as vantagens que as ferramentas proporcionam justificam chamá-las de necessárias. O nível dessas ferramentas varia: algumas *engines* se limitam a códigos, ou seja, constantes, variáveis, funções e classes relacionadas, mas outras contam com interfaces gráficas que possibilitam o desenvolvimento de um jogo sem codificação alguma. De qualquer forma, uma *game engine* precisa proporcionar, no mínimo, ferramentas para manipular sons, imagens (texto, imagens, etc), memória (dados) e controle (teclado, mouse, etc).

Palavras-chave: *game engine*, jogos eletrônicos, ferramentas de desenvolvimento.

1 Introdução

Desde os primórdios da humanidade a competição é uma forma de diversão muito popular. O objetivo de qualquer competição é testar uma habilidade individual ou grupal e destacar quem ganha. Embora essa característica ainda seja a mesma, a competição está condicionada a uma evolução que pode ser notada, por exemplo, na corrida: no começo ela era individual e só testava a velocidade e a resistência da pessoa; hoje uma corrida automobilística testa a resistência, a destreza, a inteligência e a tecnologia da equipe. Mas essa evolução não se da só na complexidade da competição, ela se manifesta da mesma forma na sua abstração. Os jogos de estratégia que nós conhecemos hoje, por exemplo, são versões abstratas das competições físicas. A aptidão física de cada criatura imaginária é determinada por alguma característica interna do jogo, porque o que o jogo de estratégia testa é o raciocínio e a estratégia da pessoa e não a sua capacidade física propriamente dita. O xadrez internacional, simulando uma guerra da Idade Média, é um caso concreto dessa evolução. Com o avanço da microeletrônica e da computação, no entanto, o jogo de estratégia ganha uma plataforma que pode simular não só um sistema de lógica, mas toda uma realidade virtual. Qualquer jogo pode ganhar uma versão eletrônica. O jogo eletrônico, portanto, não é só uma brincadeira de criança, ele é na verdade o último estágio de um passatempo milenar. Isso se confirma pela movimentação de recursos e ganhos da indústria dos jogos eletrônicos desta geração.

1.1 Motivação

Nos últimos anos, o mercado de games do Brasil tem presenciado um crescimento significativo. [**?**] O advento dos dispositivos móveis, como *smartphones* e *tablets*, e a possibilidade de comercializar seu produto *online* em lojas virtuais e assim reduzir custos favoreceu, em parte, a redução da pirataria e fez com que o usuário preferisse a compra do produto original a investir em um produto não-original. Essa mudança de comportamento por parte do consumidor fez com que um mercado, que antes era visto como inseguro, passasse a ser considerado um mercado promissor pelas empresas desenvolvedoras de software, incluindo as desenvolvedoras de *games*.

Com o crescimento da área de desenvolvimento de jogos eletrônicos, surge também a necessidade de encontrar mão-de-obra capacitada, necessidade esta que é uma das maiores reclamações das indústrias de desenvolvimento de *games* do Brasil. Por se tratar de uma área de desenvolvimento recente, é difícil encontrar profissionais capacitados na área. Uma das soluções mais simples para esta carência de mão-de-obra é incentivar estudantes, sejam eles de nível técnico ou universitário, a aprender sobre as ferramentas e técnicas mais utilizadas no desenvolvimento de um jogo eletrônico. Assim, torna-se de suma importância a implementação de ferramentas que facilitem o primeiro contato do estudante com essa complexa área de desenvolvimento, motivo este que nos motivou a criação da *SAGA Game Library*.

1.2 Objetivos

A *SAGA Game Library* foi desenvolvida tendo em foco o meio acadêmico. Com o aumento do mercado de desenvolvimento de jogos eletrônicos no país, surge a necessidade de investir na capacitação de profissionais para atender a essa demanda. Não apenas profissionais do setor precisam estar em constante atualização, mas os

agora estudantes e futuros profissionais também precisam de capacitação. É para esse último que é direcionada esta biblioteca de desenvolvimento. Seu objetivo primário é possibilitar ao usuário, seja ele um estudante ou entusiasta, o primeiro contato com o mundo do desenvolvimento de jogos.

É certo que já existem muitas *game engines*, inclusive em C++, mas o estudo é o piso de todas as descobertas científicas, o que justifica e motiva o desenvolvimento de uma biblioteca de jogos didática. Esta é a nossa proposta: uma camada de orientação a objetos envolvendo a Allegro de uma forma simples e didática. Simplicidade, eficiência e aprendizado são as palavras-chave da *SAGA Game Library*.

2 Motivação

Nos últimos anos, o mercado de games do Brasil tem presenciado um crescimento significativo. [?] O advento dos dispositivos móveis, como *smartphones* e *tablets*, e a possibilidade de comercializar seu produto *online* em lojas virtuais e assim reduzir custos favoreceu, em parte, a redução da pirataria e fez com que o usuário preferisse a compra do produto original a investir em um produto não-original. Essa mudança de comportamento por parte do consumidor fez com que um mercado, que antes era visto como inseguro, passasse a ser considerado um mercado promissor pelas empresas desenvolvedoras de software, incluindo as desenvolvedoras de *games*.

Com o crescimento da área de desenvolvimento de jogos eletrônicos, surge também a necessidade de encontrar mão-de-obra capacitada, necessidade esta que é uma das maiores reclamações das indústrias de desenvolvimento de *games* do Brasil. Por se tratar de uma área de desenvolvimento recente, é difícil encontrar profissionais capacitados na área. Uma das soluções mais simples para esta carência de mão-de-obra é incentivar estudantes, sejam eles de nível técnico ou universitário, a aprender sobre as ferramentas e técnicas mais utilizadas no desenvolvimento de um jogo eletrônico. Assim, torna-se de suma importância a implementação de ferramentas que facilitem o primeiro contato do estudante com essa complexa área de desenvolvimento, motivo este que nos motivou a criação da *SAGA Game Library*.

3 Metodologia

3.1 O software Tiled

A grande maioria dos jogos eletrônicos 2D apresenta, além dos *sprites* dos personagens e itens, uma imagem representando o cenário do jogo. Dependendo da natureza do jogo, esses cenários podem possuir grandes dimensões, o que torna custoso para o software do jogo carregar e armazenar essa imagem em memória (RAM e em disco). Mas, se analisarmos a imagem que representa o cenário, vamos verificar que o mesmo é formado por pequenas partes que se repetem com muita frequência. Assim, aproveitando dessa característica e com o objetivo de diminuir o consumo de memória e o desempenho ao carregar imagens de resolução elevada, foi desenvolvida uma técnica conhecida como *TileMap*. A técnica consiste no uso de uma imagem, chamada *tileset*, contendo pequenos pedaços de imagens, conhecidos como *tiles*, que são as imagens que se repetem em grande quantidade na imagem do cenário de jogo. Estes *tiles* são usados para criar uma imagem composta denominada *tiled layer*.

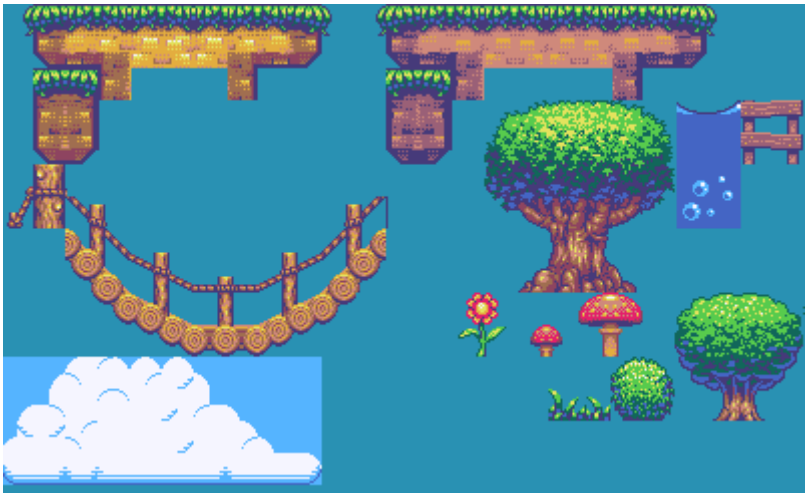

Figura 3.1: *Exemplo de tileset.*

O cenário final do jogo pode ser constituído de um único *tiled layer* ou ser resultante da combinação de dois ou mais *tiled layers*. Através da técnica de *Tilemap*, torna-se possível construir inúmeros cenários, com variadas dimensões, usando como base o mesmo *tilesets*, aumentando a economia de memória e não reduzindo o desempenho no carregamento de imagens.

Figura 3.2: *Exemplo de um cenário construído com tileset.*

O software Tiled [http://www.mapeditor.org/ 2013] é uma ferramenta gratuita desenvolvida em C++ para a criação de layouts e mapas usando *tilesets* baseado na técnica de *Tilemap*. De simples manuseio e grande versatilidade, o Tiled faz a edição de várias camadas de *tiles* e salva tudo em um formato padronizado de extensão ".tmx". Uma das principais vantagens do formato TMX é sua organização, detalhamento e praticidade, sendo que seu conteúdo pode ser lido através do uso de um *parser* para arquivos XML.

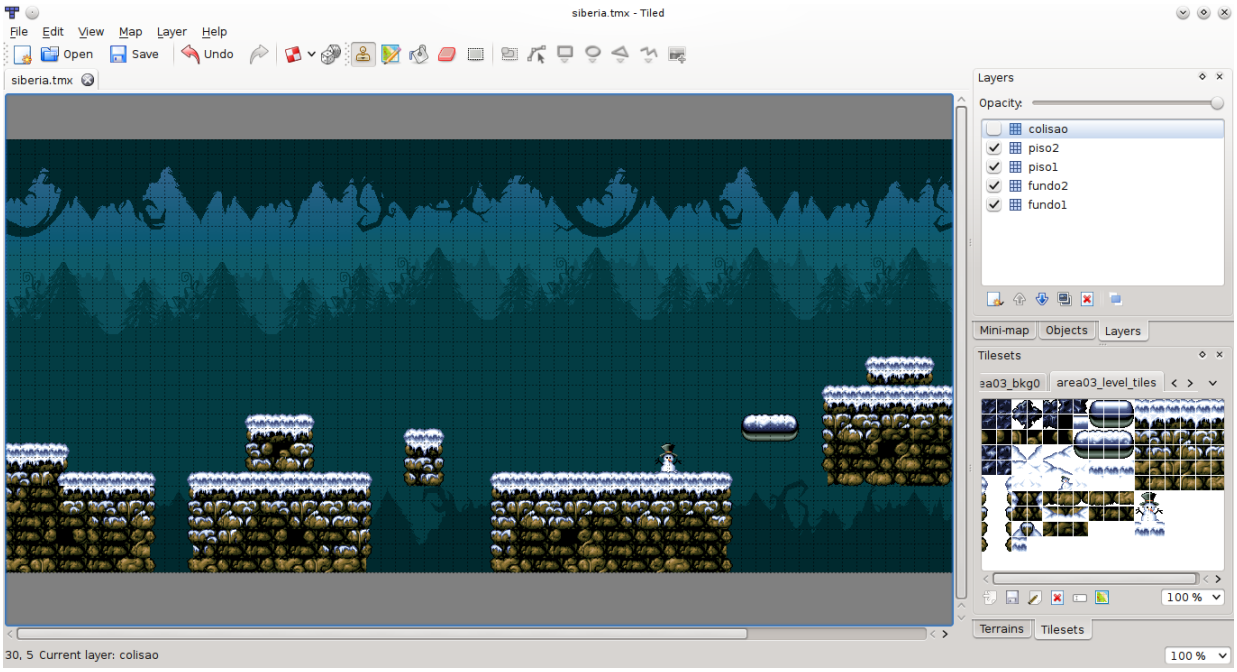

Figura 3.3: *Interface do software Tiled.*

O Tiled é um editor de níveis que suporta mapas com projeções ortogonais e isométricas e ainda permite que objetos personalizados sejam salvos como imagens na resolução que desejar. Tem suporte também a comandos externos, *plugins* e formatos usados por outros editores. É compatível com diversas *engines* de criação de jogos e fornece meios de comprimir seus dados de modo a diminuir o tamanho em disco do arquivo TMX. É possível, ainda, redimensionar e alterar o mapa posteriormente, criar múltiplos mapas em uma única sessão e ainda salvar ou restaurar até nove vezes. Com ele pode-se especificar o tamanho de cada *tile* em um *tileset*, ou criar um mapa sem tamanho estrito sobre as imagens [Brunner 2012]. Mesmo que o desenvolvedor não queira que seu jogo seja baseado em *tiles*, o software é ainda uma excelente escolha como um editor de níveis. Pode-se usá-lo também nas entidades invisíveis, tais como áreas de colisão e o aparecimento de objetos dentro do mapa. Por sua simplicidade ela pode ser usada por programadores iniciantes ou experientes.

O processo de criação de um mapa com o Tiled é feito basicamente usando os passos abaixo:

1. Escolher o tamanho do mapa e tamanho do *tile* base;
2. Adicionar *tilesets* vindos de imagens;
3. Adicionar quaisquer objetos que representem algo abstrato;
4. Salvar o mapa no format TMX;
5. Importar o arquivo TMX e interpretá-lo para o jogo.

O Tiled é totalmente gratuito. Esse detalhe aliado à sua facilidade de uso e versatilidade, tornaram-no extremamente popular em meio à comunidade de desenvolvedores de jogos, onde não só estudantes e entusiastas, mas também empresas e profissionais da área, passaram a adotá-lo como editor de níveis padrão. Seguindo esse pensamento, o nosso *framework* a ser desenvolvido também possui suporte nativo aos níveis construídos usando esse software.

4 Desenvolvimento

O desenvolvimento da *SAGA Game Library* ou SGL, assim como todo *software*, passou por diversas etapas para que no final torna-se possível obter um produto condizente com a proposta do trabalho.

O desenvolvimento de um *software*, de maneira geral, sempre é composto das seguintes etapas:

- Especificação dos requisitos do *software*: Descrição do objetivo e do se espera do *software*.
- Projeto do sistema: decisão dos conceitos relacionado ao que deve ser implementado, incluindo a escolha da linguagem de programação adequada, sistema operacional alvo, bibliotecas e ferramentas auxiliares.
- Implementação: O próprio desenvolvimento do *software*. Consiste na transformação de todo conteúdo formulado na fase de projeto em código.
- Teste e depuração: Fase que consiste no teste do *software* já implementado e procura por erros e correção destes.
- Documentação: A fase final do desenvolvimento consiste em documentar o *software* criado, incluindo manuais de uso da biblioteca.

A SGL também seguiu de maneira consistente as etapas acima. A seguir serão descritas as particularidades de cada uma delas.

4.1 Especificação dos requisitos do projeto

O objetivo inicial deste trabalho de conclusão de curso consistia na implementação de um *game* online que utilizasse toda a base teórica vista durante o curso de graduação. Entretanto após analise do conteúdo e recursos necessários para realizar tal projeto, conclui-se que o mesmo demandaria um custo alto, tanto em relação a mão-de-obra quanto ao conhecimento necessário para fazê-lo. Com base nessa conclusão, seguimos em busca de outros temas de projeto ainda relacionados a área de jogos eletrônicos. Após alguns estudos sobre o mercado de desenvolvimento de jogos, chegamos ao consenso de desenvolver uma biblioteca de desenvolvimento de jogos voltada para o meio acadêmico.

A escolha do público alvo da biblioteca teve grande influência nos requisitos exigidos no desenvolvimento desse *software*. Uma vez que a biblioteca deveria ser voltada ao meio acadêmico, ou seja, ao ensino, ela deveria apresentar uma baixa curva de aprendizado sem dispensar recursos importantes para uma *game engine*. A seguir, estão listados os requisitos que a biblioteca deverá apresentar ao final de sua implementação.

4.1.1 Baixa curva de aprendizado

como dito anteriormente, a biblioteca é voltada para estudantes e entusiastas que possuem pouca ou mesmo nenhuma experiência na área de programação de jogos eletrônicos. Assim, torna-se essencial que a *engine* seja de fácil uso, de modo que o usuário sinta-se a vontade e amparado para desenvolver seus projetos e não desestimulado por utilizar uma ferramenta que apresenta um modo intuitivo de uso.

4.1.2 Gerenciamento automático de recursos

em desenvolvimento de jogos, os chamados recursos ou *resources*, são arquivos de imagens, fontes de texto *True Type* e arquivos de áudio. O gerenciamento de recursos possibilita considerável redução no uso da memória RAM destinada aos recursos e no tempo de carregamento dos mesmos.

4.1.3 Arquitetura multi-plataforma

Um dos requisitos fundamentais da biblioteca. Um dos principais objetivos da SGL é ser compatível pelo maior números de plataformas possíveis de modo que o usuário sinta-se a vontade para desenvolver em sua plataforma preferida.

4.1.4 Suporte a diversos formatos de arquivos

Suporte a arquivos de imagens (.PNG, .JPG e .BMP), de font *True Type* e reprodução de arquivos de áudio (.ogg e .wav). São os tipos de arquivos mais comuns utilizados em jogos. As fontes *True Type* são preferidas ao invés da fontes *bitmap*, por estas últimas não suportarem *anti-alising*.

4.1.5 Renderização acelerada por *hardware* e gráficos 2D

Um pré-requisito para todas *game engine* é fazer uso dos *hardwares* mais atuais. Algumas API's como a popular SDL, possuem suporte apenas a renderização por software para gráficos 2D, o que aumentava consideravelmente o uso do processador durante as rotinas de desenho do jogo. O único modo de obter a aceleração por *hardware*, usando diretamente a placa de vídeo, era utilizar a OpenGL para realizar as rotinas de renderização. Esse hibridismo entre SDL e OpenGL aumenta em muito a complexidade de implementação de um jogo onde a aceleração por *hardware* se faz necessária. Com isso em mente, a *SAGA Game Library* deveria fornecer essa aceleração por *hardware* para rotinas de renderização 2D de um

modo que o desenvolvedor que estivesse utilizando a biblioteca não precisasse de preocupar em usar outras API's para este fim.

4.1.6 Suporte a eventos de sistemas e a dispositivos de entrada

A biblioteca deveria ser capaz de receber eventos de teclado, *mouse* e *joystick*. Suporte a dispositivos com tela *tounch* era uma opção secundária.

4.1.7 Suporte ao editor de níveis Tiled

A edição de níveis ou cenários são uma das tarefas mais complexa no desenvolvimento de um jogo. Todo o processo desde a elaboração artística do cenário até a maneira com o jogador irá interagir com o mesmo, exigem muito planejamento e demanda muito esforço em sua implementação. Com o objetivo de facilitar esse trabalho, surgiram os editores de níveis.

Os editores de níveis são *softwares* destinados a construção de cenários em jogos, o que faz deles ferramentas de grande importância para uma *game engine*. Neste ponto, existem duas opções disponíveis para que deseja implementar uma *engine*. Desenvolver seu próprio editor ou utilizar um editor externo, desenvolvido por terceiros. Desenvolver um editor próprio demanda muito estudo e tempo de desenvolvimento além do que já é necessário para implementar a *engine*. Tendo isso em mente, chegou-se a conclusão que o melhor seria oferecer suporte a um editor externo, no qual o editor Tiled foi escolhido.

O software Tiled [1] é uma ferramenta gratuita desenvolvida em C++ para a criação de layouts e mapas usando *tilesets* baseado na técnica de *Tilemap*. Ele suporta mapas com projeções ortogonais e isométricas e ainda permite que objetos personalizados sejam salvos como imagens na resolução que desejar. Tem suporte também a comandos externos, *plugins* e formatos usados por outros editores. É

[1] Tiled: http://www.mapeditor.org/

possível, ainda, redimensionar e alterar o mapa posteriormente, criar múltiplos mapas em uma única sessão e ainda salvar ou restaurar até nove vezes. Com ele pode-se especificar o tamanho de cada *tile* em um *tileset*, ou criar um mapa sem tamanho estrito sobre as imagens [Brunner 2012].

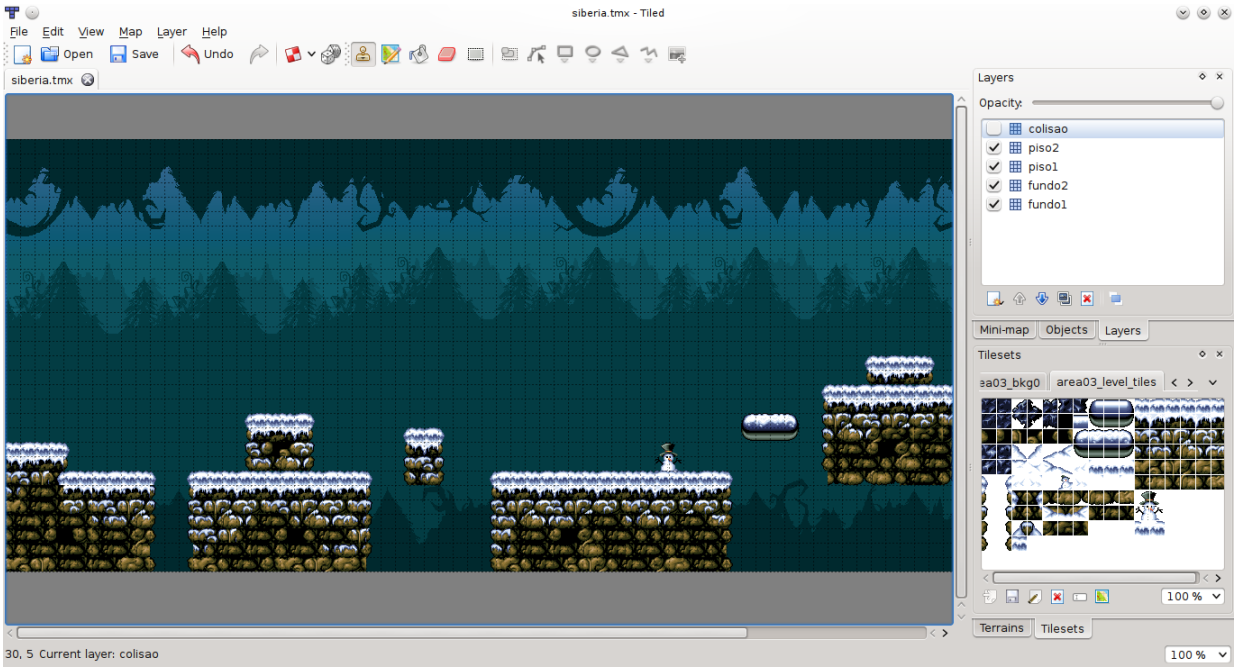

Figura 4.1: *Interface do software Tiled.*

Mesmo que o desenvolvedor não queira que seu jogo seja baseado em *tiles*, o software é ainda uma excelente escolha como um editor de níveis. Pode-se usá-lo também nas entidades invisíveis, tais como áreas de colisão e o aparecimento de objetos dentro do mapa. Por sua simplicidade ela pode ser usada por programadores iniciantes ou experientes.

4.2 Projeto do sistema

Após definido os requisitos do sistema, a próxima etapa foi dar início ao desenvolvimento do projeto. O primeiro passo foi definir, com base nos requisitos e

no conhecimento dos envolvidos qual seria a linguagem de programação mais adequada ao problema e qual API deveríamos usar para as rotinas de renderização, leitura dos dispositivos de entrada, reprodução de áudio e etc.

4.2.1 Linguagem de Programação

Até a década de 90 cada jogo tinha a sua *engine*, feita para possibilitar a maior eficiência no uso da memória e da unidade de processamento possível, de acordo com as exigências de cada jogo. Um jogo que só usava formas geométricas, por exemplo, não precisava tratar imagens na sua *engine*. O nível da microeletrônica e da computação já possibilita o uso de *engines* genéricas, mas o desenvolvimento de uma ainda demanda uma programação muito próxima da máquina.

É por isso que a escolha da linguagem de programação precisa ser feita com cuidado. Considerando o conhecimento da equipe e o propósito do projeto, que era uma *engine* didática, para influenciar o designe e o desenvolvimento de jogos de acordo com o nosso alcance, as linguagens de programação selecionadas para a análise foram Actionscript, C++, C# e Java, de acordo com a simplicidade, o poder e a compatibilidade de cada uma.

Estudo Comparativo

O Actionscript é uma linguagem orientada a objetos desenvolvida pela Macromedia. O que no início era uma ferramenta para controlar animações se tornou uma linguagem de script tão complexa que podia ser usada no desenvolvimento de um jogo. Embora essa linguagem ainda seja muito usada no desenvolvimento de jogos de web, o que provocou a decisão contrária a ela foi a expectativa de que o HTML 5 viesse a incorporar o Javascript e, dessa forma, modificar ou inutilizar o Actionscript.

O C# (C Sharp) é uma linguagem multi-paradigma da Microsoft feita para o desenvolvimento de sistemas próprios para a plataforma .NET. C# e Java com-

partilham a mesma simplicidade na leitura e na codificação, assim como a mesma forma de interpretação e compilação, mas um programa em C# está mais próximo da máquina do que um programa em Java. A escolha parecia feita quando nós entendemos que a ligação do C# com a Microsoft poderia custar a compatibilidade do nosso jogo.

Java é uma linguagem orientada a objetos desenvolvida pela Sun, hoje possuída pela Oracle. A sua fama de espaçosa e pesada não é coerente com a realidade: hoje a linguagem conta com a Compilação na Hora ou *Just in Time Compilation (JIT Compilation ou só JIT)* , para que a sua execução não seja mais interpretada. Mas a sua principal característica é a compatibilidade: a Máquina Virtual do Java ou Java Virtual Machine (JVM) é uma plataforma virtual que pode ser feita compatível para qualquer plataforma física.

Por mais evoluída que seja a JVM, no entanto, o Java não admite o acesso à máquina necessário para o desenvolvimento de uma *game engine*, senão com o uso do C++, por meio da Interface Nativa do Java ou Java Native Interface (JNI). Em outras palavras, para usar o Java, nesse caso, nós teríamos que usar o C++. Esta, por sua vez, não é a mais simples na codificação, mas não tem limitação alguma tanto em termos de compatibilidade quanto em termos de acesso à máquina.

A Escolha: C++

C++ (C Mais Mais ou C Plus Plus) [Perucia 2007] é uma linguagem de programação multi-paradigma, com suporte para a programação imperativa e a programação orientada a objetos, de uso geral, desenvolvida por Bjarne Stroustrup, para formar uma camada de orientação a objetos sobre a linguagem de programação C. O C++ possibilita a programação de baixo nível assim como a programação de alto nível e por isso é considerado uma linguagem de programação de nível médio em termos de proximidade da máquina.

Após o estudo comparativo acima, concluímos que a linguagem C++ seria a

melhor opção para o projeto em questão. Tal decisão foi baseada em diversos fatores como experiência e domínio da linguagem pelos envolvidos no projeto, o desempenho das aplicações escritas nesta linguagem e o fato de C++ ser a linguagem de programação mais usada em desenvolvimento de jogos. Uma vez que o objetivo da biblioteca é ser uma porta de entrada para essa área, a escolha da linguagem faz com que o usuário já se familiarize com a mesma. Por último e não menos importante, o C++ é uma linguagem 100% compatível com a API escolhida como base para a nossa *engine* que será abordada a seguir.

4.2.2 Allegro

Nas nossas pesquisas para escolher uma biblioteca com a qual trabalhar, duas se destacaram: a Allegro e a SDL. A SDL (Simple Direct Media Layer - Camada de Mídia Direta Simples) é uma popular biblioteca multimídia simples de usar, multiplataforma, de código aberto, e amplamente usada para fazer jogos e aplicações multimídia. Ela poderia também atender as nossas necessidades, mas a Allegro se destacou por ter um código mais limpo e intuitivo, e rotinas específicas para o desenvolvimento de jogos, como renderização acelerada por hardware e suporte nativos a diversos formatos de imagens e arquivos de áudio. Por esta razão ela foi escolhida.

Allegro [2] é uma biblioteca gráfica multiplataforma, de código fonte aberto e feita na sua maioria em C, mas utilizando internamente também Assembly e C++. Seu nome é um acrônimo recursivo que representa "*Allegro Low Level Game Routines*" ("Rotinas de jogo de baixo nível Allegro"). Funciona em diversos compiladores e possui rotinas para a manipulação de funções multimídia de um computador, além de oferecer um ambiente ideal para o desenvolvimento de jogos, tornando-se uma das mais populares ferramentas para esse fim atualmente. Originalmente desenvolvida por Shawn Hargreaves, ela se tornou um projeto colaborativo, com colaboradores de todo o mundo.

[2] Allegro: http://alleg.sourceforge.net/readme.html

Ela possui suporte nativo para rotinas de que utilizam gráficos 2D, embora seja possível utilizá-la em conjuntos com outras API's, como OpenGL e DirectX, para desenvolvimento de aplicações que utilizem gráficos 3D. Apesar de não ser suficiente para o completo desenvolvimento de um jogo, existem pequenas bibliotecas adicionais (add-ons), feitas para serem acopladas à Allegro, permitindo assim a sua extensão. Através desses add-ons é possível, por exemplo, obter suporte a arquivos MP3, GIF, imagens JPG e vídeos AVI. Isso é útil para que o usuário não tenha que colocar uma porção de funções que não usa na hora de distribuir seu jogo, incluindo somente as partes que for utilizar, diminuindo consideravelmente o tamanho do mesmo.

Atualmente a biblioteca se encontra na sua quinta versão. Allegro 5 foi completamente reescrita e não apresenta compatibilidade com as suas versões anteriores. Foi feito um esforço para tornar a API mais consistente e segura, o que trouxe melhorias funcionais e uma grande mudança na sua arquitetura, sendo agora orientada a eventos e possuindo suporte nativo a aceleração por hardware. Possui a suporte a eventos gerados por dispositivos de entradas como teclado, *mouse* e *joystick* além de funções para desenho de primitivas gráficas, leitura e gravação de seu próprio tipo de arquivo de configuração (muito útil para armazenar configurações e dados de jogos) e é totalmente modular.

A Allegro 5.0 suporta as seguintes plataformas:

- Windows (MSVC, MinGW);
- Unix/Linux;
- MacOS X;
- iPhone;
- Android (Suporte provido pela Allegro 5.1, que ainda se encontra instável).

A API atualmente se encontra nas versões 5.0.10 (estável) e 5.1.2 (instável). A

SAGA Game Library foi desenvolvida usando como base a versão 5.0.10, por a mesma ser estável.

Principais Recursos e Funções

A seguir, encontramos um conjunto dos principais recursos da Allegro 5, começando com as funções mais gerais da biblioteca.

- **al_init()**: Inicializa a biblioteca Allegro, dando valores a algumas variáveis globais e reservando memória. Deve ser a primeira função a ser chamada.
- **al_exit()**: Encerra a Allegro. Isto inclui retornar ao modo texto e remover qualquer rotina que tenha sido instalada. Não há necessidade de chamar essa função explicitamente, pois, normalmente, isto é feito quando o programa termina.

As rotinas de vídeo:

- **ALLEGRO_DISPLAY**: Tipo que representa a janela principal. A biblioteca permite que se trabalhe com múltiplas janelas.
- **al_create_display(width, height)**: Cria uma instância da janela, retornando um ponteiro para ALLEGRO_DISPLAY. Os parâmetros indicam as dimensões em pixels.
- **al_flip_display()**: Função para atualizar a tela.
- **al_destroy_display(var)**: Finaliza a instância *var* do tipo ALLEGRO_DISPLAY ×
.

As rotinas para manipulação de arquivos de imagem:

- **ALLEGRO_BITMAP**: Tipo que representa o arquivo de imagem carregado pela Allegro.

- **al_init_image_addon()**: Inicializa o add-on da Allegro 5 para utilização de imagens.

- **al_load_bitmap("example.jpg")**: Carrega a imagem indicando no parâmetro o nome e tipo. Ela deve estar previamente salva na pasta do programa. Recebe o caminho relativo ou absoluto da imagem a ser carregada, retornando um ponteiro para o tipo ALLEGRO_BITMAP.

- **al_draw_bitmap(bitmap, x, y, mirror)**: Função para desenhar a imagem na tela. Os parâmetros são o bitmap a ser desenhado, as posições x e y e as flags de espelhamento (0, ALLEGRO_FLIP_HORIZONTAL, ALLEGRO_FLIP_VERTICAL).

As rotinas de áudio:

- **ALLEGRO_SAMPLE**: Tipo que representa arquivos pequenos, geralmente efeitos sonoros.

- **ALLEGRO_AUDIO_STREAM**: Tipo para representar arquivos grandes, de forma que o arquivo não é carregado de uma vez para a memória. Geralmente representa os arquivos que irão compor uma trilha sonora.

- **al_install_audio()** e **al_init_acodec_addon()**: A primeira inicializa as funções relativas ao áudio. A segunda inicializa os codecs necessários para carregar os diversos formatos de arquivo suportados. Fornece suporte a alguns formatos, como Ogg, Flac e Wave.

- **al_set_audio_stream_playing(musica, true)**: Função que recebe o arquivo de áudio já carregado no primeiro parâmetro e um tipo booleano no segundo (true para fazê-la tocar ou false, em caso contrário).

- **al_destroy_audio_stream(musica)** e **al_destroy_sample(sample)**: Funções de desalocação dos arquivo de áudio carregados pela Allegro.

A Allegro ainda possui muitos recursos que não foram citados devido a sua quantidade. Posteriormente, a API será explorada mais a fundo conforme os recursos da *SAGA Game Library* forem sendo mostrados.

4.2.3 Suporte ao Tiled

O Tiled faz a edição de várias camadas de *tiles* e salva tudo em um formato padronizado de extensão ".tmx". Uma das principais vantagens do formato TMX é sua organização, detalhamento e praticidade, sendo que seu conteúdo pode ser lido através do uso de um *parser* para arquivos XML.

Existem inúmeros *parsers* de arquivos XML, cada um com suas vantagens e desvantagens. Entre os *parser* mais populares podemos citar a RapidXML e a TinyXML. Segundos comparação do próprio desenvolvedor, a RapidXML apresenta a maior velocidade da leitura de dados entre os *parsers* as mais populares mas o escolhido foi a TinyXML, por sua simplicidade de uso e velocidade que não apresenta uma diferença significativa a sua concorrente RapiXML.

A biblioteca TinyXML

É uma biblioteca escrita em C++ que analisa uma sequência de entrada no formato XML permitindo o acesso aos dados contidos nesse último. Em outras palavras, ela realiza o *parser* de uma arquivo .xml e armazena a informação em objetos C++ que podem ser manipulados livremente.

A TinyXML [3] pode ser facilmente integrada em outros programas, bastando apenas adicionar seus arquivos ao projeto. Com ela é possível realizar o acesso aos dados direta ou iterativamente, alteração da estrutura através de inserção e remoção de elementos, remoção de espaços duplicados e a gravação para ficheiros em formato XML. TinyXML é uma estrutura extremamente compacta e robusta, elaborada para um rápido e fácil aprendizado. Pode ser usada para fins de código

[3] http://www.grinninglizard.com/tinyxml/

aberto ou comerciais. Ela é compatível com UTF-8, de modo a permitir que arquivos XML sejam manipulados em qualquer linguagem. Também apresenta grande simplicidade em seu uso e rapidez na leitura de dados.

Tiled: Codificação e compactação de dados

Um dos recursos mais interessantes do Tiled é a possibilita de exportar os dados contidos no arquivo .tmx de forma compactada e codificação. A principal vantagem do uso dos algoritmos compactação e codificação é a redução do tamanho em disco do arquivo .tmx resultante e aumento da velocidade de carregamento do cenário, uma vez que a biblioteca carrega os dados codificados e/ou compactados e os decodifica/descompacta a nível de software ao invés de realizar a leitura dos mesmo em disco.

A compactação de dados fornecida pelo Tiled é realizada pelas bibliotecas ZLIB [4] e GZIP [5], enquanto a codificação é realizada pelo algoritmo Base64 fornecido por [6], também fornecido por uma biblioteca externa. São fornecidas as opções de exportar os dados no arquivo .tmx na forma codificada e compactada ou apenas na forma codificada. Também são oferecidas as opções de exportar os dados em formato puro (sem codificação) XML ou no formato CVS.

A seguir podemos verificar a redução do tamanho em disco de um arquivo .tmx em relação ao seu tamanho em disco sem codificação/compactação.

- Arquivo XML puro(sem codificação/compactação): 31,3 KB;
- Com codificação Base64: 9,6 KB;
- Com codificação Base64 e compressão GZIP: 2,2 KB;
- Com codificação Base64 e compressão ZLIB: 2,1 KB;

[4] ZLIB: http://www.zlib.net/
[5] GZIP: http://www.gnu.org/software/gzip/gzip.html
[6] Algoritmo Base64: http://www.adp-gmbh.ch/cpp/common/base64.html

- No formato CVS: 4,9 KB.

A *SAGA Game Library* prove suporte total e otimizado as 5 opções de exportação acima. Ela, assim com o Tiled, também faz uso das bibliotecas GZIP e ZLIB para descompactação e também realiza a decodificação dos dados em Base64 contidos no arquivo .tmx (detalhes de uso serão abordados posteriormente).

4.3 Implementação da biblioteca

A *SAGA Game Library* é uma *engine* orientada a objeto desenvolvida em C++ sobre a API Allegro 5. Ela possui todos os recursos que uma *engine* básica possui, como suporte a criação de *sprites* animados e estáticos, criação de cenários usando o editor de níveis Tiled, suporte ao uso de fontes de texto TTF e a diversos formatos de arquivos de áudio. Também possui gerenciamento de automático de recursos (imagem, fontes e áudio), o que possibilita economia significativa de memória. A biblioteca fornece uma estrutura para implementação do *game loop* e suporte para eventos de teclado, *mouse* e *joystick* também estão presentes.

Após a fase de projeto, deu-se sequência a fase de estudo do problema. Está fase resumiu-se simplesmente no estudo do problema a fim de torná-lo orientado a objetos. Internamente, a *SAGA Game Library* consiste em um *package* denominado *sgl* e subdivididas em 4 outros *packages* com funções específicas: *image, font, audio e input.*

- image: consiste nas classe exclusivamente relacionadas a parte gráfica da *engine*, o que inclui manipulação de arquivos de imagens como *sprites* animados e estáticos e carregamento e renderização de cenários do jogo;

- font: é o responsável pela parte textual da biblioteca. Ele carrega uma fonte no formato TTF (“True Type Font”) padrão do Windows e a utiliza para escrever textos na tela de jogo;

- audio: *package* destinado ao carregamento, manuseio e reprodução dos arquivos de áudio;
- input: incluí todas as funções relacionadas aos comandos recebidos pelos dispositivos de entradas do computador (teclado, *mouse* e *joystick*).

Abaixo podemos visualizar o esquema da organização da *engine*. A seguir, serão abordados os principais recursos da biblioteca bem como os detalhes da suas respectivas implementações.

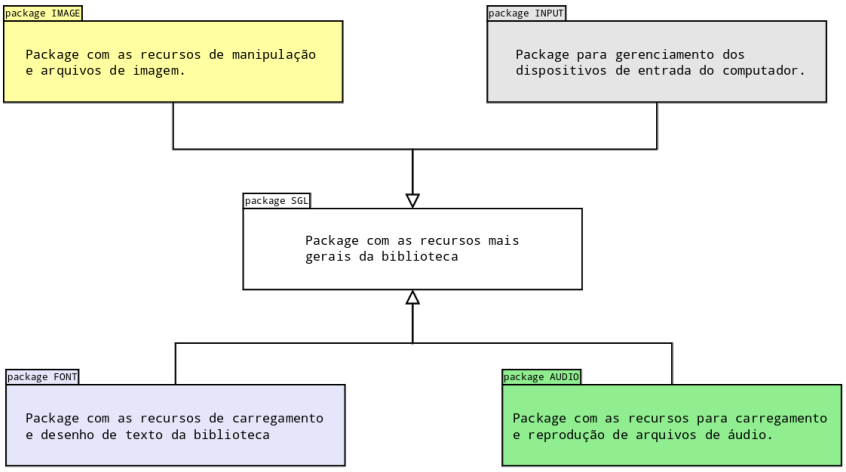

Figura 4.2: Esquema estrutural da SGL.

4.3.1 SGL

É o pacote mais geral, e que engloba todos os outros. Quando a funcionalidade de uma classe não é específica ou é usada como ferramenta auxiliar em outras classes, ela é colocada nesse pacote.

AllegroStarter

Cada um dos módulos da Allegro deve ser inicializado antes de seu uso. A Classe AllegroStarter é responsável e também por desalocar os recursos quando

o programa é fechado. Uma exceção é lançada caso algum dispositivo apresente problemas durante a inicialização. Também contém informações sobre a atual versão da Allegro.

SGLException

É a classe que captura e trata as exceções que possam ocorrer durante a execução do programa, como inicialização dos componentes da allegro e carregamento de arquivos de imagem, fonte e áudio. A classe é uma especialização de std::exception. Abaixo temos um exemplo de uso desta classe.

```
1
2 if( !al_init() ) {
3 throw sgl::Exception( "Failed to initialize ALLEGRO_Lib." );
4 }
```

CodigoFonte/ExemploSGLException.cpp

Caso ocorra algum erro de inicialização da Allegro, a função *al_init()* retornará *false*, fazendo com que o programa execute o conteúdo do *if* e lance a exceção. No console, teremos a mensagem: "*terminate called after throwing an instance of 'sgl::Exception' what(): Failed to initialize ALLEGRO_Lib.*".

Color

A classe Color possui recursos para trabalharmos com diversos formatos de especificação de cores, como RGB e HTML. Os objetos dessa classe podem ser usadas, por exemplo, para colorir a tela ou alterar a cor de uma determinada fonte de texto. Ela aceita dois construtores. Com o primeiro deles é possível definir cores no formato RGB. Para isso, o construtor recebe três parâmetros que variam de 0 a 255, um para vermelho, outro para verde e outro para azul, respectivamente. O segundo construtor aceita strings no formato html ou o nome em inglês de uma cor, desde que ele já esteja pré-definido. Alguns dos nomes válidos são: *cyan*,

lightgreen, *green* entre outros. Caso o nome de uma cor inexistente seja passado como parâmentro, o contrutor cria um objeto com as coordenadas RGB iguais a 0 (cor preta). A seguir temos alguns exemplos de uso da classe Color.

```
1  // Defini  o das cores vermelho e azul escuro usando o primeiro
       construtor.
2
3       Color vermelho(255,0,0);
4       Color azulEscuro(0,0,139);
5
6  // Defini  o das cores rosa passando um nome pre-definido e das
       cores verde escuro e coral no formato html.
7  // Note que o simbolo # e opcional.
8
9       Color rosa("pink");
10      Color verdeEsc("#006400");
11      Color coral("FF7F50");
12
13 // Convertendo formatos.
14
15   std::string cor;
16   cor = vermelho.getName();    // cor = red
17   cor = azulEsc.toHTML();      //  cor = #00008b
```

CodigoFonte/ExemploColor.cpp

Video

Classe responsável por gerenciar todos os recursos de vídeo da *engine*. Através dela, tem-se acesso a todas as rotinas pertinentes (de atualização de tela, posicionamento, e outros eventos) para o gerenciamento de vídeo. A classe também possibilita a escolha por parte do usuário de criar um *display* no formato de janela ou em *fullscreen*, e também permite a escolha de uma API para renderização 3D, como OpenGL (Linux e Windows) e DirectX (Windows).

Quando criarmos um objeto da classe Video, devemos definir as dimensões do

display, dado pelas coordenadas x e y conforme o desenho e o modo (WINDOWED, FULLSCREEN) no construtor da classe. Por padrão, a cor de fundo do vídeo é preta, podendo ser alterada. Também existe a possibilidade de adicionar um ícone e um título para nossa janela. A função *refresh()* é responsável por por transferir o conteúdo da memória de vídeo para o *display* e deve ser chamada sempre que seja necessário atualizar o *display*, caso contrário, nenhum das ações realizadas pelas rotinas de desenho serão visualizadas.

A principal dificuldade na implementação da classe foi aprender como a Allegro fazia uso da memória de vídeo e dos conceitos envolvidos como taxa de atualização do *display*, uso do *vsync* e de como poderíamos tirar o máximo proveito disso. O principal fator foi quanto ao desempenho da renderização da tela. Isso foi resolvido ajustando a Allegro para que a mesma armazena-se todos as imagens a serem desenhadas na memória de vídeo ao invés da memória RAM. Essa configuração apresentou resultados significativos no uso do processador. Enquanto as imagens armazenadas na RAM utilizavam 25% da CPU, as mesmas imagens armazenadas na memória de vídeo não consumiam mais de 2% do processador durante o processo de desenho no *display*.

A seguir temos a classe Video em detalhes.

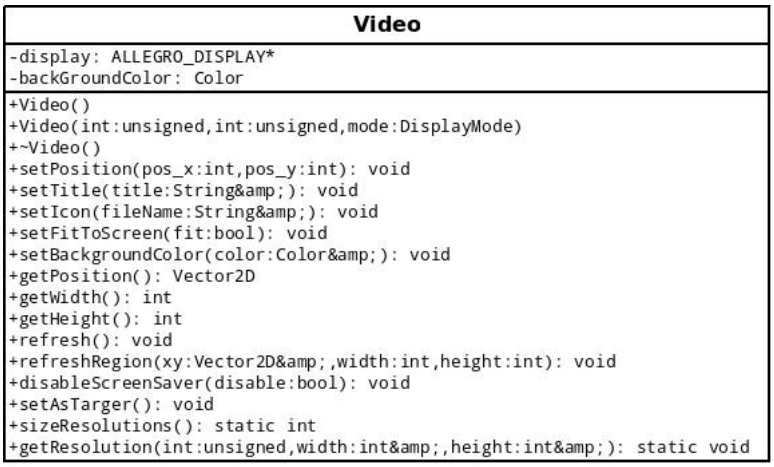

Figura 4.3: UML da classe Video.

e um exemplo de uso da mesma.

```
1
2  // Criamos uma instancia de video
3  Video video( 300, 100, DisplayMode::WINDOWED );
4
5  // Definimos o icone da janela do display
6  video.setIcon("nice.jpg");
7
8  // Mudanca da cor de fundo da tela.
9  video.setBackgroundColor( Color("aqua") );
10
11 // Definindo um titulo para a janela.
12 video.setTitle( "Saga Game Library" );
13
14 // Atualizando o display.
15 video.refresh();
```

CodigoFonte/ExemploDisplay.cpp

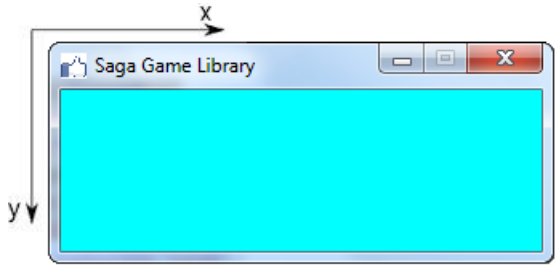

Figura 4.4: Display criado com o uso da classe Video.

Vector2D

A classe Vector2D é uma das classe mais importantes da *SAGA Game LIbrary*. A classe Vector2D tal qual o nome diz, define um vetor bidimensional, iniciando-se no ponto (0,0) da tela e indo até o ponto (x,y) definido pelo usuário, sendo responsável pelo posicionamento de qualquer entidade, seja ela um *sprite* ou um

texto, no *display*. A escolha do uso de vetores ao invés de simples coordenadas deve-se a vantagens que o uso de vetores oferece. Muitas operações matemáticas como deslocamento, rotações e escalonamentos são feitas utilizando vetores. Até mesmo efeitos de física, como gravidade e aceleração são executados usando como base os conceitos de mecânica clássica. A vantagem do vetor sobre um par de coordenadas (x,y) deve-se a quantidade de informação que o primeiro carrega. Enquanto um ponto representa apenas uma localização no espaço, um vetor possui uma direção, sentido e magnitude.

Assim como ocorre na geometria, os vetores também possibilitam diversas operações algébricas entre eles, através da sobrecarga de operadores. Soma e subtração de vetores, normalização são algumas das operações possíveis e que podem ser usadas para se obter interessantes resultados, como uma simulação de gravidade ilustrada na figura abaixo, onde temos um vetor representando o deslocamento do personagem e outro representando a força gravitacional.

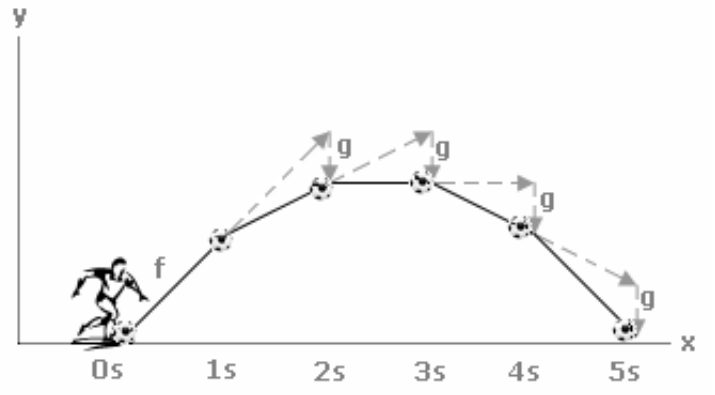

Figura 4.5: Uso de vetores na simulação de gravidade.

Referências Bibliográficas

BRUNNER, N. Introduction to tiled map editor: A great, platform-agnostic tool for making level maps. 2012.

Editor de mapas: Tiled. 2013.

PERUCIA, A. S. *Desenvolvimento de Jogos Eletronicos - Teoria e Pratica*. [S.l.]: Novatec Editora., 2007.