Renato dos Santos Cerqueira

Felipe Pedrosa Martinez

Title Placeholder

Rio de Janeiro - RJ, Brasil

21/12/2012

Renato dos Santos Cerqueira

Felipe Pedrosa Martinez

Title Placeholder

Monografia apresentada para obtenção do Grau de Bacharel em Ciência da Computação pela Universidade Federal do Rio de Janeiro.

Orientador:
Adriano Joaquim de Oliveira Cruz

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
INSTITUTO DE MATEMÁTICA
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Rio de Janeiro - RJ, Brasil

21/12/2012

Monografia de Projeto Final de Graduação sob o título *"Title Placeholder"*, defendida por Renato dos Santos Cerqueirae Felipe Pedrosa Martinez e aprovada em 21/12/2012, no Rio de Janeiro, Estado do Rio de Janeiro, pela banca examinadora constituída pelos professores:

Prof. Ph.D. Adriano Joaquim de Oliveira Cruz
Orientador

???
Universidade ???

???
Universidade ???

Resumo

O objetivo deste trabalho é fazer um *engine* de jogo para o *videogame Nintendo DS*^{TM} e também um editor de fases e configurações, como seria feito numa equipe de desenvolvimento de um jogo grande, dando a possibilidade aos *designers* de fazerem seus *sprites* e criarem as fases com eles, sem que fosse necessário mexer com códigos.

Abstract

The objective of this paper is to make a game engine to the Nintendo DS^{TM}system and a level and configurations editor, as it would be done in a development team in a big game, giving designers the possibility to make their sprites and create their levels without touching actual source code.

Dedicatória

Agradecimentos

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução p. 11

1.1 Objetivo deste trabalho . . . . . p. 12

1.2 Estrutura da monografia . . . . . p. 13

2 Desenvolvendo para o Nintendo DS p. 14

2.1 Motivação . . . . . p. 14

2.2 Especificações . . . . . p. 14

2.3 Desenvolvimento . . . . . p. 15

3 A criação do jogo p. 17

4 A criação das ferramentas p. 18

4.1 Introdução . . . . . p. 19

4.2 O processo criativo . . . . . p. 19

4.2.1 Engine: O formato de background . . . . . p. 20

4.3 GFXTool: A ferramenta de edição gráfica . . . . . p. 21

4.4 Engine: Movimento e colisões . . . . . p. 27

5 Conclusões e trabalhos futuros p. 28

5.1 Conclusões . . . . . p. 29

Referências Bibliográficas p. 30

Anexo A – Figuras p. 31

Anexo B – Listagens de código p. 32

Anexo C – Ferramentas utilizadas p. 36

C.1 Qt . . . . . p. 36

Anexo D – Ferramentas utilizadas p. 37

Lista de Figuras

4.1 Uma idéia da nossa interface . . . . . p. 22

4.2 A nossa primeira interface . . . . . p. 23

A.1 Comparação de cores . . . . . p. 31

Lista de Tabelas

1 Introdução

"*Não há fé inabalável senão aquela que pode encarar a razão face a face, em todas as épocas da Humanidade.*"
Allan Kardec

Neste capítulo são apresentados o objetivo desta monografia e a estrutura da mesma.

1.1 Objetivo deste trabalho

A área de jogos eletrônicos é uma área relativamente nova, se comparada a outros ramos da ciência da computação. No entanto, por mais que compreenda conceitos importantes de computação, engloba conceitos importantes que fogem do seu escopo. Para o desenvolvimento de um jogo eletrônico, além de conhecimentos de linguagens de programação, algoritmos e estruturas de dados, interface humano-computador, inteligência artifical, etc; é preciso considerar aspectos mais subjetivos que dizem respeito a um jogo de forma geral, seja ele eletrônico ou não. O gameplay, ou a mecânica do jogo, ou seja, a forma com que o jogo vai se desenrolar e o seu funcionamento, é um exemplo claro de que o seu desenvolvimento e sua concepção vão muito além do que pode ser programado, compilado e executado em um computador.

É possível observar o crescimento da área se notarmos que o primeiro exemplo conhecido de jogo eletrônico aparece em 1947, quando Thomas T. Goldsmith Jr. e Estle Ray Mann introduziram sua patente para um Dispositivo de Entretenimento usando Tubo de Raios Catódicos em
[Goldsmith Jr. et al. 1948]. Quando vemos jogos produzidos na época do Atari 2600 e Magnavox Odyssey, estes eram criações de um único desenvolvedor, muitas vezes em períodos curtos de tempo, sem nem ao menos identificar o time de desenvolvimento. Era comum a aparição de *Easter Eggs* que os desenvolvedores usavam para tentar identificar a sua criação [Katie Salen e Eric Zimmerman 2006].

Mas observamos que com a tecnologia cada vez mais aprimorada, é possível vermos conceitos básicos que definiam os jogos daquela época serem trocados por formas cada vez mais rebuscadas. Podemos observar o desenvolvimento de novas áreas de jogos, onde antes haviam poucos estilos bem definidos, como corrida, luta e aventura, hoje há muitos mais, há jogos casuais, sociais e de multiplay massivo.

Hoje em dia a indústria de jogos conta com super produções com uma quantidade quase infindável de desenvolvedores, designers, criadores de fases, dentre outros tantos profissionais trabalhando cada um com sua especialidade.

Neste trabalho o que nos propomos a fazer é nos inserir no contexto desses times de criação modernos, vendo apenas uma pequena parte dessa grande cadeia de desenvolvimento: a criação de ferramentas para facilitar a interação entre equipes de ramos diferentes. Mais especificamente, criando um jogo em duas dimensões, do tipo plataforma para o videogame portátil Nintendo DS. Para isso, desenvolvemos uma engine, responsável por controlar todas as partes envolvidas no funcionamento do jogo como áudio, vídeo, estruturas de dados, controle de

personagens, fases, itens etc. Parte do nosso objetivo é tornar esta engine simples e de fácil reutilização. Isto foi feito através de ferramentas de criação e configuração dos componentes pertinentes ao mundo do jogo.

Nos propomos a fazer o papel do desenvolvedor: Criar as ferramentas, imaginando como os outros times as usariam e aprimorando ao máximo para esse fim.

1.2 Estrutura da monografia

No capítulo 4 é apresentado o estado da arte na área de segurança de redes ...

2 *Desenvolvendo para o Nintendo DS*

2.1 Motivação

Há poucos trabalhos que desenvolvam para plataformas diferentes do PC, seja pelo maior número de usuários da plataforma, maior versatilidade de ferramentas de desenvolvimento ou mesmo maior facilidade para encontrar documentação relativa ao desenvolvimento. A nossa motivação, além dessa pequena quantidade de trabalhos, é a de termos em nossas mãos, num console real, algo funcional feito por nossas mãos, e ao desenvolvermos as ferramentas, levarmos essa possibilidade para mais pessoas. Apesar das dificuldades no caminho.

2.2 Especificações

Nesse trabalho, nos focamos nas duas primeiras iterações do Nintendo DS, o Nintendo DS original, lançado no final de 2004, que normalmente é chamado de "DS Phat". E a segunda iteração, o Nintendo DS Lite, lançado no meio de 2006.

Ambos possuem hardware muito parecido, e as maiores diferenças são a estética e o contraste e iluminação da tela. Vamos às especificações:

Nintendo DS: Peso: 275 gramas Dimensões: 148.7mm x 84.7mm x 28.9mm Telas: Duas telas, ambas com 3 polegadas, de LCD TFT (Thin-Film Transistor) com 18-bit de cor, resolução de 256x192. As duas telas tem um espaço entre elas de aproximadamente 21mm, e a tela de baixo é possui touchscreen resistivo, que responde a um ponto de pressão por vez, fazendo a média no caso de vários pontos serem pressionados de uma vez. Possui iluminação traseira, que pode ser desligada por software. Entradas: Possui duas entradas de cartucho, uma para o formato de cartucho do seu antecessor, o Gameboy Advance(GBA), na parte inferior,e uma para seu próprio formato de cartucho na parte superior. Processadores: Possui dois processadores ARM, um ARM946E-S a 67MHz, que é o processador principal responsável principalmente pelo loop de jogo e renderização de vídeo e um coprocessador ARM7TDMI a 33MHz,

responsável pelo som, wi-fi, e, quando em modo GBA, passa a ter 16MHz, e passa a ser responsável pelo processamento principal. Memória: Possui 4MiB de RAM principal, expansíveis através da entrada de Gameboy Advance, no entanto, essa expansão só foi usada em jogos oficiais pelo Opera Browser. Wireless: Possui uma conexão IEEE802.11b, compatível com encriptação por WEP. WPA e WPA2 não são suportadas. Essa conexão também pode ser usada para comunicação entre consoles.

Nintendo DS Lite: Peso: 218 gramas Dimensões: 133mm x 73.9mm x 21.87mm Telas: Mesma especificação que o original, no entanto, possui quatro níveis de iluminação que podem ser reguladas por software. A tela também possui um contraste melhor que o original. Existem algumas diferenças em relação ao original, como a mudança do chip que controla o Wireless, o controlador do touch também é ligeiramente diferente, mas para fins práticos, o resto da configuração é igual ao original.

2.3 Desenvolvimento

O desenvolvimento oficial no DS segue os mesmos moldes do desenvolvimento em consoles diversos: a Nintendo possui um time interno chamado de Intelligent Systems Co., Ltd. que é responsável pelas ferramentas de desenvolvimento para vários consoles da Nintendo, dentre eles, o Nintendo DS. O desenvolvedor interessado pode entrar em contato com a empresa em busca das ferramentas, que incluem uma unidade especial que é conectada ao computador por meio de uma porta USB. Essa unidade possui o dobro de memória de uma unidade normal, para facilitar o Debug. No entanto, para adquiri-las, é necessário assinar um Acordo de Confidencialidade (do inglês, Non-Disclosure Agreement), que não permite que se comente ou escreva sobre esses kits de desenvolvimento. Por causa disto, é difícil encontrar mais detalhes acerca do funcionamento dessas unidades.

No entanto, normalmente, desenvolvedores pequenos não tem acesso ao kit de desenvolvimento oficial, seja por causa dos altos custos para adquirir o hardware, seja pela dificuldade de adquirir as licenças. Por isso não é fácil ver desenvolvedores pequenos publicando jogos para consoles.

Para nós, cabe recorrer ao desenvolvimento não oficial, apelidado de Homebrew. A cadeia de ferramentas usada é chamada devkitArm. Ela serve para compilar programas em C e C++ para consoles com processadores ARM. Ao contrário do desenvolvimento oficial, não é possível produzir cartuchos de jogos Homebrew. Para testarmos, no entanto, podemos fazer uso de emuladores ou de uma unidade normal com uso de um flash cart, que não é suportado pela

nintendo.

Em cima do devkitArm existe a libnds, feita especificamente para o hardware do DS. São funções de baixo nível que acessam todo o hardware: som, video, wireless, controle, e outras funções de hardware. O desenvolvimento nessa biblioteca não é exatamente fácil, já que muitas vezes é necessário o uso de endereços de memória, cópias diretas da memória para a memória de vídeo, dentre outras coisas que dificultam o desenvolvimento. Assim, surge em cima da libnds algumas bibliotecas que facilitam o uso.

Nesse caso, usamos a PALib, uma biblioteca escrita com intenção de facilitar o desenvolvimento de jogos em duas dimensões, que usem Sprites e mapas de fundo, que é o nosso caso.

Ainda assim, o uso dessas ferramentas permanece um mistério para a maioria, já que o jeito como as coisas são feitas (inclusões de imagens, geração dessas imagens e etc) é bem diferente de como é feito num programa normal, de desktop. E daí entra o conjunto de ferramentas que desenvolvemos. Como as ferramentas são, em sua maioria, visuais, o uso dessas ferramentas para criação assustam menos do que um conjunto pré-definido de lugares onde escrever arquivos e tipos de arquivos desconhecidos ao usuário.

3 A criação do jogo

Um jogo necessita de algumas partes constituintes, como por exemplo, suas fases e cenários, seus ítens, inimigos e personagem principal. No caso, tentamos cobrir essas partes em algumas ferramentas diferentes.

Em primeiro lugar, o usuário cria os cenários usando a BGTool, onde ele desenha os cenários em que o jogo vai se passar. Esses cenários ficam organizados logicamente em fases, e essas fases aparecem encadeadas para o usuário. Depois, o usuário cria os

4 A criação das ferramentas

"*Navegar é preciso, viver não.*"

Luís de Camões

Neste capítulo é apresentado o desenvolvimento da parte para o portátil Nintendo DS.

4.1 Introdução

O que é exatamente um jogo de plataforma em duas dimensões? Podemos pensar em exemplos clássicos, como Super Mario World para o Super Nintendo, ou Sonic The Hedgehog para o Sega Mega Drive.

Nem todos os jogos plataforma são em duas dimensões. Mas vamos aqui nos focar num jogo plataforma em duas dimensões para facilitar o entendimento e o desenvolvimento de soluções. Nessa modalidade, uma característica comum presente em muitos jogos é o uso de *tiles*, figuras de tamanho pré-definido, que compõem todo o mundo, como um grande quebra-cabeças. Nesse tipo de jogo, todos os objetos costumam ser feitos da combinações de sprites e muitas vezes um sprite é usado mais de uma vez em diferentes posições, para vários inimigos, por exemplo. Já os tiles são usados para compor o cenário, como pequenos tijolos.

Podemos definir um jogo de plataforma 2D por suas características comuns: o jogador controla um personagem que se movimenta para os lados, pode ou não saltar, ter ou não algum tipo de arma e em geral o objetivo é percorrer uma série de fases derrotando pequenos inimigos, para ao final derrotar algum grande inimigo e recuperar alguma coisa. Claro que essa é uma definição bem aberta, e é nela que vamos tentar nos inspirar ao escrever a parte que será executada no Nintendo DS.

Nosso objetivo é criar uma engine que leia algum tipo de arquivo de configuração, onde estarão detalhadas informações sobre quais são as imagens que compõem o personagem principal (ou personagens, no caso de haver uma escolha); informações sobre os inimigos como por exemplo a qual tipo de movimento ele obedece; descrição dos itens, como por exemplo qual efeito ele causa no personagem principal ou nos inimigos e informações sobre as fases, posicionamento de itens, inimigos e as imagens que as compõem.

4.2 O processo criativo

Na última seção, definimos o nosso objetivo. Assim, sabemos as características que queremos alcançar ao fazer nosso jogo. Devemos então começar a buscar ferramentas para alcançar o nosso objetivo. Começamos então nos focando nos elementos mais básicos: vamos pensar em como desenharemos o mundo, ignorando a existência dos inimigos e do jogador. Vamos pensar em como desenharemos o plano de fundo e os elementos imóveis do mundo.

Tentaremos seguir a idéia apresentada em [Frederico Mameri e Rômulo Nascimento 2006], cuja aplicação é direta para nós.

Enquanto não criamos o editor de fases, faremos uma pequena demonstração carregando uma matriz manualmente em nosso código, bem simples, delimitada por caracteres que representam determinados *tiles*.

Estamos usando o kit de desenvolvimento *homebrew*[1]para o Nintendo DS, e com ele, a PALib, uma biblioteca que facilita o uso dos recursos de hardware do DS, que são um pouco limitados e curiosos, devido aos seus dois processadores e telas.

Ao investigarmos a fundo, descobrimos que o DS suporta mapas com *tiles*, ou seja, podemos criar um mapa e depois desconstruí-lo em pequenas imagens que serão repetidas várias vezes. Vamos então descobrir como podemos usar dessa característica.

4.2.1 Engine: O formato de background

Investigamos o formato que é aceito pelo DS, em [Martin Korth 2007]. Quando uma figura é colocada como plano de fundo, precisamos de quatro partes:

- Arquivo Pal
  Neste arquivo, temos informações sobre a paleta usada no arquivo. São descritas todas as cores contidas no arquivo. O padrão é de 256 cores. O Formato usado no arquivo é A1B5G5R5, o que significa que tem um bit para o caso da cor ser transparente, 5 para o verde, 5 para o azul e 5 para o vermelho. A primeira cor no arquivo é a cor usada como transparência.

- Arquivo Tiles
  Neste arquivo, ficam os tiles, cada um de 8x8 pixels. Cada pixel do tile referencia uma cor do arquivo Pal. Neste arquivo, os tiles são escritos em blocos (vetores) de 64 bits.

- Arquivo Map
  Neste arquivo, fica o mapa do background, ele referencia os tiles. Os bits 0-9 indicam o tile, o bit 10 indica se é espelhado na horizontal e o bit 11 indica se é espelhado na vertical.

- Arquivo .c
  Este arquivo constrói a struct que será usada no programa, informa algumas coisas como

[1]O termo Homebrew se refere ao desenvolvimento com o uso de um kit de desenvolvimento extra-oficial, produzido por uma comunidade. Muitas vezes o kit de desenvolvimento oficial tem custo proibitivo e é voltado somente para grandes empresas de desenvolvimento. Desenvolvedores que queiram desenvolver em casa se valem desses kits não-oficiais, muitas vezes de código aberto, que são desenvolvidos em geral fazendo algum tipo de engenharia reversa no hardware original.

o tipo do mapa, o seu tamanho, os ponteiros para as regiões onde ficaram os três arquivos acima depois de linkados, tamanho do tiles em bytes e tamanho do map em bytes.

Veja a listagem de código B.1

Temos alguns detalhes a discutir nesse ponto. Como visto em [Martin Korth 2007], o DS tem mais de um formato de background, o que está descrito acima é o mais simples - e menos poderoso - dentre eles. Escolhemos ele como ponto de partida, para, se necessário, implementar os outros. Por exemplo, existem modos de usar mais de um arquivo de paleta, ou ainda usar mais tiles, mas sem o espelhamento. Ignoraremos esses outros formatos e modos, e nos focaremos no que está descrito acima.

Portanto, precisaremos que a nossa ferramenta dê a saída nesse formato. Por enquanto, para os testes, pensamos em usar ferramenta que vem junto com o kit de desenvolvimento, ela só converte uma imagem já pronta (ou seja, um mapa que já tenha sido desenhado) para este formato. Mas assim poderíamos fazer testes com relação a movimentação do background. No entanto, é muito difícil editar os arquivos de background manualmente, de modo que teremos de dar início a ferramenta de edição antes do esperado. Vamos começar fazendo ela como uma simples ferramenta de edição de background.

4.3 GFXTool: A ferramenta de edição gráfica

Começaremos então a criar a ferramenta. A idéia é que tenhamos uma área principal, onde será mostrado o que temos atualmente no mapa, uma área na lateral onde estarão os tiles que o usuário poderá usar para compor o background e algum botão que faça possível que o usuário edite, no próprio programa, os tiles. Ou crie novos.

Comecemos então por uma idéia básica, a de deixar o usuário importar uma imagem já pronta, e que o programa identifique os tiles contidos na imagem. Desta maneira, poderemos usar uma fase já pronta para fazer nossos testes, e o usuário poderia migrar um trabalho anterior para a nova engine. Numa grande empresa, poderia ser o caso de estar havendo uma adaptação de um jogo de uma plataforma antiga para o Nintendo DS.

Precisamos então decidir como faremos a interface. Como a parte do código da engine para o DS deverá ser feito em C++, pensamos que o ideal seria que o resto do projeto também fosse feito em C++, assim mantemos um certo padrão de linguagem. Com isso, buscamos as alternativas para a interface.

Como queremos que o projeto seja utilizável tanto em Windows, quanto em Linux, que será

Figura 4.1: Uma idéia da nossa interface

nossa principal plataforma de desenvolvimento, precisamos escolher uma biblioteca gráfica que suporte os dois sistemas operacionais e cujo custo de trocar de um pra outro seja muito baixo, ou seja, que não seja necessário reeescrever código para isso.

A solução que encontramos é utilizar a biblioteca Qt. A IDE para criação de interfaces, o QtCreator, deixa que as interfaces sejam criadas somente arrastando componentes e em seguida associando cliques a funções. E usando as funções do Qt para fazer as tarefas, o custo de troca entre Windows e Linux é basicamente zero. Falaremos mais sobre o porque da decisão da biblioteca no Anexo C.1.

Assim, fazemos a primeira janela do nosso programa, e o resultado é bem parecido com o que nós pensamos.

Com a nossa interface definida, agora temos que nos preocupar em carregar o arquivo. Vamos considerar primeiro o problema de ler um arquivo já pronto e conversão desse arquivo para um formato do tipo “mapa de tiles”, onde nós identificamos os tiles, e iremos compor o background a partir desses tiles. Fazemos isso com o código B.2.

Vamos entender o que fazemos nesse código. A leitura da imagem fica por conta do Qt. Preenchemos o fundo do grid que vai conter nosso background com preto, em seguida, iteramos por toda a imagem, dividindo-a em quadrados de 8x8, ou seja, nossos tiles. Em cada um desses quadrados, identificamos se esse tile já foi incluído no nosso vetor de tiles. Se já, pegamos o índice desse vetor caso contrário, incluímos no vetor de tiles. Depois dizemos que nessa posição do background está o tile com esse índice.

Inicialmente, não deixamos que o usuário amplie o mapa, ou seja, a imagem que ele carregar

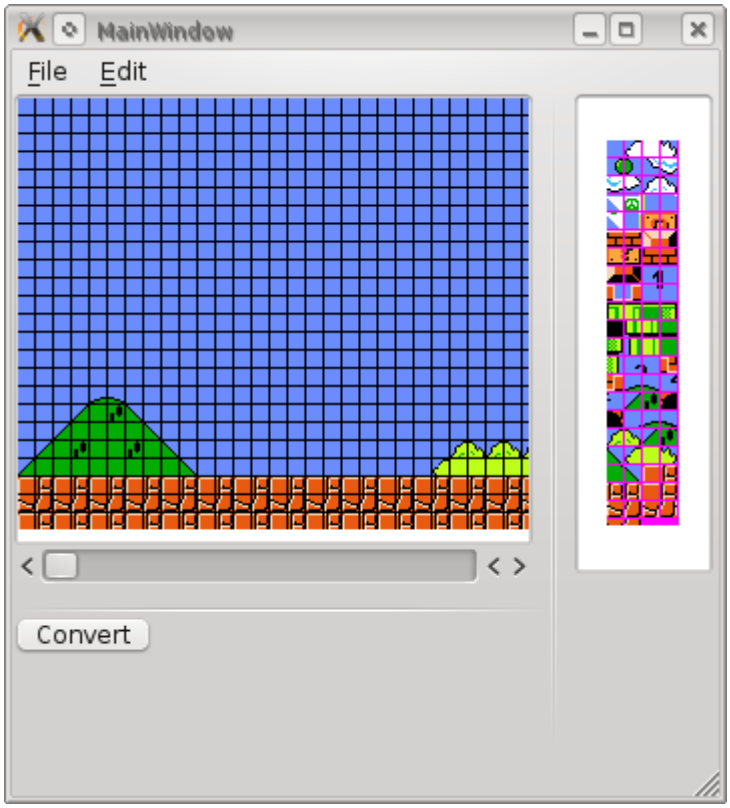

Figura 4.2: A nossa primeira interface

inicialmente tem que ser composta por tiles e já do tamanho final do mapa. Nossa intenção para depois, no entanto, é que o usuário possa criar um mapa vazio e carregar somente o conjunto de tiles. Do contrário, nosso software se resumiria a ter a mesma função da ferramenta já existente, a PAGfx, somente com uma interface mais rebuscada.

Como já comentamos, o hardware do DS suporta que pedaços de mapa sejam espelhados, tanto na vertical quanto na horizontal. No nosso código, não tratamos nenhum desses casos. Esse problema será visto mais adiante. No momento estamos mais preocupados em ter algo que permita que exportemos os backgrounds para que possamos começar a trabalhar na engine.

Agora que já lemos o background, temos que implementar as seguintes funções: seleção de tiles, usar o tile selecionado para pintar no mapa e exportar o mapa para o formato do DS. Como já exibimos na pequena janela o tile, basta que associemos os eventos de clique à janela, para que possamos implementar a funcionalidade de seleção de tiles.

Nós em primeiro lugar fizemos um código bem rudimentar, que simplesmente achava a posição do tile a partir da posição que o usuário havia clicado, e desenhava um quadrado ao redor dessa posição. No entanto, ele desenhava múltiplos quadrados, no caso do usuário sair clicando, e não funcionava no caso em que o usuário clicava duas vezes no mesmo quadrado.

Mexemos no código até que o usuário pudesse selecionar um tile com um clique, e desfazer a seleção clicando novamente. E também passamos a guardar a informação sobre o índice do tile selecionado. Afinal, isso seria necessário para quando quiséssemos pintar usando o tile selecionado. Uma funcionalidade interessante seria selecionar grupos de tiles, mas nós decidimos não a implementar num primeiro momento.

Com isso, implementar o uso do tile para pintar no mapa era o próximo passo lógico. Tratamos disso fazendo a janela um pouco mais complexa, adicionamos um botão "pintar" e quando este botão está selecionado, o sprite selecionado é usado para pintar na janela principal. Agora, já pensando em como faremos para exportar o mapa, precisamos tomar algumas medidas e refatorar algumas partes do código. Em primeiro lugar, como vimos anteriormente, vamos precisar de três partes: uma paleta de cores, os sprites presentes, e um mapa desses sprites. No momento nosso código não guarda nenhuma dessas informações. Então o que vamos refatorar é o código de leitura da imagem inicial.

Precisamos fazer com que ao ler a imagem, e processar os sprites, sejam guardadas as cores que formos encontrando. Guardamos então essas cores num vetor, para que depois, quando formos processar a saída, possamos fazer os sprites se relacionarem com essas cores. Os sprites já estão guardados eles mesmos num vetor e o basta então que o fundo também seja armazenado numa matriz, relacionando-se com os sprites.

Ao menos a princípio, escolhemos, como é comum em aplicações gráficas, o magenta para ser o transparente, ou seja, a primeira cor da paleta. Do mesmo modo, enquanto processamos as cores, já faremos a passagem para A1R5G5B5, que tem somente 5 bits de cor, ao contrário dos 8 que temos. Guardaremos então os 5 bits mais significativos. Veja a figura A.1 para ter uma idéia da diferença que tirar os três bits menos significativos faz.

Ou seja, o procedimento passa a ser:

Listing 4.1: Pseudo-código de carregamento de arquivo

```
LeImagem();
CorTransparente = Magenta;
TamanhoDoSprite = 8x8;
InsereNaPaleta(CorTransparente);
ParaCadaUm pixelsNaHorizontal faca
  ParaCadaUm pixelsNaVertical faca
    Comeca
      Cor = Faz3BitsMenosSignificativosDeCadaCorZero(CorDoPixel);
      Se Cor naoExiste em paleta
        InsereNaPaleta(Cor);
```

```
      InserePixelNoSprite(Cor,Sprite);
      Se Sprite temTamanho tamanhoDoSprite
      Comeca
        Se Sprite naoExiste em VetorDeSprites
          Comeca
            InsereNoVetorDeSprites(Sprite);
            IndiceSprite = VetorDeSprites.Ultimo;
          Termina;
        Senao
          IndiceSprite = IndiceNoVetorDeSprites(Sprite);

        PintaNaTela(Sprite);
        PintaNaTela(QuadradoGrid, PosicaoSprite);
        MatrizMapa[PosicaoSprite] = IndiceSprite;

        ZeraSprite(Sprite);
      Termina;
    Termina;

ParaCadaUm VetorSprites faca
  PintaNoGridDeSprites(Sprite)
```

Com isso, chegamos basicamente onde queríamos chegar. Podemos agora usar o nosso programa para fazer o background para ser testado no DS. Porém, falta ainda dar a saída para o formato da PALib. Mas agora isso é só uma questão de dar a saída do jeito certo, já que estamos com todos os dados preparados.

Fazemos isso com o seguinte código:

Listing 4.2: Pseudo-código de escrita de arquivo

```
arquivoPal = AbreArquivo(pal.bin);
contador = 0;
enquanto ( contador < vetorPaleta.tamanho )
{
  cor = vetorPaleta[contador];

  bitAlto = 1;
  bitAlto = shiftLeft(bitAlto,5);
  bitAlto = bitAlto + shiftRight(cor.azul,3);
  bitAlto = shiftLeft(bitAlto,2);
  bitAlto = bitAlto + shiftRight(cor.verde,6);
```

```
  bitBaixo = shiftRight(seisBitsMaisBaixos(cor.verde),3);
  bitBaixo = shiftLeft(bitBaixo,5);
  bitBaixo = bitBaixo + shiftRight(cor.vermelho,3);

  escreveChar(arquivoPal, bitBaixo);
  escreveChar(arquivoPal, bitAlto);
}
enquanto ( contador < 256 )
{
  escreveChar(arquivoPal, 0);
  escreveChar(arquivoPal, 0);
}
FechaArquivo(arquivoPal);

arquivoTiles = AbreArquivo(tiles.bin);
paraCada tile em vetorTiles
{
  paraCada pixel em tile
  {
    indicePaleta = devolveIndicePelaCor(pixel, vetorPaleta);
    escreveChar(arquivoTiles, indicePaleta);
  }
}
contador = 0;
enquanto contador < 64
{
  escreveChar(arquivoTiles, 0);
}
fechaArquivo(arquivoTiles);

arquivoMap = AbreArquivo(map.bin);
paraCada indice em matrizMap
{
  escreveChar(arquivoMap,indice);
}
fechaArquivo(arquivoMap);
```

Assim, agora podemos usar qualquer programa de edição de imagem para fazer os tiles, e em seguida, podemos usá-los para construir algum tipo de background para a nossa engine. O DS suporta até cinco backgrounds simultâneos, para que eles façam parallax ou simplesmente para dar ilusão de profundidade. Ainda não implementamos essa funcionalidade no nosso software, mas podemos circular essa limitação simplesmente editando os cinco backgrounds em

momentos diferentes.

4.4 Engine: Movimento e colisões

Agora, já tendo os meios para fazer o plano de fundo, é hora de voltar para a engine, começar a pensar no movimento básico do personagem principal, e nas colisões dele (e de outros personagens, como inimigos) com o ambiente. Nesse ponto, estamos observando as idéias mostradas em [Raigan Burns e Mare Sheppard 2009, N Tutorial A] e
[Raigan Burns e Mare Sheppard 2009, N Tutorial B].

Assim, a nossa idéia para a colisão foi de implementarmos, no mapa de tiles, quais tiles são sólidos e quais não são. Do mesmo modo, podemos fazer tiles que só são sólidos a partir de determinadas direções. Assim, podemos implementar plataformas onde o personagem consegue subir, mas uma vez em cima, não cai. Ao mesmo tempo, escolhemos como padrão para o personagem uma caixa envolvente em formato de pílula, assim como é feito pela Unity 3D.[2]

Desse modo, nossa primeira preocupação é fazer a colisão entre o personagem e o mundo. Tentaremos fazer as rotinas o mais genéricas possíveis, para que possam ser aproveitadas para todos os nossos objetos dinâmicos. (Personagens, itens que andam e inimigos.)

3

[3]http://unity3d.com/ - Engine para jogos em 3D.

5 *Conclusões e trabalhos futuros*

"*Nada se cria, nada se perde, tudo se transforma.*"

Lavousier

Neste capítulo é apresentado as conclusões e alguns trabalhos futuros ...

5.1 Conclusões

Alguns itens interessantes para a conclusão de um projeto de graduação

Qual foi o resultado do seu trabalho? melhora na área, testes positivos ou negativos? Você acha que o mecanismo gerado produziu resultados interessantes? Quais os problemas que você encontrou na elaboração do projeto? E na implementação do protótipo? Que conclusão você tirou das ferramentas utilizadas? (heurísticas, prolog, ALE, banco de dados). Em que outras áreas você julga que este trabalho seria interessante de ser aplicado? Que tipo de continuidade você daria a este trabalho? Que tipo de conhecimento foi necessário para este projeto de graduação? Para que serviu este trabalho na sua formação?

Referências Bibliográficas

[Frederico Mameri e Rômulo Nascimento 2006]*A Fast Culling Algorithm for 2D and 2.5D Side-scrolling Games*.

[Goldsmith Jr. et al. 1948]Goldsmith Jr., Thomas T., Ray e Mann Estle. *Cathode-ray tube amusement device*. December 1948. 2455992. Disponível em: <http://www.google.com/patents?vid=2455992>.

[Katie Salen e Eric Zimmerman 2006]Katie Salen; Eric Zimmerman. Adventure as a video game. adventure for the atari 2600. [1983-84]. In: *The Game Design Reader: A Rules of Play Anthology*. [S.l.]: MIT Press, 2006. p. 690–713.

[Martin Korth 2007]Martin Korth. *"GBATEK - Gameboy Advance / Nintendo DS - Technical Info"*. 2007. Disponível em: <http://nocash.emubase.de/gbatek.htm>.

[Raigan Burns e Mare Sheppard 2009]Raigan Burns; Mare Sheppard. *"N Tutorial A - Collision Detection and Response"*. 2009. Disponível em: <http://www.metanetsoftware.com/technique/tutorialA.html>.

[Raigan Burns e Mare Sheppard 2009]Raigan Burns; Mare Sheppard. *"N Tutorial B - Broad-Phase Collision"*. 2009. Disponível em: <http://www.metanetsoftware.com/technique/tutorialB.html>.

ANEXO A – Figuras

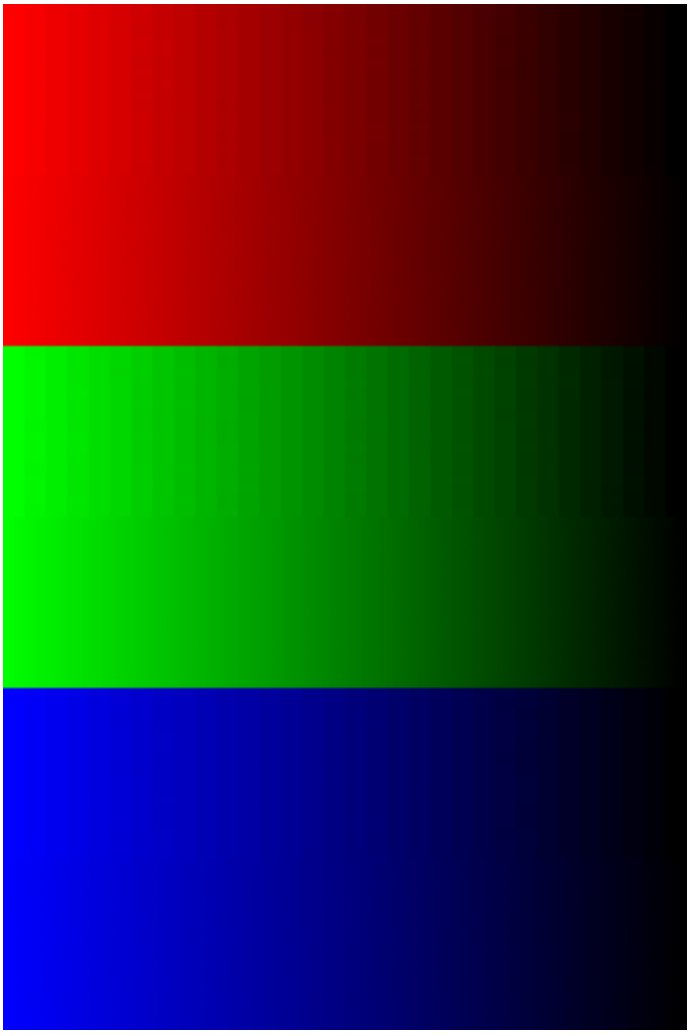

Figura A.1: Comparação de cores: Em cima, a escala sem os três bits menos significativos. Embaixo, com eles. Note a gradação mais suave na parte com mais bits.

ANEXO B – Listagens de código

Listing B.1: Exemplo de arquivo de mapa de background .c:

```
#include <PA_BgStruct.h>

extern const char bg_Tiles[];
extern const char bg_Map[];
extern const char bg_Pal[];

const PA_BgStruct bg = {
  PA_BgLarge,
  3392, 256,

  bg_Tiles,
  bg_Map,
  {bg_Pal},

  5312,
  {27136}
};
```

Listing B.2: Lendo os tiles e os identificando (v1)

```
void MainWindow::on_pushButton_clicked()
{
    QGraphicsView *w = ui->visualizationView;
    QGraphicsView *s = ui->spritesView;
    QGraphicsScene *scn = new QGraphicsScene(w);
    QGraphicsScene *sScn = new QGraphicsScene(s);

    w->setScene(scn);
    s->setScene(sScn);
    QPixmap pix("../gfx/teste.png");
```

```
QImage img(pix.toImage());
int newHeight = pix.height()/8-1 + pix.height();
int newWidth = pix.width()/8-1 + pix.width();
QImage imgGrid;
imgGrid = QImage(newWidth, newHeight, img.format());

std::cout << "Height " << newHeight << std::endl << "Width " <<
   newWidth << std::endl;

imgGrid.fill(QColor(0,0,0).rgb());
QImage spriteGrid;
{
    std::vector<QImage> sprites;

    int i; int j;
    int startI; int startJ; startI = startJ = 0;

    int spritesI = pix.height() / 8;
    int spritesJ = pix.width() / 8;

    for ( i = 0 ; i < spritesI ; i++ )
    {
        for ( j = 0 ; j < spritesJ ; j++ )
        {
            QImage sprite = QImage(8,8,img.format());
            for ( int k = 0 ; k < 8 ; k++ )
            {
                for ( int l = 0 ; l < 8 ; l++ )
                {
                    sprite.setPixel(l,k,img.pixel(j*8 + l,i*8+k));
                    imgGrid.setPixel(j*9+l,i*9+k, img.pixel(j*8 + l,i
                       *8+k));
                }
            }

            int exists = 0;

            for ( std::vector<QImage>::iterator it = sprites.begin();
               it != sprites.end() ; it++ )
            {
                if ( (*it == sprite) )
                {
                    exists = 1;
```

```
                    break;
                }
            }

            if ( !exists )
            {
                std::cout << "New sprite at " << j*8 << "," << i*8 <<
                   std::endl;
                sprites.push_back(sprite);
            }
        }
    }

    spriteGrid = QImage(8*4+4, 2*sprites.size()+(2*sprites.size()/8-1),
        img.format());
    spriteGrid.fill(QColor(255,0,255).rgb());
    int m = 0;
    int n = 0;
    for ( std::vector<QImage>::iterator it = sprites.begin(); it !=
       sprites.end() ; it++ )
    {
        for ( int k = 0 ; k < 8 ; k++ )
        {
            for ( int l = 0 ; l < 8 ; l++ )
            {
                spriteGrid.setPixel(l+m*9,k+n*9,(*it).pixel(l,k));
            }
        }

        if ( m != 3 ) m++;
        else { m = 0; n++; }
    }
}

QPixmap pixGrid;
pixGrid = QPixmap::fromImage(imgGrid);
QPixmap pixSprGrid;
pixSprGrid = QPixmap::fromImage(spriteGrid);

scn->setSceneRect(pixGrid.rect());
scn->addPixmap(pixGrid);

sScn->setSceneRect(pixSprGrid.rect());
```

```
    sScn->addPixmap(pixSprGrid);
    w->show();
    s->show();
}
```

ANEXO C – Ferramentas utilizadas

C.1 Qt

ANEXO D – Ferramentas utilizadas

Foi feita uma análise de algumas ferramentas que são muito usadas por atacantes (hackers) para a confecção de ataques. Estas ferramentas são muito úteis em vários aspectos, tais como: (1) o levantamento de informações sobre o alvo, (2) que tipo de serviços estão disponíveis no alvo, (3) quais as possíveis vulnerabilidades do alvo, entre outras informações. As ferramentas analisadas foram o *nmap* , o *nessus* , o *saint* , além de alguns comandos de sistemas operacionais (UNIX-Like e Windows-Like) usados para rede, tais como o *ping, nslookup e whois*.