Desenvolvimento de jogos

Marcelo Elias Del Valle (marceloelias@iname.com)
Colaboradores: Mário Lacroix e Renato Soeiro

02/01/2002

Sumário

Parte I

Módulo básico - C/SDL

Capítulo 1

Introdução

Embora o mercado de trabalho na industria de entretenimento brasileira seja enorme, desenvolvimento de jogos parece ser hoje, em 2002, uma área restrita ao exterior. Os poucos desenvolvedores brasileiros que se aventuram na área acabam, por muitas vezes, se espelhando nos desenvolvedores de outros países, tanto em tecnologia quanto em inspiração/enredo/história.

O brasil não é forte em desenvolvimento de jogos, contúdo, não por falta de talentos, muitos profissionais daqui são inclusive "exportados" - desejam muito desenvolver jogos e não encontram trabalho na área aqui no Brasil, então procuram fora.

A pergunta é: se temos mercado, talentos e muita capacidade, por que o número de empresas de desenvolvimento de jogos no brasil é tão pequeno ainda? Na opinião pessoal do autor desse texto, pelos seguintes motivos:

1. Concorrência muito forte com empresas estrangeiras já consolidadas na área

2. Falta de seriedade da grande maioria dos desenvolvedores brasileiros

3. Sub-valorização da área

Não é preciso ser gênio para enxergar o ítem 2. Existem faculdades especializadas em cursos de desenvolvimento de jogos nos EUA, Japão e Alemanha e a maioria das empresas nesses países vem desenvolvendo esse tipo de tecnologia muito antes do computadores começar a popularizar aqui no Brasil.

O ítem 3 talvez seja o mais facilmente percebido no dia a dia. Raras são as pessoas que conseguem encarar desenvolvimento de jogos como trabalho. A maioria encara, no máximo, como distração. Isso faz com que a área não seja encarada com seriedade por muitos desenvolvedores, que acabam dando baixa prioridade a isso (ítem 2) e faz também com que muitos outros tipos de profissionais encarem isso como motivo de riso, tornando comuns frases como: "Vocês deveriam estar desenvolvendo algo sério, que realmente tivesse utilidade".

Desenvolvimento de jogos, contudo, deve ser encarada como uma área séria tanto no tocante à pesquisa quanto à aplicação. Em países de primeiro mundo, essa é uma das áreas que puxam vários e vários ramos de pesquisa, pricipalmente aqueles relacionados com computação gráfica e com a indústria bélica.

Porém, o motivo maior que faz com que desenvolvimento de jogos seja uma área de tão grande importância, principalmente no Brasil, é simples: é um poderoso tipo de mídia, aliás, de multimídia. Esse é o ponto mais importante a ser destacado aqui. Existem no Brasil dois tipos de profissionais diretamente relacionados com desenvolvimento de jogos: programadores e artistas.

Esses dois, cada um do seu lado, costumam enxergar desenvolvimento de jogos como sendo uma sub-área de artes ou de programação. O fato é que desenvolvimento de jogos não pode ser encarado dessa maneira. Desenvolvimento de jogos deve ser comparado com áreas como televisão, histórias em quadrinhos, construção de websites e outros similares, não como um produto desse tipo de mídia, pelo contrário, desenvolvimento de jogos é uma área totalmente singular, mas assim como essas mídias, exerce forte influência na cabeça das pessoas e muitas vezes abrange profissionais de várias áreas.

Essa é, portanto, a primeira coisa a ser aprendida sobre o curso: esse não é um curso de programação de jogos, nem de desenho ou modelagem ou recursos computacionais, é um curso sobre desenvolvimento de jogos e tudo o que essa expressão abrange, incluindo aí modelagem, responsabilidade, história, metodologia de projeto, adequação à realidade brasileira, matemática, trabalho em grupo e, claro, programação.

1.1 Audiência esperada

Esse curso é dividido em vários módulos e cada módulo tem suas peculiaridades com relação audiência esperada. Esse primeiro módulo

é mais voltado para programadores, embora também possa ser lido por artistas mais voltados para a área de exatas.

1.2 Pré-requisitos

Basicamente, é necessário saber programar em linguagem C. Um bom conhecimento sobre ponteiros e sobre estruturas ajudaria bastante no acompanhamento desse módulo.

Antes de entrarmos na parte de programação, contúdo, definiremos vários conceitos, úteis para profissionais de qualquer área (relacionada a desenvolvimento de jogos), como metodologia de projetos e uma explicação sobre como funcionam as cores e a parte gráfica de computadores e equipamentos similares.

No tocante a harware e software, o leitor que quiser experimentar os exemplos descritos nesse texto, o que é altamente recomendável, deverá possuir uma máquina com espaço em disco o suficiente para instalar o ambiente cygwin e SDL no caso do windows e gcc e SDL no caso de linux, sendo que uma máquina com processamento maior ou igual a de um pentium 100 seria recomendável.

Usuários de máquina Apple com MacOS X e usuários de BeOS também são capazes de seguir o livro, mas deverão ter alguns compilador compatível com gcc e SDL instalados em seu sistema. Esse texto só descreve, porém, a instalação desses ambientes em linux e windows.

1.3 Como funciona o curso?

Esse curso é patrocinado pelo web portal MPage - O Portal Brasileiro de Desenvolvimento de Jogos e é ministrado inteiramente via internet. Isso não significa que é necessário estar conectado para fazer o curso, mas significa que uma conexão à internet com suporte à www é necessário.

O curso é composto de vários módulos, cada um com documentação relacionada, a qual é disponibilizada para todos os usuários do portal. Acesse o portal em http://mlinuxer.cjb.net `http://mlinuxer.cjb.net`para tornar-se um usuário.

O curso não provê apenas documentação, provê também de recursos como fórum, mural de avisos e lista de discussão, utilizando os recursos do portal, reuniões via bate-papo e exercícios. No caso dos exercícios, o aluno matriculado acessa listas de exercícios deixadas no

portal, resolve as mesmas e envia por email para correção (uma por vez, normalmente). Essas listas são corrigidas e enviadas de volta para o aluno, de forma que ele consiga ter ciência de como está indo no curso, consiga ter e sanar dúvidas e aprender o máximo possível.

Futuras versões desse curso, de forma mais interativa, usando recursos de vídeo e áudio conferência, são desejadas e provavelmente serão implementadas um dia, tão logo essas tecnologias popularizem o suficiente para ser viável o seu uso.

1.4 Licença

Toda a documentação desse curso é licenciada pela Licença Pública GNU `http://www.gnu.org`**. Em termos gerais, isso significa que esse texto pode ser distribuido livremente, desde que bem preservados os direitos autorais tanto do autor original quanto de autores de modificações. Caso você pretenda distribuir esse documento, seria ótimo se fosse feita uma notificação para o webmaster do portal.**

Por fim, essa documentação não pode ser vendida. Nada impede, contúdo, que alguém ganhe dinheiro com produtos associados a ela (por exemplo, distribuir essa documentação em CDROM e cobrar pela mídia e pelo serviço de gravação).

Capítulo 2

Noções básicas

Agora que já foi falado bastante sobre o curso, será abordado de forma inicial o assunto do módulo propriamente dito.

2.1 Um pouco de história

Os jogos existem desde a pré-história. É natural do ser-humano jogar, tanto é que jogar é divertido. Jogar pode ser divertido por diversos motivos, talvez por treinar capacidades no jogador sem que esse tenha de assumir responsabilidades (outra coisa que existe desde a pré-história) ou ter de fazê-lo de forma entediante.

Os jogos eletrônicos e de computador só surgiram na segunda metade do século XX, quando os avanços tecnológicos permitiram seu surgimento. Muitos ainda se lembram das antigas máquinas de fliperama, do famoso telejogo, do atari e de todos os videogames similares que viram surgindo em sequência.

Naquela época, por recursos de processamento e memória serem altamente escassos e pelo fato de ser necessário um conhecimento técnico aprofundado para desenvolver jogos para essas máquinas, desenvolvimento de jogos era uma área quase totalmente restrita a engenheiros. Isso continuou assim inclusive com o surgimento dos PCs (computadores pessoais) da linha x86, que rodavam o sistema operacional DOS.

Uma das partes mais difíceis de desenvolver jogos nessa época era saber programar os chips no computador (mais epecificamente os chips de audio e vídeo) de forma a produzir animações, gráficos e soms da forma mais rápida e eficiente possível a partir de comandos do usuário em dispositivos de entrada.

Para isso, programadores da época usavam linguagem assembly, própria para se usar quando um chip do computador manualmente, ao invés de pedir para que o sistema operacional o faça (o que era bem mais lento). Programadores usavam essa linguagem para programar os chips gráficos CGA, VGA, SVGA, de audio como SoundBlaster e AdLib e outros comuns na época. Era incrivelmente pequeno o número de placas e chips de som e vídeo que existiam.

Uma peculiaridade importante é que o sistema DOS sempre funcionou em mono tarefa, ou seja, somente um programa é executado por vez. Isso fazia com que a programação em linguagem assembly fosse extremamente fácil, comparando ao que é hoje. Com o surgimento do modo protegido de memória, do 386dx em diante, a programação torna-se muito mais complicada. Não seria viável hoje programar um jogo em linux ou windows usando linguagem assembly.

Além disso, vários outros avanços tecnológicos aconteceram, desde aumento da quantidade de processamento e de memória dos PCs como um significativo aumento no número de dispositivos de som, vídeo e multimídia existentes hoje. Placas aceleradoras 3D que algumas vezes são mais potentes que os próprios PCs que as comportam tem se tornado cada vez mais populares.

Por fim, os avanços foram tão grandes que permitiram que não apenas pessoas com um grande conhecimento técnico sejam capazes de criar jogos, mesmo um artista pode criar hoje, sozinho, jogos relativamente complicados.

No panorâma atual vemos basicamente três frentes (grupos) de desenvolvimento de jogos no Brasil hoje em dia. Jogos multimídia feitos com ferramentas de criação prontas como Director, Click&Play e MultimediaFusion, normalmente na forma de livro interativo, jogos de pintar e infantis, todos esses se encaixam num primeiro grupo. Outro grupo seriam jogos estáticos, ou seja, de cartas ou tabuleiro, sem necessidade de performance alguma.

O último grupo, que será o assunto mais tratado nesse texto, seriam jogos orientados a performance, que além de requerem muito mais atenção do processador normalmente tem uma lógica muito mais completa e específica. Se encaixam nessa categoria, por exemplo, jogos de simulação, de estratégia em tempo real, jogos de aventura em 3D, jogos de plataforma, etc.

A tendência parece ser que cada vez mais esses grupos se dividam, sendo que em qualquer um dos três seja inevitável o predomínio de jogos online, que podem ser jogados via rede.

Além disso, esse terceiro tipo de jogo não é mais programado hoje usando-se linguagem assembly, certamente. A variedade de peças de hardware (como placas multimídia) já é tão grande que se torna necessário ao programador uma forma de criar um único programa que rode com boa performance em qualquer máquina, qualquer que seja o hardware que o usuário do jogo esteja utilizando.

Com a aparente perda crescente de monopólio que a Microsoft, produtora do windows, vem sofrendo, também é necessário que os jogos se tornem portáveis entre sistemas operacionais, como linux, windows, MacOS X, BeOS, PlayStation, etc.

Conforme veremos em partes posteriores desse texto, o uso de linguagem C com a biblioteca de programação SDL (Simple DirectMedia Layer) é a melhor combinação atualmente para obtermos esse resultado.

2.2 Conceitos básicos sobre desenvolvimento de jogos

Muitos desenvolvedores iniciantes, sobretudo programadores, pensam que criar um jogo é simplesmente sentar em frente ao computador e começar a desenvolver. Porém, essa parte, a de implementação, é normalmente cerca de 10% da criação de um jogo real. Isso vale para quase todo tipo de projeto. Antes de chegar na parte de implementação, é necessário definir muito bem como será o jogo a ser feito, analisar os problemas que surgem da definição, projetar o jogo e finalmente implementá-lo.

Etapas na criação de um jogo

É interessante que as etapas de criação de um jogo estejam formalmente definidas, mesmo que o desenvolvedor ou o time de desenvolvimento venham a quebrar as regras provenientes dessa formalização.

Essa etapas existem para que seja mais fácil a comparação entre projetos distintos, para se ter uma idéia exata do que se está fazendo e, principalmente, para que desenvolvedores (não só programadores) possam conversar entre si usando uma linguagem em comum.

As etapas de criação de um jogo podem ser definidas como abaixo, não necessariamente seguindo essa ordem:

1. Definição do enredo e do tipo de jogo (funcionamento básico)
2. Descrição formal do jogo, incluindo descrição detalhada do funcionamento e da interface
3. Criação de recursos gráficos
4. Criação de recursos sonoros
5. Análise
6. Projeto
7. Implementação
8. Depuração e testes

Tenha em mente que essas etapas não necessariamente precisam estar presentes no desenvolvimento de qualquer jogo, talvez não sejam necessárias para implementar um jogo simples como tetris, mas é extremamente util ter essas etapas em mente para se ter ciência do que está ou não sendo desenvolvido na criação do jogo.

A primeira etapa consciste em definir-se, superficialmente, como será o jogo, ou seja, qual será a linha de raciocínio empregada na descrição formal. Os peritos em metodologia de projeto custumam dizer que existe uma etapa adicional antes dessa, chamada Brain Storm, ou chuva de idéias, que seria um periodo de tempo, normalmente curto, quando os desenvolvedores anotam todas as idéias tidas pelo grupo, sem desprezar nenhuma, mesmo aquelas que parecem absurdas inicialmente. Depois do Brain Storm vem a seleção da idéia que será levada em frente e a primeira etapa da lista acima está completa.

A segunda etapa consciste na descrição completa e detalhada de como será o jogo em termos de funcionamento, interface e tudo o mais que for necessário para que o jogo esteja completamente definido. Pode-se incluir aqui uma definição do enredo, caso se trate de um jogo no qual a história é um ponto relevante.

As etapas 3 e 4 conscistem, obviamente, na criação dos recursos multimídia que seram usados no jogo. A etapa 5, de análise, normalmente é realizada por analistas ou programadores. Enquanto a descrição formal do jogo (etapa 2) descreve o jogo do ponto de vista do usuário, a análise é a etapa em que essa descrição é passada para o ponto de vista do desenvolvedor, ou seja, é feita uma nova descrição do jogo,

mas dessa vez usando-se uma linguagem matemática mais formal, que não deixe dúvidas com relação ao funcionamento do jogo. É comum o uso de linguagens de alto nível como Fusion ou UML para fazer essa descrição. Essas linguagens normalmente podem também ser usadas na etapa seguinte, de projeto.

A etapa de projeto parte da análise para descrever detalhes de como o jogo será implementado. Detalhes como algorítmos a serem usados na lógica do jogo e como os dados serão seram armazenados na memória são discutidos nessa etapa. Uma vez que isso esteja bem definido, entramos na sétima etapa, de implementação, que é quando o jogo realmente é programado e quando todas as partes são reunidas. Finalmente, os erros provenientes dessa etapa são encontrados e corrigidos na etapa de depuração e testes.

A primeira coisa que deve ficar clara é que, em alguns casos, cabe aos desenvolvedores definir que etapas viram antes ou depois, ou mesmo quais tarefas viram em paralelo. Conforme veremos mais adiante no curso, convém a utilização de fluxogramas que essas definições sejam feitas formalmente e de cronogramas, para que além de definir a ordem das tarefas também se tenha idéia de prazo.

As duas primeiras etapas são as mais importantes, pois dá muito trabalho para mudar algumas coisa definida nessas duas primeiras etapas quando o desenvolvimento já se encontra em etapas posteriores. Isso acontece porque uma etapa depende da outra, a análise será feita em cima da descrição, o projeto em cima da análise e a implementação em cima do projeto, mudar a descrição implica em mudar tudo isso.

No caso de recursos gráficos e sonoros, imaginem como seria alto o custo se 1000 imagens fossem desenvolvidas, junto com efeitos e trilha sonora para um jogo de guerra entre dois planetas e, na etapa de projeto, a definição fosse mudada para um jogo de guerra medieval.

Normalmente, as etapas 3 e 4 podem ser feitas em paralelo com as etapas 5, 6, 7 e 8, até porque normalmente envolvem tipos diferentes de profissionais. As duas primeiras etapas sempre devem vir antes e devem ter o maior detalhamento possível, em qualquer caso.

Tipos de jogos

Um jogador atento de jogos eletrônicos deve notar, mesmo que não seja um desenvolvedor, que os jogos costumam apresentar diversas semelhanças entre si. Essa semelhança não é coincidência. O custo

de desenvolvimento de um jogo é bastante alto. Portanto, muitas empresas de desenvolvimento de jogos costumam investir em máquinas de jogos (engines), ao invés de investir diretamente nos jogos propriamente ditos.

É fácil observar que há uma incrível semelhança entre jogos de tabuleiro como damas ou xadrês, entre jogos de cartas, jogos 3D em primeira pessoa, jogos de plataforma onde normalmente há um ou dois personagens que caminham lateralmente em um cenário 2D apresentado em uma tela, jogos de estratégia em tempo real, etc.

Devido a essa grande semelhança, os jogos são normalmente diferenciados pelo seu tipo. O conhecimento sobre os tipos de jogos existentes, contudo, é mais uma questão de mercado e de conhecimentos sobre atualidades que de conceitos. O importante é ter em mente que os jogos normalmente tem semelhanças e que, algumas vezes, o trabalho gasto em um jogo pode ser reaproveitado em outro.

Como exemplo, podemos citar um jogo de estratégia em tempo real 2D, onde o usuário tem uma visão em uma perspectiva fixa do cenário, e um jogo de estratégia em tempo real em 3D, com outra história, outros gráficos, etc. Isso chega a ser obvio algumas vezes, mas é importante ser formalizado.

Profissionais que participam da criação de um jogo

Em alguns jogos eletrônicos, sobretudo aqueles que faziam sucesso antigamente, na época do seu surgimento, não precisam de muitos profissionais para serem criados. Atualmente existem até ferramentas de criação de jogos que, por serem tão fáceis, chegar a se parecer com um jogo, embora cheias de limitações.

Os jogos atuais, orientados a performance, contudo, não parecem seguir o mesmo esquema. Hoje jogos são encarados como um novo tipo de mídia, assim como a TV, revistas em quadrinhos, web sites, etc. Esses jogos normalmente costumam envolver profissionais de várias áreas diferentes, incluindo aí programadores, analistas, profissionais da área de propaganda, artistas gráficos como desenhistas e modeladores, músicos e outros profissionais que exercem funções semelhantes, como arquitetos, profissionais da área de filosofia, letras, história, ciências humanas e geral, etc.

Dois tipos de profissionais costumam estar presentes no desenvolvimento de qualquer tipo de jogo, contudo, que são programadores e artistas gráficos. É muito importante que esses os desenvolvedores

de um jogo tenham uma boa integração entre si, em especial esses dois. Veremos nos capítulos seguintes detalhes sobre a integração no trabalho de artistas gráficos e programadores e discussões sobre quais devem ser as responsabilidades de cada um.

Também deve ficar bem claro que programadores e outros profissionais da área de exatas envolvidos devem ter a mente aberta o suficiente para se envolver não só em questões técnicas, mas também em várias questões relativas a outras áreas.

Formas de criar um jogo (linguagens de programação, ferramentas prontas)

Muitas pessoas se iniciam no aprendizado de linguagens de programação por um motivo bem definido: criar jogos. De fato, quando falamos de desenvolvimento de jogos eletrônicos, mesmo que não sejam jogos para computadores pessoais, normalmente estamos falando de jogos que são programados em alguma linguagem ou que foram feitos em um ferramenta programada em alguma linguagem.

A surpresa para algumas dessas pessoas ocorrem quando descobrem que, mesmo depois de terem aprendido a programar alguma linguagem ou até mesmo de já terem um conhecimento razoável sobre programação, ainda não sabem ou não tem idéia de como proceder para desenvolver jogos, principalmente se estivermos falando de jogos orientados à performance.

O que se deve ter em mente, portanto, quando se pensa nas formas de criar jogos, é que o conhecimento mais necessário e importante não é uma linguagem de programação em si ou o fato de saber como fazer isso ou aquilo, o importante é o conceito. Uma pessoa que aprende bem os conceitos por detrás da área de desenvolvimento de jogos e os conceitos de computação em si (não só de programação, mas sobre como o computador e outros dispositivos semelhantes funcionam por dentro) será capaz de criar jogos com qualidade melhor e de forma otimizada além de ser capaz de projetar os seus próprios jogos.

Isso é o que parece, inclusive, fazer mais falta no mercado de desenvolvedores de jogos atualmente. Existem muito programadores competentes que são capazes de implementar jogos projetados por outros, mas quais e quantos desses programadores são capazes de projetar jogos eles mesmos ?

São esses tipos de conceitos que veremos da próxima parte do curso em diante. Em outras palavras, agora é que começaremos a dirigir

nossa atenção para o conhecimento mais técnico. Por hora, é preciso saber algumas poucas coisas de como jogos podem ser desenvolvidos. Deve ficar bem claro que não necessariamente os jogos precisam ser desenvolvidos com essas ferramentas, maneiras alternativas podem ser usadas, mas o que será mostrado são os tipos de ferramentas que de forma mais comum costumam ser utilizadas.

Na seção anterior, mencionamos tipos de jogos existentes. Para alguns tipos de jogos de sucesso, como jogos de RPG, jogos em 3D, em primeira pessoa, de corrida, etc., existem ferramentas prontas que permitem que o usuário crie jogos mesmo sem ter muita noção de como isso está acontecendo internamente. Exemplos de ferramentas desse tipo são o RPGMaker, Mugen, RPGMaker, Macromedia Director, MultimediaFusion, Blender, DarkBasic, etc. Procure por essas ferramentas em um site de busca ou no portal MPage para maiores informações.

Essas ferramentas normalmente fornecem um interface onde o usuário não precisa se preocupar com programação para criar seus jogos. Algumas delas, porém, como Blender e Director, permitem o uso de linguagens de programação de alto nível para jogos um pouco mais complicados (blender permite uso de python e Director o uso de uma linguagem própria, LINGO). Deve-se destacar que nem sempre essas ferramentas são específicas para jogos, Director costuma também ser usado para criação de apresentações multimídia e blender para criação de imagens e animações 3D.

Quando desenvolvendo jogos orientados a performance, o programador costuma utilizar outras linguagens que permitem um acesso mais rápido aos recursos do dispositivos sendo utilizados (como, no caso do computador, placa de vídeo, som, memória) além de possibilitarem lógicas normalmente mais complicadas.

O fato é que linguagens e ferramentas prontas normalmente impõe limites com relação ao jogo a ser criado e também impõe restrições no quesito velocidade. Quando se deseja criar um jogo mais elaborado, orientado a performance, qualidade e normalmente inovação, costuma-se utilizar liinguagens como C, C++, Objective C, Pascal, etc. Quando se deseja essa características mas há problemas com relação a escrever muitos jogos, costuma-se utilizar bibliotecas de programação prontas nessas linguagens (como crystal 3D, Clanlib, SDL, etc.). Quando o importante é que o jogo seja feito no menor prazo possível e limitações não são um problema tão grande, então o uso de ferramentas prontas será suficiente.

Capítulo 3

Noções sobre recursos gráficos

Agora que já tivemos uma introdução teórica não técnica sobre como jogos são desenvolvidos, começaremos a pensar mais na parte técnica. Nesse capítulo, serão ensinados os conceitos primordiais necessários para utilização de gráficos em jogos.

A primeira parte do capítulo tratará ainda de algumas informações não técnicas como relação entre artistas gráficos e programadores quando desenvolvendo um jogo. A segunda parte já dará alguns conceitos sobre como funciona a parte gráfica de um computador, que é muito similar a de outros dispositivos, e sobre conceitos básicos como pixels, sprites, etc.

No final do capítulo é proposto um exercício prático que é parte do projeto final realizado ao longo do curso, que é a criação de um jogo orientado a performance simples mas completo. No exercício desse capítulo, será proposto que imagens sejam criadas no Blender, programa 3D da NotANumber. Para fazer o exercício desse capítulo, é assumido que o leitor já tenha alguns conceitos básicos sobre utilização do blender. Para obter esses conceitos, acesse o portal MPage, onde se encontram tutoriais que ensinam a utilizar o básico sobre o blender.

3.1 Como gráficos podem ser usados em jogos

Não é preciso explicar que gráficos podem ser utilizados em jogos, afinal, a grande maioria dos jogos famosos existentes hoje se diferencia pelos gráficos bem feitos. Como utilizar esses gráficos e como criá-los é que é a questão a ser discutida aqui.

A primeira coisa a ser aprendida é como são guardados os gráficos na memória do computador. Quando fazemos um desenho em uma folha de papel, dizemos que esse desenho está guardado no papel de forma analógica, pois chega a ser inviável tentarmos olhar para um ponto no papel e dizer qual é a cor daquele único ponto. Qual é o tamanho do ponto? Qual exatamente é aquela cor? Quando desenhamos em um papel, tudo isso é muito impreciso.

Quando falamos de gráficos digitais ou digitalizados, contudo, tudo isso está muito bem definido. Cada imagem no computador deve ser encarada como um conjunto de pontos, dispostos em uma forma retângular (por exemplo o próprio monitor de vídeo), onde cada ponto corresponde a uma cor bem definida.

Como é isso? Imagine que sua placa de vídeo esteja configurada para uma resolução de 800x600 pixels. Isso significa que quando você olha para o monitor de vídeo você vê é como um retângulo que exibe 800 pontos na largura por 600 na altura. No total, estão sendo apresentados 800 vezes 600 = 480000 pontos.

Cada ponto, por sua vez, é um número que corresponde a uma cor. Além da resolução de pontos, verifique a configuração de sua placa de vídeo e perceba que além da resolução de pontos, há uma resolução de cores. A resolução de cores costuma ser referenciada como sendo de 8, 16, 24 ou 32 bits. O que significa isso?

Como qualquer programador deve saber, sempre que armazenamos um número na memória do conputador estamos usando uma quantidade de memória para armazenar esse número. Quando usamos um byte (8bits) de memória, podemos armazenar um número que vai de 0 a 255. Se quisermos armazenar um número mais alto, como 1000, temos de alocar mais memória. Com dois bytes (16bits) podemo armazenar dois números que vão de 0 a 255 ou podemos armazenar um número que vai de 0 à 65536 ou ainda três números que vão de 0 a 32 e outro que vai de 0 a 1.

A explicação para esse fato vem da matemática. Cada bit pode assumir o valor 0 ou o valor 1. Se temos 8 bits, temos 2 elevado a 8 = 256 combinações possíveis.

Quando falamos então que a resolução de cores é de 8bits, dizemos que estamos usando apenas um número que vai de 0 a 255 para armazenar essa cor. Essa forma de armazenar as cores e de armazenar os gráficos na memória terá fundamental importância quando fizermos programas em capítulos posteriores que exibam gráficos no vídeo.

Vamos pensar agora no seguinte: como cada número desses pode armazenar uma cor? Qual a relação que existe entre uma cor e um número? Sabe-se da física que qualquer cor que o olho humano consegue captar corresponde à uma parcela de vermelho, outra de verde e outra de azul. Toda cor que conseguimos enxergar pode ser formada apenas a partir dessas três.

Em inglês, vermelho é red, verde é green e azul é blue. É por esse motivo que costumamos nos referênciar a cores que são descritas por uma intencidade de vermelho, uma intencidade de verde e outra de azul por CORES RGB (Red Green Blue). Quando usamos um número para armazenar uma cor, portanto, devemos de alguma forma guardar a informação sobre a intencidade de cada uma dessas 3 cores. Veremos agora como isso é feito nos computadores em cada resolução de cores. Tenha em mente que o importante é aprender os conceitos por enquanto, veremos implementações em formas de programas com a libSDL depois.

Uma nota importante deve ser feita: alguns dispositivos, principalmente os antigos, usam resoluções menores que 8 bits para guardar a cor de um ponto. Essas resoluções já cairam em desuso e não serã tratadas aqui.

3.1.1 8 Bits - 256 cores - imagens baseadas em palette

Usando 8 bits para armezenar a cor, quais seriam as formas possíveis de armazenarmos cada intencidade? Poderiamos usar 2 bits (número de 0 até 3) para cada cor (sobrariam 2), mas poderiamos representar pouquíssimas cores com isso. A idéia é que simultâneamente sejam exibidas 256 cores diferentes no vídeo, mas cada uma dessas pode ser qualquer cor em termos de intencidade RGB. Como é feito então?

É usada uma palette de cores, que armazena as cores RGB para cada cor das 256 possíveis. Referenciamos então cada uma dessas 256 cores por um índice (que vai de 0 até 255), sendo que esse índice pode ser representado por um byte. Para cada índice, porém, são usados três bytes, um para guardar o vermelho, outro para o verde e outro para o azul. É novamente usado um número de 0 a 255 para armazenar a informação sobre a intencidade de cada uma dessas 3 cores.

Quando o computador opera no modo de 8 bits, portanto, o programa que acessa o gráfico deverá informar ao chip de vídeo uma localização na memória onde se encontra essa palette. Além disso, toda imagem que estiver armazenada na memória convencional ou na memória de

vídeo deverá representar cada ponto por um byte, que contém um número de 8 bits que corresponde a um índice nessa palette.

3.1.2 16 bits - 65.536 cores

Quando usamos 16 bits de cor, não há necessidade de palette. Usamos normalmente apenas os 15 primeiros bits e guardamos a intencidade de cada cor RGB de 5 em 5 bits. Como 2 elevado a 5 é 32, então usamos um número de 0 até 31 para armazenarmos a intencidade de cada cor. Perceba que embora possamos mostrar simultâneamente mais que 256 cores ao mesmo tempo na imagem, há uma quantidade menor de cores disponíveis.

3.1.3 24 bits - 16.777.216 cores

Quando usamos 24 bits para cada cor, temos 3 bytes, um para cada intensidade de vermelho, verde e azul. A utilização desse mode fica bastante simples, pois na hora de programar podemos usar três variáveis do tipo char (em C) e atribuir a elas um valor de 0 a 255, simplesmente.

3.1.4 32 bits - 16.777.216 cores + transparência

Quando o formato de 32 bits é utilizado, as intensidades das cores RGB são guardadas nos primeiro 24 bits, como na resolução de 24 bits. Os últimos 8 bits guardam informação sobre a transparência da cor. Usar imagens que suportam transparência normalmente é um pouco mais lento mas poupa muito trabalho de implementação, além de ser uma ferramenta extremamente poderosa.

Esses último 8 bits, que correspondem a um número de 0 a 255, costumam ser chamados de canal Alpha da cor. Sempre que ouvir essa expressão, lembre-se que estão falando sobre a transparência.

Finalizando essa seção, deve ser dito que dado o formato como essas imagens são armazenadas, podemos utilizá-las de várias formas em jogos. Existem jogos em 3D e jogos em 2D. Em jogos 3D, costuma-se utilizar imagens na forma de texturas, representando pele, grama, rostos, etc. Em jogos 2D, as imagens são copiadas para a tela (que na verdade também é uma imagem de um dos tipos acima) uma em cima da outra, e o resultado final é o que vemos em vários jogos. Tudo isso ficará bem mais claro no decorrer do curso.

3.2 Como criar gráficos a serem utilizados em jogos (que ferramentas usar)

São inúmeras as ferramentas que podem ser utilizadas para criarmos imagens, muitas com licenças comerciais, outras grátis, uma para um sistema, outras para outro, mas os conceitos aprendidos em uma normalmente não são perdidos quando passamos a usar outra. Além de ferramentas para a criação de imagens, também existem ferramentas para tratamento de imagens. Algumas mais conhecidas serão citadas no decorrer do texto.

A primeira ferramenta que é recomendada para acompanhar esse curso é o Blender. Veja o portal da MPage para obter informações sobre como utilizar esse programa, pois o conhecimento sobre como usar uma ferramenta assim é muito útil para quem quer desenvolver jogos. Para muitos exercícios no livro talvez sejam necessárias imagens criadas com o blender.

Como usar cada uma das ferramentas é assunto para vários livros e não será discutido com detalhes aqui. Mas é extremamente recomendado que o leitor conheça as seguintes ferramentas:

1. Image Magick `http://imagemagick.sf.net`
2. GIMP - GNU Image Manipulation Program `http://www.gimp.org`

São ferramentas muito uteis no tratamento e manipulação de imagens. Image Magick é um conjunto de aplicativos para manipulação de imagens e conversão de formatos de arquivo. GIMP é o programa da GNU para tratamento de imagens e aplicação de efeitos. Em seu website há bastante documentação disponível.

3.3 Trabalho entre artistas gráficos e programadores

Os dois tipos de profissionais mais necessários no desenvolvimento de um jogo talvez sejam artistas e programadores. Uma boa integração entre ambos os profissionais é essencial para que o desenvolvimento seja bem sucedido. Veremos quais são as responsabilidades básicas de cada um nesse desenvolvimento e discutiremos um pouco sobre a semelhança entre jogos e outros tipos de mídia onde artistas estão mais presentes que programadores.

3.3.1 Quais são as responsabilidades dos artistas

Artistas gráficos são normalmente responsáveis pela criação das imagens a serem utilizadas no jogo. As imagens podem ser desenhadas a

mão e digitalizadas por meio de um scanner, equipamento próprio para a função, ou modeladas em 3D usando-se ferramentas como blender, 3D Studio, Maya, K3D, etc. Em raros casos, são usados programas vetoriais (como sketch, KIllustrator e Corel Draw) ou de desenho 2D (PaintBrush, xpaint, kpaint) para criar as imagens.

Uma vez criadas as imagens, costuma-se usar programas de tratamento de imagens como GIMP, Photoshop ou mesmo ImageMagick para que as imagens sejam tratadas ou efeitos sejam aplicados. Artistas gráficos normalmente são também responsáveis por isso.

Finalmente, artistas gráficos são responsáveis por criar os arquivos contendo as imagens e animações necessárias, no formato correto e na resolução correta a ser lida pelo jogo. Outros tipos de artistas responsáveis por efeitos sonoros, música, interface, etc normalmente seguem responsabilidades muito semelhantes.

3.3.2 Quais são as responsabilidades dos programadores

Programadores são responsáveis pela etapa de implementação do jogo. Com relação aos artistas os programadores devem definir exatamente quais serão os formatos gráficos (no caso de artistas gráficos, claro) a serem utilizados no jogo, quantas e quais imagens serão necessárias e como essas imagens devem ser fornecidas aos programadores.

É importante que programadores estejam sempre em contato com artistas e responsáveis pela parte de enredo do jogo de forma a estarem sempre cientes de qual infra-estrutura é necessária e de que modo pode-se prover maior liberdade para quem decide como será o jogo. O programador faz o papel da infra-estrutura, artistas cuidam da beleza e os responsáveis pelo enredo e por como será o jogo cuidam do conteúdo. Um bom diálogo entre todos os tipos de profissionais ajuda a deixar claro para o grupo qual será o jogo feito, como será esse jogo e como poderia ser tal jogo.

3.3.3 A semelhança entre criação de jogos e outros tipos de mídia

De fato, jogos hoje tem uma participação massiva no estilo de vida das pessoas. Quando falamos de mídia, devemos ter ciência de que estamos falando de algo que tornou o mundo naquilo que conhecemos hoje. Cada vez mais fica claro para o ser humano que o mundo é enorme no quesito informação, mas cada vez mais está dificil para as pessoas saber como encontrar a informação que interessa.

Jogos, assim como quadrinhos, cinema ou mesmo novelas e seriados não costumam conter somente uma história, mas também muita informação. O que o autor desse texto espera que fique claro para quem está lendo é que desenvolver jogos não é só técnica, é também uma responsabilidade muito grande. Se um dia você, o leitor, desenvolver um jogo onde indiretamente é passada a informação de que a coisa mais importante do mundo é lutar contra alienígenas, pode parecer incrível mas possivelmente grupos anti-alienígenas surgiram no mundo se seu jogo fizer sucesso.

Você, o desenvolvedor, é quem decide como será o jogo. Você acata parte da responsabilidade das consequências que seu jogo trouxer, caso venha a trazer alguma.

3.4 Noções sobre formatos gráficos

JPEG, TGA, BMP, PCX, GIF, PNG, DXF, 3DS, BLEND, TIF, XPM, são apenas algumas das várias extenxões de arquivos gráficos. Cada extenxão corresponde a um formato no qual as imagens (conjuntos de pontos onde cada ponto corresponde a uma cor) são salvas. Para cada formato, pode-se ainda ter variações no que diz a resolução de pontos, de cores, qualidade da imagem, transparência, etc.

É interessante para o desenvolvedor ter conhecimento sobre os formatos de arquivos gráficos mais comuns, para que possa ter parâmetros na hora de decidir qual formato usar. Segue explicação de alguns conceitos importantes sobre formatos e uma descrição sucinta de alguns tipos de arquivos.

3.4.1 Conceitos importantes

- Transparência absoluta e canais ALPHA

Formatos gráficos que suportam transparência podem suportar transparência absoluta ou canal ALPHA. Quando o canal ALPHA é suportado, normalmente o gráfico é mais demoradamente exibido, mas o formato descreve (conforme já dito acima) uma porcentagem de transparência para cada pixel (ponto gráfico) da imagem. Quando é suportada transparência absoluta, alguns pixels são marcados como transparentes ou não, sem nível de transparência. Essa forma de obter transparência nas imagens costuma ser bem mais rápida na hora de exibir a imagem, principalmente se for utilizado RLE.

- RLE - Run-Length Encoding

RLE é uma forma utilizada por vários formatos gráficos para diminuir o tamanho ocupado pela imagem na memória. Conciste basicamente em gravar os dados na memória de forma que fique escrito algo como "28 pixels pretos a partir da posição P" ao invés de escrever 28 vezes a cor preta.

- Gráficos vetoriais

Gráficos vetoriais não são como os gráficos que estudamos até agora, chamados de pixmaps, que são conjunto de pontos onde cada ponto corresponde a uma cor. Quando uma imagem é salvada em formato vetorial estamos dizendo que se uma circunferência é desenhada na imagem, não há a informação de quais pixels estão na cor da circunferência e quais não estão, a informação guardada é do tipo "existe uma circunferência de cor C na posição P da imagem com raio R". Esse é apenas um exemplo ilustrativo para mostrar a diferença entre os dois tipos de formatos.

3.4.2 Tipos comuns de arquivo

- JPEG, JPG

Formato de arquivo altamente utilizado na internet. A grande vantagem desse tipo de arquivo é seu tamanho, normalmente muito pequeno, mesmo para imagens de grandes resoluções. Esse formato também permite que uma imagem seja salva com perda de informações (perda de qualidade) para que seu tamanho fique menor. É possível salvar imagens em várias resoluções de cores, mas não suporta transparência. Demorado para carregar do disco.

- TGA - TARGA

Suporta várias resoluções de cores e transparência (ALPHA somente), além de ter uma compactação razoável. Também é um pouco demorado para carregar do disco.

- ICO - Windows Icon

Ícones do windows, resolução de pontos e cores altamente limitada mas pequeno e leve. Suporta transparência absoluta.

- BMP - Windows Bitmap

Formato gráfico padrão do windows. Sem qualquer compactação ou suporte para transparência, mas com suporte para várias resoluções de cores, (8, 16 ou 24). Os arquivos costumam ficar muito grandes nesse formato. BMP bem do nome bitmap ou mapa de bits, tem esse nome por motivos históricos, pois quando só existiam imagens preto e branco era usado um único bit para cada cor.

- XPM - X PixMap

Formato gráfico padrão do X Window System, sistema gráfico comum em *nix, incluindo linux. Mesmas características do BMP, mas suporta transparência absoluta. XPM vem do nome X PIXMAP, que significaria "Mapa de Pixels do X".

- GIF - Formato gráfico da Compuserve

Suporta somente imagens com resolução de cor de 8bits, baseadas em palette. Também suporta transparência absoluta. Esse formato é ainda muito usado na internet, mas pelo fato da Compuserve, criadora do algorítmo, ter começado a tentar cobrar pelos direitos de uso desse formato após o formato estar absolutamente popularizado na rede, muitos começaram a trocá-lo pelo formato .PNG.

- PNG - Resposta para o GIF

Sucessor do GIF, livre de cobranças relacionadas a sua utilização. Suporta tudo o que o GIF suporta e, de quebra, imagens com resolução maior que 8 bits e canais ALPHA ao invés de transparência absoluta.

- PCX - ZSoft PC Painbrush Format

Formato antigamente usado em jogos desenvolvidos para DOS. Esse formato foi criado pela ZSoft para seu produto PC PaintBrush. A utilização mais comum era com resoluções de 8 bits, mas existem versões do formato que suprtam mais cores. A única vantagem desse formato hoje é o fato de ser bem conhecido e fácil de se implementar sem uso de bibliotecas gráficas. Para algo mais profissional, não costuma mais ser usado.

- DXF - Formato para troca de dados

Formato usado para guardar gráficos vetoriais 2D e 3D. Programas do tipo CAD ou de modelamento costumam suportar esse formato.

- BLEND - Formato Blender

Formato de arquivo do programa Blender. Guarda várias informações dentro dele, não apenas uma imagem. Normalmente são guardados também modelos 3D, animações e, por vezes, até programas em linguagem python.

3.5 Sprites, o que são?

Sprites existem desde o início do desenvolvimento dos jogos eletrônicos. Quando criamos um jogos, existem determinadas entidades, representadas no video por imagens, que são controladas pelo jogador ou por uma lógica programada no computador. Essas entidades normalmente correspondem aos personagens presentes nos jogos e são chamadas de sprites. Grosso modo, toda entidade visual que obedece a uma lógica qualquer é chamada sprite. Esse talvez não tenha sido o sentido original da palavra, mas parece ser o mais adequado hoje em dia. Essa notação é mais comum em jogos 2D, embora também seja usada em jogos 3D.

As características de um sprite são simples de serem observadas, basta imaginarmos os jogos tradicionais. Quase sempre existem personagens nesses jogos, sendo que esses personagens tem ações que podem ser tomadas (no caso de um humano poderia ser andar para um lado, andar para outro, pular, etc) e cada ação é uma animação composta por várias imagens e algumas vezes som.

Por hora, é o que é necessário dizer sobre sprites, apenas para que o leitor não se perca com terminologia. Adiante, no curso, veremos exemplos práticos do uso de sprites, inclusive no projeto desse módulo, o jogo1.

3.6 Descrição para o jogo do projeto desse módulo - jogo 1

Segue abaixo a descrição formal (primeira etapa do desenvolvimento) do jogo projeto a ser desenvolvido durante o curso. É interessante prestar atenção nos detalhes para ir se acostumando. Todo desenvolvedor com um mínimo de profissionalismo faz uma descrição do tipo antes de qualquer outra coisa.

- Nome do jogo: Blocos
- Resolução do jogo: 640 x 480 x 32bits(ALPHA)

Note que o jogo poderá rodar em qualquer resolução na máquina da pessoa, inclusive 8bits, mas internamente no programa usaremos 32bits.

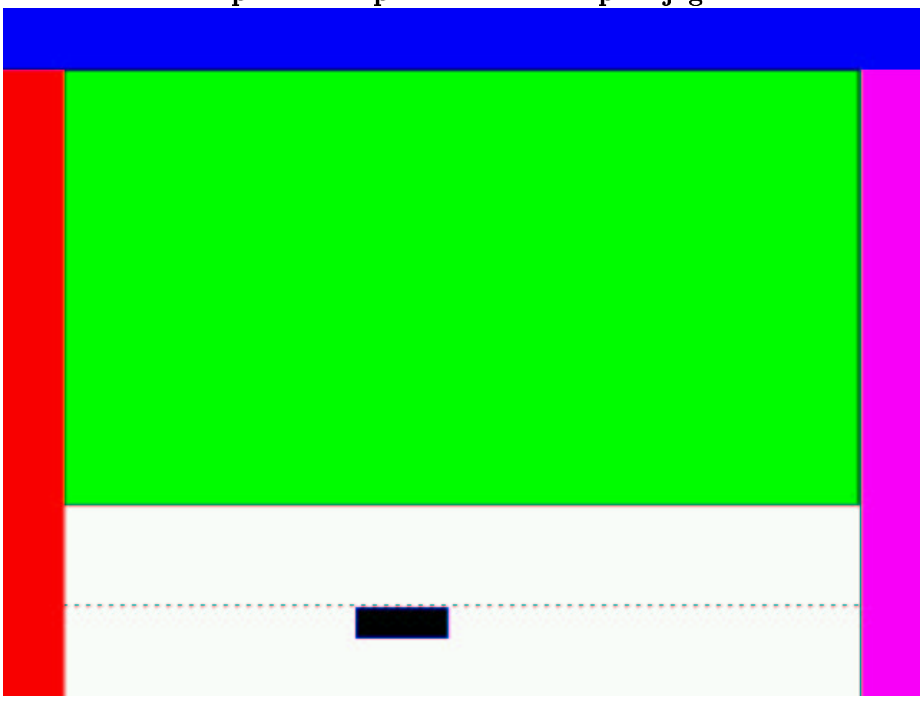
3.6.1 Descrição da interface e dos níveis:

Ao iniciar o jogo, o usuário vê um menu com as seguintes opções:

1. Escolher nível
2. Sair

Escolhendo a opção 1, ele vê um menu de todos os níveis disponíveis. Seleciona um com o teclado e tecla ENTER para começar a jogar aquele nível. Acabando o nível ele volta para o menu.

Cada nível corresponde a uma tela da resolução indicada acima dividida em quadrados de 20x20 pixels, ou seja, uma tela de 32x24 quadrados. A tela é dividida como demonstrado na FIGURA1. Na figura, as partes em vermelho e em azul servem para exibir número de vidas, pontos e outras coisas do gênero. A parte rosa é reservada e a parte verde é a parte onde ficam os blocos a serem acertados. O bloco preto é o sprite controlado pelo jogador.



A parte verde é a parte que muda de nível para nível. Essa parte é composta por 14x16 retângulos de 40x20 pixels, que correspondem aos blocos. As partes vermelha e rosa tem 24 quadrados de altura e 2 de largura. A parte azul tem 32 de largura e 2 de altura. O bloco preto tem tamanho inicial de 40x20, mas pode ter seu tamanho alterado no decorrer do jogo.

3.6.2 Funcionamento do jogo

O jogo começa com um nível carregado e com o bloco preto com a coordenada x no centro da tela. Uma bola fica em cima do bloco do jogador até que o jogador pressione a tecla de espaço. Uma vez pressionada, a bola é lançada para cima. Cada vez que a bola colide, seu movimento é alterado, de forma a parecer que um choque perfeitamente elástico ocorreu.

Cada vez que a bola colide em um bloco, o bloco sofre uma reação, que varia de acordo com o tipo de bloco e com o tipo de bola atual. As reações são descritas a seguir. O objetivo do jogador é quebrar todos os blocos que podem ser quebrados sem que a bola caia para baixo da linha tracejada. Para não deixar a bola cair, o jogador deve movimentar o bloco para os lados de forma que esse colida com a bola.

Quando um bloco quebra, por vezes deixa cair um bônus, representado por um retângulo de 40x20 pixels. Caso o jogador consiga fazer seu sprite colidir com o bônus, o bônus é acionado. Os tipos de bônus são descritos a seguir.

3.6.3 Tipos de blocos

- 0 - Bloco padrão, quando a colisão acontece ele é destruido.
- 1 - Quando a colisão acontece, se transforma no bloco 0.
- 2 - quando a colisão acontece, se transforma no bloco 1.
- 3 - Bloco indestrutível, não quebra nunca

3.6.4 Tipos de bola:

- 0 - Tipo padrão, muda seu movimento quando colide com os blocos.
- 1 - Indestrutível, quebra qualquer bloco destrutível mas não muda seu movimento ao colidir.

3.6.5 Tipos de bônus:

- 0 - Faz o usuário ganhar uma vida
- 1 - Muda tipo de bola para 0
- 2 - Muda tipo de bola para 1
- 3 - Adiciona uma nova bola ao jogo, ou seja, o jogador passa a ter uma bola a mais para quebrar os blocos.
- 4 - Altera o tamanho do sprite do jogador para 60x20 caso seja diferente, ou para 40x20 caso já tenha esse tamanho.
- 5 - Altera o tamanho do sprite do jogador para 30x20 caso seja diferente, ou para 40x20 caso já tenha esse tamanho.
- 6 - Adiciona barreira no chão ou retira a mesma, caso já exista. Não deixa a bola passar do chão, ou seja, o usuário não tem como perder enquanto o bônus está ativo.

3.6.6 Colisões e velocidades:

Cada bola tem uma velocidade em pixels para a coordenada X e Y. Essa velocidade vai de -MAX_VEL_X até MAX_VEL_X para x e -MAX_VEL_Y até MAX_VEL_Y para y. Sempre que ocorre uma colisão e o movimento muda, as velocidades são alteradas. Quando a bola colide com uma superfície plana, apenas uma das coordenadas é invertida, nada mais. Quando a bola bate em uma quina, as velocidades x e y são ajustadas de acordo com o ângulo que a bola faz com a quina.

3.6.7 Níveis:

O jogo procura os arquivos contendo os níveis em uma pasta (um diretório) fixa. Cada arquivo .blc na pasta corresponde ao nível. Para cada arquivo .blc, deve existir um arquivo .png (imagem) com o mesmo nome, correspondendo à imagem de fundo do nível. Por exemplo, se existe um arquivo nivel1.blc, deve existir outro nivel1.png. Caso não exista, uma imagem padrão é usada.

A imagem deve abranger toda a área correspondente à área verde da tela. O formato do arquivo .blc é apenas uma sequência de 16 linhas, cada uma contendo 14 números inteiros que correspondem ao tipo de bloco sendo utilizado. Cada inteiro corresponde a um bloco contido na área correspondente à área verde da figura. Caso o número seja negativo, isso significa que ná há bloco naquela posição.

3.7 Exercício 1: Criação e tratamento das imagens necessárias para o jogo1

Com a descrição do jogo feita, a primeira coisa é criarmos uma lista das imagens necessárias. Olhando para a descrição, é possível chegar à lista de imagens abaixo.

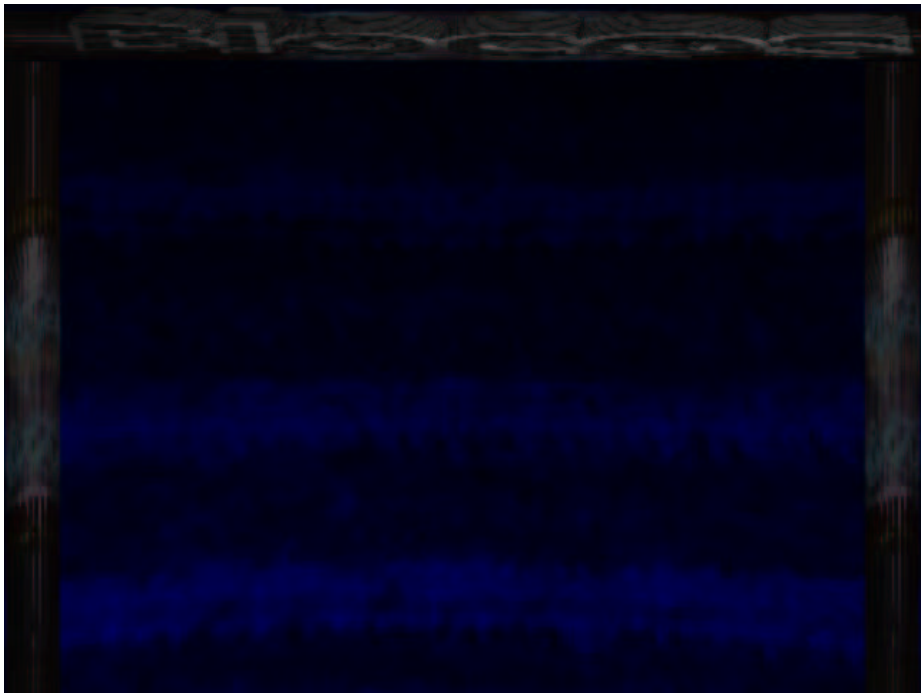
3.7.1 Lista de imagens

1. Fundo geral - Abrange a tela inteira, uma imagem de 640x480. Todo o resto é desenhado em cima dessa imagem.
2. Barreira no chão, não deixa a bola passar - 600x20
3. Sprite principal diminuido - 30x20
4. Sprite principal tamanho natural - 40x20
5. Sprite principal aumentado - 60x20
6. Bloco tipo 0 - 40x20
7. Bloco tipo 1 - 40x20
8. Bloco tipo 2 - 40x20
9. Bloco tipo 3 - 40x20
10. Bola tipo 0 -20x20
11. Bola tipo 1 - 20x20
12. Bônus tipo 0 - 40x20
13. Bônus tipo 1- 40x20
14. Bônus tipo 2- 40x20
15. Bônus tipo 3- 40x20
16. Bônus tipo 4- 40x20
17. Bônus tipo 5- 40x20
18. Bônus tipo 6- 40x20

Pronto. Agora basta criar e tratar as imagens.

3.7.2 Criação das imagens

A seguir, segue a lista de imagens criadas no blender. Sinta-se livre para usar as imagens abaixo em seu jogo, mas até pelo fato das imagens serem simples, recomendo usar o blender para criá-las. Acesse o portal para ter acesso aos tutoriais que ensinam como usar o blender.

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 

10. 

11. 

12. 

13. 

14. 

15. 

16. 

17. 

18. 

3.7.3 Tratamento das imagens

Uma vez que todas as imagens estejam criadas, precisamos tratá-las de forma que fiquem no formato na resolução de cores que queremos, e suportando transparência da maneira que decidirmos. No caso, decidimos que será melhor usar imagens no formato .png, resolução de cores de 32bits com transparência ALPHA.

Perceba que algumas das imagens acima são transparentes em alguns pontos. Talvez não seja possível ver que alguns pontos não são totalmente transparentes por causa do fundo desse documento e das capacidades do formato de arquivo usado para salvar esse documento sendo lido, mas no jogo isso se tornará mais claro.

Dado um arquivo criado no blender chamado imagem.tga, por exemplo, pode-se usar o programa convert, do pacote ImageMagick citado anteriormente, para transformar a imagem em um arquivo com essas características. Basta usar o seguinte comando:

- convert -verbose -type TrueColorMatte -transparent black imagem.tga imagem.png

Esse comando assume que toda cor preta da imagem original (arquivo imagem.tga) ficará transparente na imagem gerada (arquivo imagem.png). No caso de o arquivo original já ter sido feito com transparência, coisa comum de se fazer usando o blender, pode-se usar o seguinte comando para transformar a imagem em um .png preservando a transparência:

- convert -verbose imagem.tga imagem.png

Esse exemplo foi dado apenas para demonstrar levemente o poder do pacote ImageMagick. É um dos pacotes gráficos mais uteis para desenvolvedores de jogos. Contudo, quando esse texto foi escrito o ImageMagick só estava disponível para ambiente *nix, incluindo linux. Dessa forma, apresentaremos na próxima subseção como fazer a primeira conversão usando o GIMP, o programa de manipulação gráfica da GNU, disponível para linux e para windows.

Antes disso, porém, uma nota sobre a resolução escolhida. Para esse jogo, escolhemos usar imagens com ALPHA. Isso é algo muito requintado e permite que o designer que estiver fazendo as imagens use e abuse dos recursos de transparência para deixar o jogo muito bonito. Contúdo, é um formato que consome muitos recursos na hora de rodar o jogo, fazendo com que o jogador precise de uma boa máquina para rodar o jogo.

No nosso caso isso não fará muita diferença, pois desenharemos poucos sprites, como será discutido posteriormente, e também a tela não tem scroll, ou seja, o fundo não muda. Porém, mantenha esse fato em mente quando estiver desenvolvendo seus próprios jogos.

3.7.4 Usando o GIMP para tratar uma imagem

Capítulo 4

Uma breve revisão sobre programação em C

4.1 Condicionais

4.2 Loops

4.3 Estruturas (structs)

4.4 Funções

4.5 Outros conhecimentos necessários

4.6 Exercícios (esses exercícios deveram dar ao leitor a noção se ele tem ou não os conhecimentos necessários sobre programação para prosseguir o curso)

4.7 Indicação de referências e cursos sobre C

Capítulo 5

Programação

5.1 Apresentação das ferramentas usadas no restante do curso (cygwin ou DevC++ no windows ou linux, ambos com SDL)

5.1.1 Como instalar cygwin (windows) ou gcc (linux)

5.1.2 Como instalar SDL & cia

5.2 Por que usar essas ferramentas e não outras (explicar que quem deve ser competente é o desenvolvedor e não só a ferramenta)

5.3 Formas de criar um jogo - IDE Versus Makefile

5.4 Bibliotecas - descrição geral

5.5 Edição de código (XEmacs)

5.6 Makefile

5.7 DDD, Data Display Debugger (Para depurar código em linux)

5.8 Insight

5.9 Exercício (criando um programa simples em SDL que exibe um texto na tela, usando as ferramentas acima)

Capítulo 6

Um pouco de prática, programando brincadeiras em SDL

Capítulo 7

Usando sons

Capítulo 8

Introdução (leve) a orientação a objetos

8.1 Diferença entre um programa estruturado e um orientado a objeto

8.2 Orientação a objetos não está associada com a linguagem de programação

8.3 Orientação a objetos em C (a que usaremos no curso)

8.4 Classes e objetos

8.5 Outras noções (herança, templates, UML, etc. Não serão ensinados, mas a existência será citada)

8.6 Exemplo (O programa 6.6 feito em C orientado)

8.7 Exercícios (perguntas e respostas)

Capítulo 9

Projeto 1 (Finalmente)

9.1 Descrição do jogo

A primeira estapa do desenvolvimento é a criação da descrição formal do jogo. Essa descrição já foi feita na seção 3.6, antes de criamos as imagens necessárias para uso no jogo.

9.2 Fluxograma e cronograma de projeto

9.3 Análise e modelamento (como será o modelamento de classes do jogo, ou seja, orientação a objetos do jogo)

9.4 Aprontando os recursos necessários (no caso, apenas as imagens e alguns sons)

9.5 Implementação

9.6 Testes, Depurações e possíveis melhoramentos

9.7 O que vem a seguir (descrição dos próximos módulos)