Criando um jogo multiplataforma completo

Computer Graphics and Game Technology
CGGT - 20/04/2007

Alessandro Ribeiro da Silva

Conteúdo

- Apresentação
- O que é preciso para se criar um jogo
- Especificando o ambiente e ferramentas
- Metodologia de desenvolvimento
- Descrição do jogo: tetris
- Etapas de desenvolvimento
- Teste da versão beta
- Considerações finais

Apresentação

- CGGT - Computer Graphics and Game Technology group
  - Voltado para estudos nas áreas:
    - Computação Gráfica
    - Jogos
  - Temas ja abordados:
    - HDR, Técnicas procedurais, Renderização detalhada

Apresentação

- CGGT - Computer Graphics and Game Technology group
  - Esta é a segunda palestra do ciclo de palestras
    - As palestras foram colocadas em uma ordem evolutiva a fim de mostrar toda a abrangência do desenvolvimento de jogos até chegar em detalhes de como implementar os efeitos

Apresentação

- Participe do grupo através de nossa lista de discussão

https://listas.dcc.ufmg.br/mailman/listinfo/cggt

- Web-site

http://www.dcc.ufmg.br/projetos/cggt

O que é preciso para se criar um jogo

- Força de vontade ;-D
- Idéia simples
  - Para o primeiro jogo
- Tempo disponível
  - Duração deste projeto (+- 1 semana)
- Conhecimentos razoáveis:
  - Linguagem de programação
  - Lidar com diversas bibliotecas
  - Noção de estética (para GUIs)
  - Ter jogado jogos do mesmo estílo para ter um ponto de início

Especificando o ambiente e ferramentas

- Linguagem habitual
  - Lembrem-se de Murphy: Tenha certeza de que irá aparecer algum problema que somente sua experiência irá lhe ajudar a resolver
    - Exemplo: porque os números grandes quando em ponto-flutuante retornam o mesmo número em dados intervalos?

Especificando o ambiente e ferramentas

- Número inteiro:
  - 4294967200
  - 4294967295 = 2^32 - 1
- Ponto-flutuante (IEEE 32bits):
  - 4294967296 - acreditem: todo o intervalo inteiro acima resulta no mesmo número em ponto flutuante!!!
    - O número em PF possui um limite de dígitos consecutivos que são significativos
    - A representatividade de um número em ponto flutuante é tão limitada quanto a de um número inteiro.

Especificando o ambiente e ferramentas

- Linguagem habitual
  - Apesar de conhecer diversas linguagens e programar em C/C++, Java, C# e Python, minha linguagem habitual é C/C++

Especificando o ambiente e ferramentas

- Compilador
  - Qual escolher? Porque?
    - Visual Studio (Micro$oft)
    - C++ Builder (Borland)
    - GCC (GNU)
    - GCC ou XLC (IBM)

Especificando o ambiente e ferramentas

- Compilador
  - Quero que seja portável (windows/linux)
    - Visual Studio (XNA/DX) : pago
      - XBOX360
      - Windows
    - C++ Builder : pago
      - Windows
    - GCC : free opensource <<<<<< Nossa escolha
      - Plataformas (mac,linux,windows,...)

Especificando o ambiente e ferramentas

- GCC (GNU Compiler Collection)
  - Linux: costuma vir embutido nas principais distribuições (*Gentoo, Debian, Fedora, etc...)
  - Windows: www.mingw.org
    - Utilizar o mingw facilita o aproveitamento do mesmo makefile
    - Você pode montar seu kit de desenvolvimento pessoal para windows e leva-lo em seu pen-drive
  - Versão alvo 4.1.1 (final mais estável conhecida)

Especificando o ambiente e ferramentas

- Você estar fazendo do zero não quer dizer que tenha que reinventar a roda.
  - É possível se utilizar uma engine de jogo ou gráfica (ogre3d, irlitch, cristal space, etc...)
    - Não vemos o que está acontecendo
  - Leitores de arquivos podem ser feitos mas existem projetos muito bons:
    - libpjeg, libpng, oggVorbis, cairo, etc...

Especificando o ambiente e ferramentas

- APIs, bibliotecas e ferramentas
  - Não é necessário o domínio completo das ferramentas utilizadas
    - Manuais de referências servem para ser consultados

Especificando o ambiente e ferramentas

- APIs, bibliotecas e ferramentas
  - Controle de janelas e entrada de dispositivos
    - *GLUT, SDL_video
    - DXInput - somente entrada
  - Renderização e modelagem
    - D3D, *OpenGL
    - *Blender, 3DMax, Maya
  - Som(serão abordados na última palestra)
    - *OpenAL, SDL_sound, *oggVorbis
    - SoundForge, *Ardour, Vegas, *Muse, *Rosengarden, *Audacity, ...

Especificando o ambiente e ferramentas

- APIs, bibliotecas e ferramentas
  - Imagens (leitura e tratamento/criação)
    - *libpng, Devil, libjpeg, cairo, ...
    - *Gimp, InkScape, M$PaintBrush, Photoshop
  - Arquivo XML
    - *TinyXML, ...

Especificando o ambiente e ferramentas

- Modelo de processamento
  - Multi-Thread
    - Bom para tirar proveito de plataformas multi-core
    - Torna o projeto Complexo
      - Sincronização entre os processos/threads e controle de transferencias de estados de renderização e do jogo
  - Single-Thread
    - Simples

Especificando o ambiente e ferramentas

- Modelo de processamento (Single-Thread)
  - Tudo acontece no loop da aplicação

Metodologia de desenvolvimento

- Criação de camadas de abstração
  - Abstrações são amigas e te ajudam a deixar o código limpo

Metodologia de desenvolvimento

- Exemplo:
  - Criar textura em OpenGL (sem abstração)

```
glGenTextures(1,&texID);
glEnable(GL_TEXTURE_2D);
glTexEnvi(GL_TEXTURE_ENV,...);
glBindTexture(target,texID);
glTexParameteri(..., GL_TEXTURE_WRAP_S, ...);
glTexParameteri(..., GL_TEXTURE_WRAP_T, ...);
glTexParameteri(..., GL_TEXTURE_MAG_FILTER, ...);
glTexParameteri(..., GL_TEXTURE_MIN_FILTER, ...);
glTexImage2D(...);
```

Metodologia de desenvolvimento

- Exemplo:
  - Criar textura em OpenGL (com abstração)

```
TextureObject *tex = new TextureObject();
```

- Procure facilitar a vida!!!

Metodologia de desenvolvimento

- O programa como um todo pode ser visto como um conjunto de modulos
  - Ajudantes para renderização (GLHelpers)
  - Ajudantes de reprodução de som (ALHelpers)
  - Leitores de arquivos
  - Leitores de imagens
  - Etc...

Metodologia de desenvolvimento

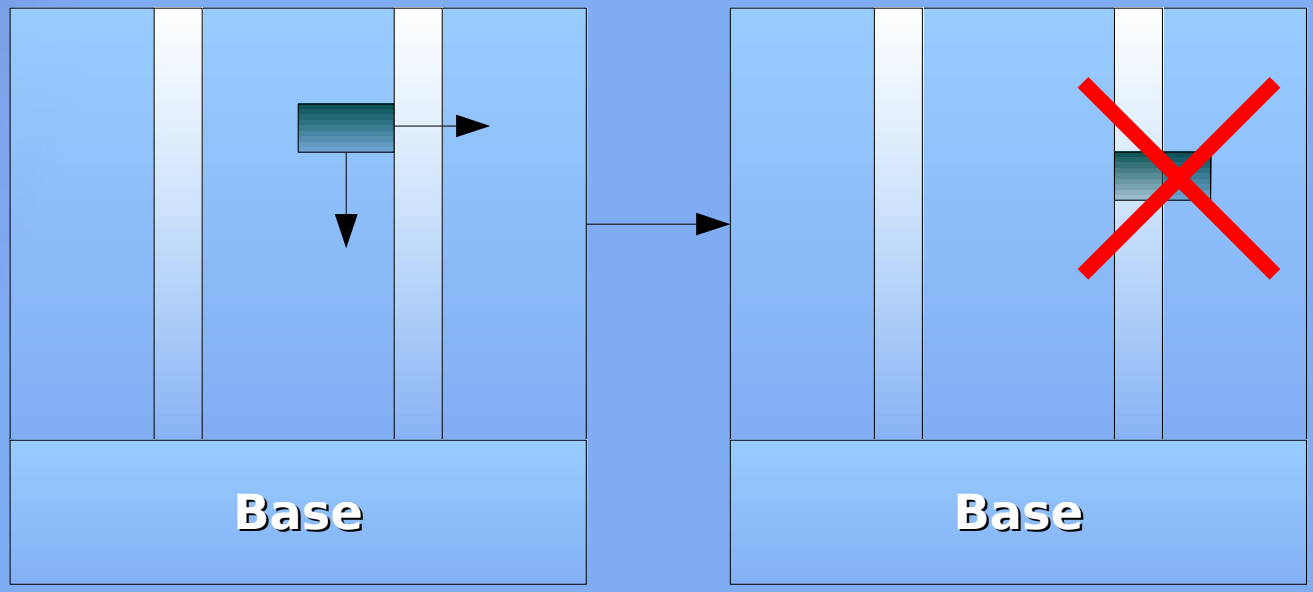
- Cada modulo antes de ser acoplado na aplicação passou pelos passos:
  - Implementação em projeto separado
  - Teste exaustivo
  - Se teste bem sucedido, então o módulo é utilizado

Descrição do jogo: tetris

- Jogo simples em lógica e gráficos
  - Sou programador e não modelador, qualquer extrapolação pode exigir tempo

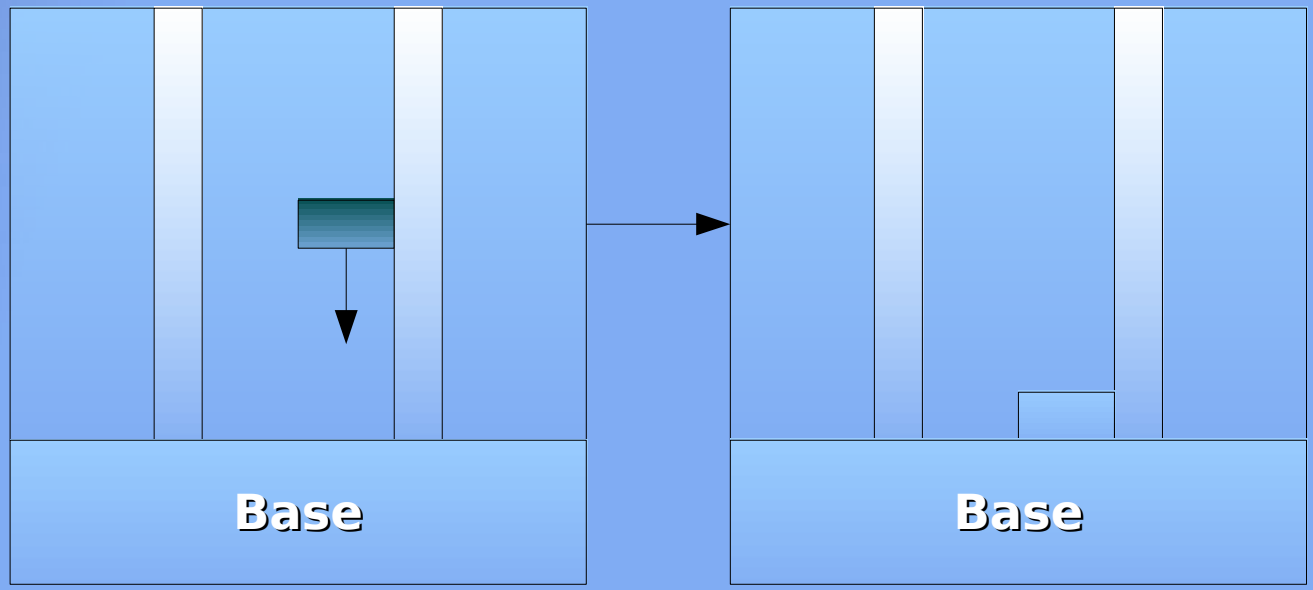
Descrição do jogo: tetris

- Lógica do jogo:
  - Peças descem
    - Não ultrapassam duas barreiras verticais



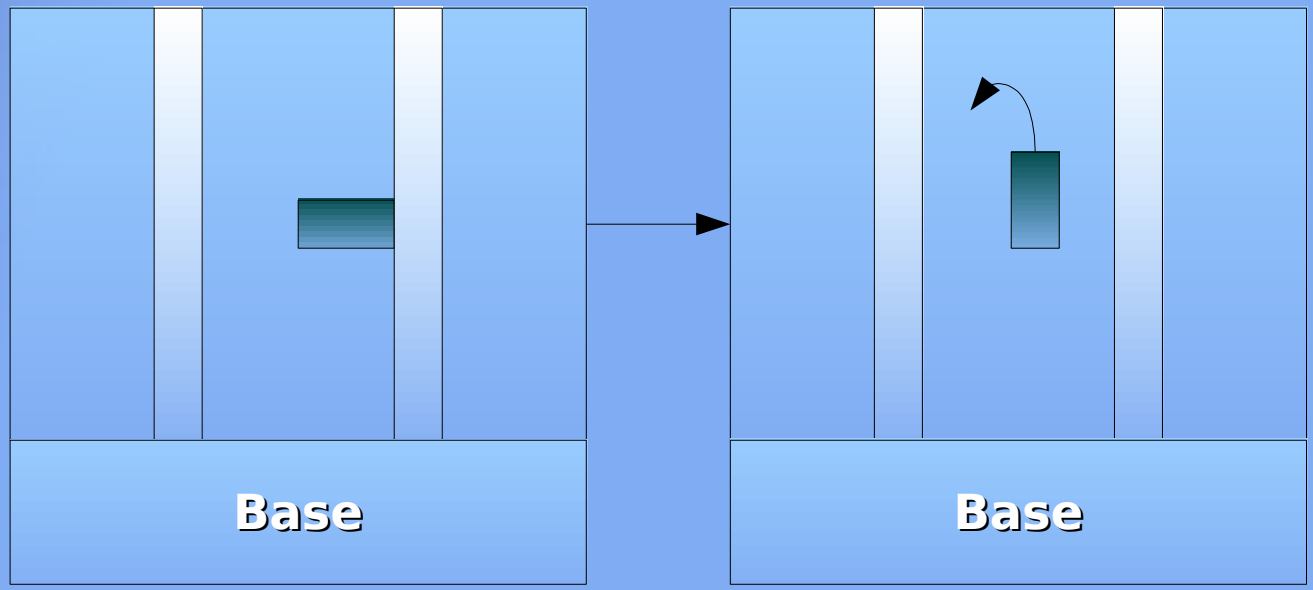
Descrição do jogo: tetris

- Lógica do jogo:
  - Peças descem
    - Ao encontar na base ele passa a fazer parte da base



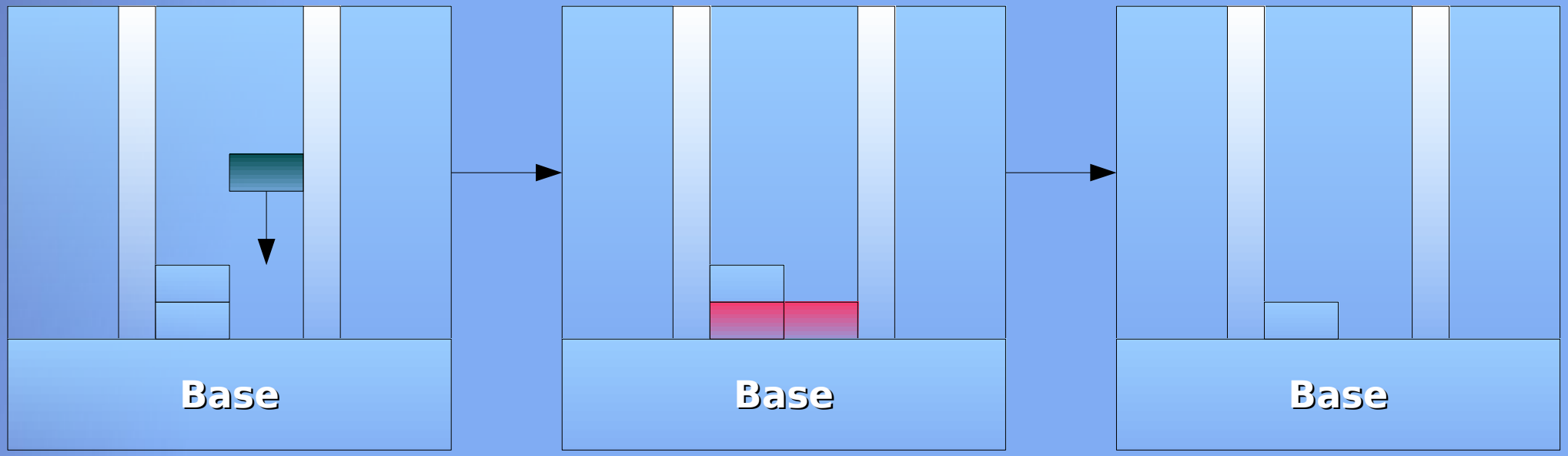
Descrição do jogo: tetris

- Lógica do jogo:
  - Peças podem ser rotacionadas
    - Aqui se considera somente um sentido



Descrição do jogo: tetris

- Lógica do jogo:
  - Existe um muro que vai crescendo
    - Ao se completar uma linha inteira, a mesma será removida acrescendo pontos ao jogador

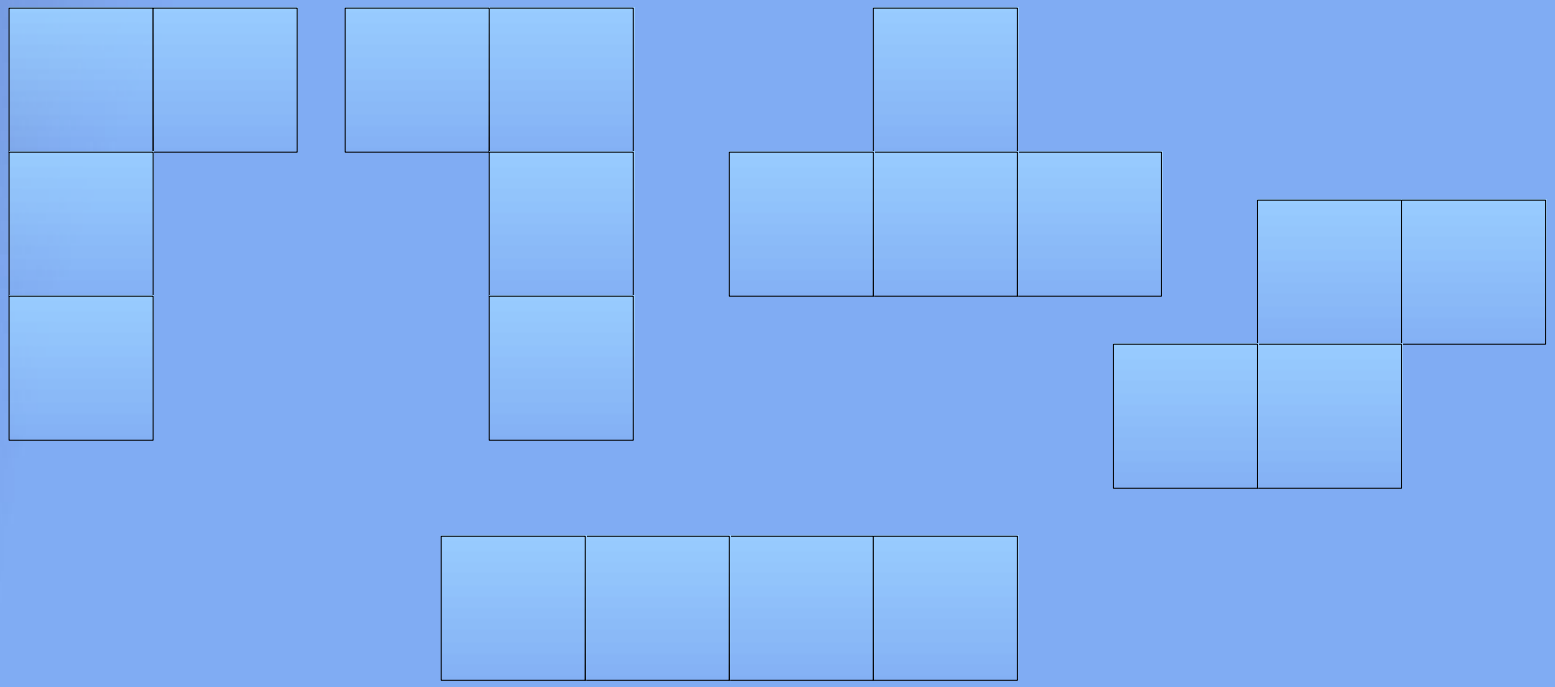


Descrição do jogo: tetris

- Lógica do jogo:
  - A mudança de nível é realizada a cada 10.000 pontos
    - O nível é representado pela velocidade com que a peça desce
    - A implementação que foi feita existem apenas 4 níveis para não deixar o jogo impossível de se jogar.
  - O jogo acaba quando não for possível colocar mais nenhuma peça.

Descrição do jogo: tetris

- Lógica do jogo:
  - Personagens
    - Configurações diversas compostas por 4 quadrados



Descrição do jogo: tetris

- Lógica do jogo:
  - IA
    - Não tem IA complicada para se implementar
      - No máximo existirá um gerador de números aleatórios que indicará a próxima peça para o jogador

Revisão

- Quer fazer um jogo?
- Sugestão de onde começar:
  - Escolha da plataforma alvo
  - Definição de ferramentas necessárias
  - Definição do jogo propriamente dito
- Mãos a Obra!!!!

Etapas de desenvolvimento

- Criação do controle da lógica
  - Teste parcial da lógica
- Sistema de entrada/saida
- Renderização de fontes
- Sistema de simulação
- Criação da GUI
- Som (tema da última palestra)
- Efeitos visuais
- Estados persistentes do jogo
- Possíveis melhorias

Etapas de desenvolvimento
Criação do controle da lógica

Etapas de desenvolvimento
Criação do controle da lógica

- Vamos começar a implementação do zero!!!!
- Ja sabemos como será o jogo, então podemos perguntar:
  - O que ele exige como entrada?
  - Quais as restrições de movimento?
  - Qual a máquina de estados?
    - Aqui é possível imaginar TODA a máquina de estados, pois o jogo é simples
    - O que não seria possível em um MMORPG ...extremo.. ;-)

Etapas de desenvolvimento
Criação do controle da lógica

- O que o jogo exige como entrada?
  - Apenas o controle do movimento da peça
    - Andar para esquerda
    - Andar para direita
    - Rotacionar
    - Cair mais rapido

Etapas de desenvolvimento
Criação do controle da lógica

- Quais as restrições de movimento?
  - Não se pode mover infinitamente para um lado
  - Uma peça somente pode ser rotacionada se houver um espaço que a contenha

Etapas de desenvolvimento
Criação do controle da lógica

- Qual a máquina de estados?
  - Uma tabela é o suficiente para armazenar o estado do muro
  - Como todas as peças apresentam o comportamento de deslocamento e rotação
    - Podem ser implementadas com herança

Etapas de desenvolvimento
Criação do controle da lógica

- Qual a máquina de estados?
  - tabela

```
#define altura 20
class TableMounter2D{
  bool table[altura][9];
  public:
  bool insert(UnitMounter2D *unit);
  TableMounter2D();
  bool collide(UnitMounter2D *unit);
  void consoleDraw();
  ...
};
```

Etapas de desenvolvimento
Criação do controle da lógica

- Qual a máquina de estados?
  - Classe pai de todas as peças

```
class UnitMounter2D{
  int posx,posy;
  vec2 positions[4];//quatro unidades por peça
  protected:
  int state;
  void setUnit(const int index, const vec2 &pos);
  public:
  virtual void rotate();//a classe filha implementa a rotação
  bool colide(const AABB &aabb);
  void consoleDraw();
  ...
};
```

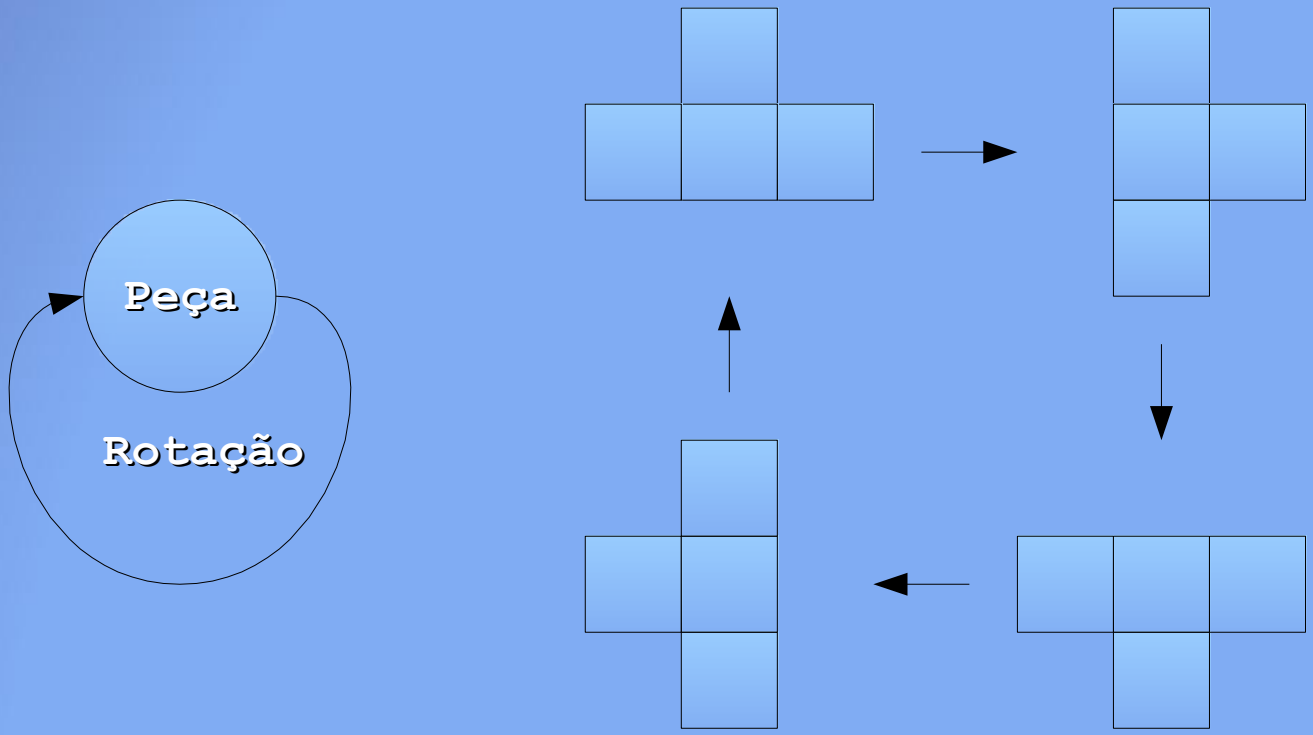
Etapas de desenvolvimento
Criação do controle da lógica

- Qual a máquina de estados?
  - Exemplo da implementação de uma rotação

```
...
  void Unit1::rotate(){
    switch(state){
     case 0:
      setUnit(0,vec2(0,0)); setUnit(1,vec2(1,0)); //####
      setUnit(2,vec2(2,0)); setUnit(3,vec2(3,0));
      state = 1;
     break;
     case 1:                                              // #
      setUnit(0,vec2(0, 0)); setUnit(1,vec2(0, 1)); // #
      setUnit(2,vec2(0, 2)); setUnit(3,vec2(0, 3)); // #
      state = 0;                                          // #
     break;
    }
  }
...
```

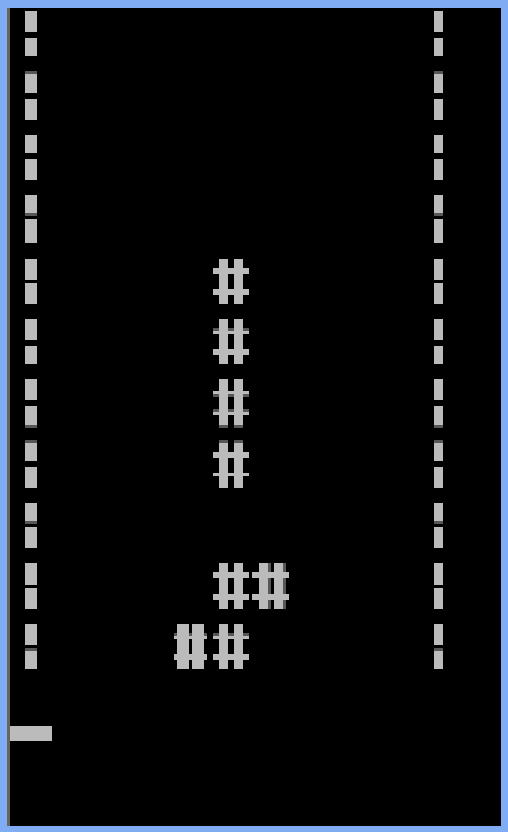
Etapas de desenvolvimento
Criação do controle da lógica

- Qual a máquina de estados?



Etapas de desenvolvimento
Criação do controle da lógica

- Depuração do comportamento de descida da peça em modo console
  - Vai encarar? !



Etapas de desenvolvimento
Sistema de entrada/saida

Etapas de desenvolvimento
Sistema de entrada/saida

- Coisas importantes para um jogo:
  - Entrada: mouse, teclado, joystick, pads, leitura de arquivos, etc...
  - Saida: video, som, rede, gravação de arquivos etc...

- Aqui será abordado o sistema de entrada/saida referentes ao teclado/mouse e saida de video.

Etapas de desenvolvimento
Sistema de entrada/saida

- Fazer um código de entrada/saida multiplataforma pode exigir muita codificação (WinAPI ou X11)
  - É aconselhado a utilizar uma API multiplataforma
    - *GLUT, SDL, etc...

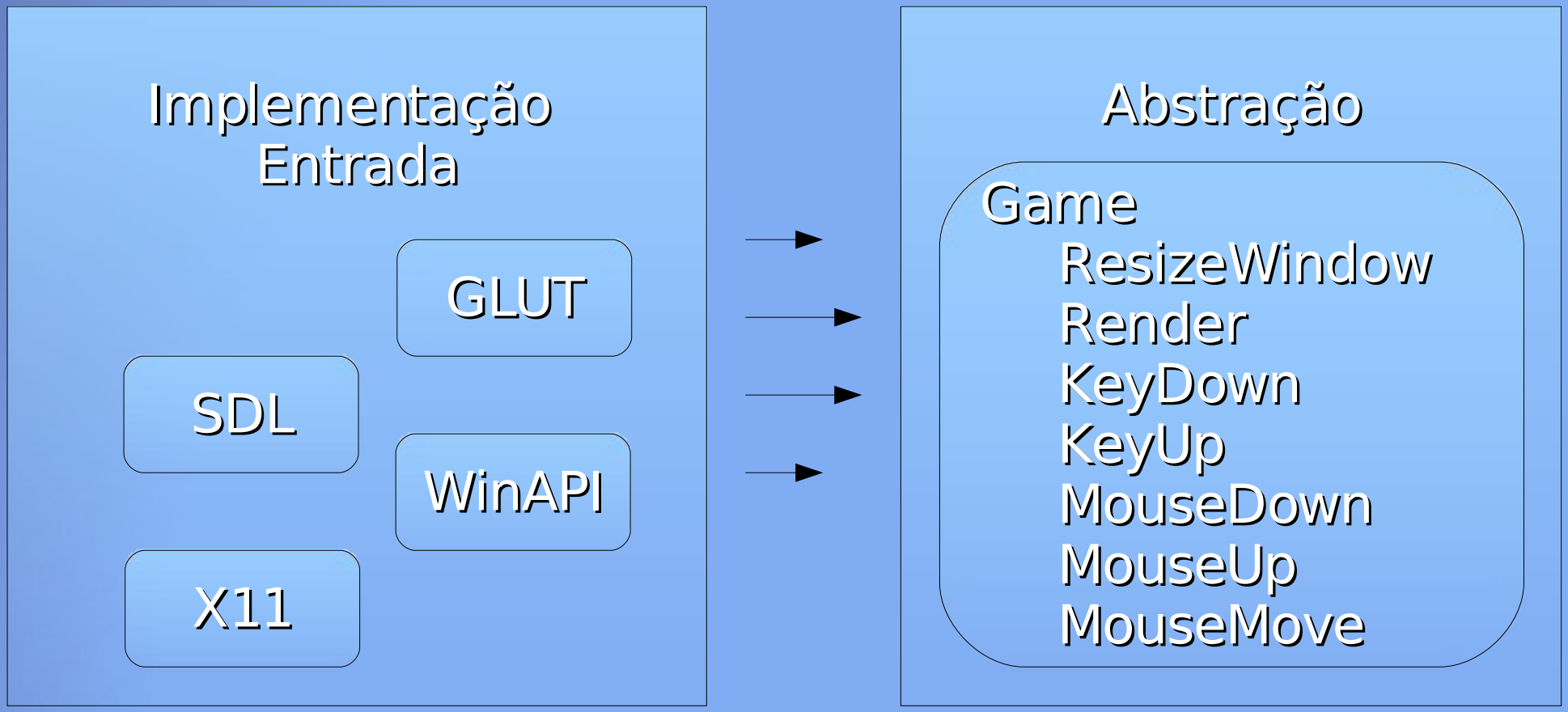
Etapas de desenvolvimento
Sistema de entrada/saida

- Lembrando: abstrações são amigas

- Definição de entradas de mouse e teclado:
  - Pressionar/soltar botão do teclado
  - Movimento passivo/ativo de mouse

Etapas de desenvolvimento
Sistema de entrada/saida

- Devem-se manter as semanticas das funções ou métodos definidos



Etapas de desenvolvimento
Sistema de entrada/saida

```
class Game{
   ...
  public:
  Game(int w,int h);
  ~Game();
//rendering functions
  void resizeWindow(const int w,const int h);
  void idleDraw(const unsigned int time_ms);
//keyboard functions
  void keyDown(const int key);
  void keyUp(const int key);
//mouse functions
  void mouseDown(const int btn);
  void mouseUp(const int btn);
  void mouseMove(const int x,const int y);
};
```

Etapas de desenvolvimento
Sistema de entrada/saida

- Movimento de mouse:
  - Passivo : Encontrado em aplicações que utilizam a posição corrente do cursor sobre a janela
  - Ativo : Encontrado em aplicações que exigam algum tipo de deslocamento infinito para alguma direção. Ex.:
    - jogos de tiro de primeira pessoa
    - TP1 de CG - Video ou Demonstração

Etapas de desenvolvimento
Sistema de entrada/saida

- Teclado (modos de utilização, invente o seu):
  - Guardar estado do teclado e utilizar o conjunto de teclas pressionadas na simulação.
    - Boid de CG - Video ou Demonstração

Etapas de desenvolvimento
Sistema de entrada/saida

- Teclado (modos de utilização, invente o seu):
  - Realizar uma ação assim que uma tecla é precionada (tetris - nosso projeto!!!)

Etapas de desenvolvimento
Sistema de entrada/saida

```
void Tetris::player1PushBtn(const BtnDefs btn){
  switch(btn){
    case Left:
      currentUnit->posx++;
      if(table.collide(currentUnit))
        currentUnit->posx--;
    break;
    case Rotate:
      currentUnit->rotate();
      if(table.collide(currentUnit)){
        currentUnit->rotate(); // no comments
        currentUnit->rotate();currentUnit->rotate();
      }else
        whoosh->play();//se conseguir rodar, toca um som
    break;
    case Down: setSimulationInterval_ms(25);
               keyState[Down] = true;
    ...
  }
}
```

```
enum BtnDefs{
  Rotate = 0,
  Down,
  Left,
  Right,
  AllBtnDefs
};
```

Etapas de desenvolvimento
Renderização de fontes

Etapas de desenvolvimento
Renderização de fontes

- Como a API escolhida foi OpenGL, então a renderização de fontes é um fator complicador
  - OpenGL não renderiza fontes
- Existem 2 tipos de renderizações principais para fontes:
  - Através de imagens
  - Através de vetores e curvas
- O controlador de saida pode também possuir funções para renderizar fontes (SDL,GLUT)
  - É interessante saber utilizar estes recursos, mas aqui os controladores de saida servirão apenas para abrir uma janela e controlar a entrada!!!

Etapas de desenvolvimento
Renderização de fontes

- Renderização a partir de vetores e curvas
  - Utiliza linhas e poligonos para desenhar fontes
    - Possível através de XGL ou WGL
  - Pode se tornar pesado, pois são renderizados vários polígonos para se aproximar da curva original da fonte

Etapas de desenvolvimento
Renderização de fontes

- Renderização a partir de imagens
  - Mascara de bits
  - *Imagem em formato RGB ou RGBA
  - Textura

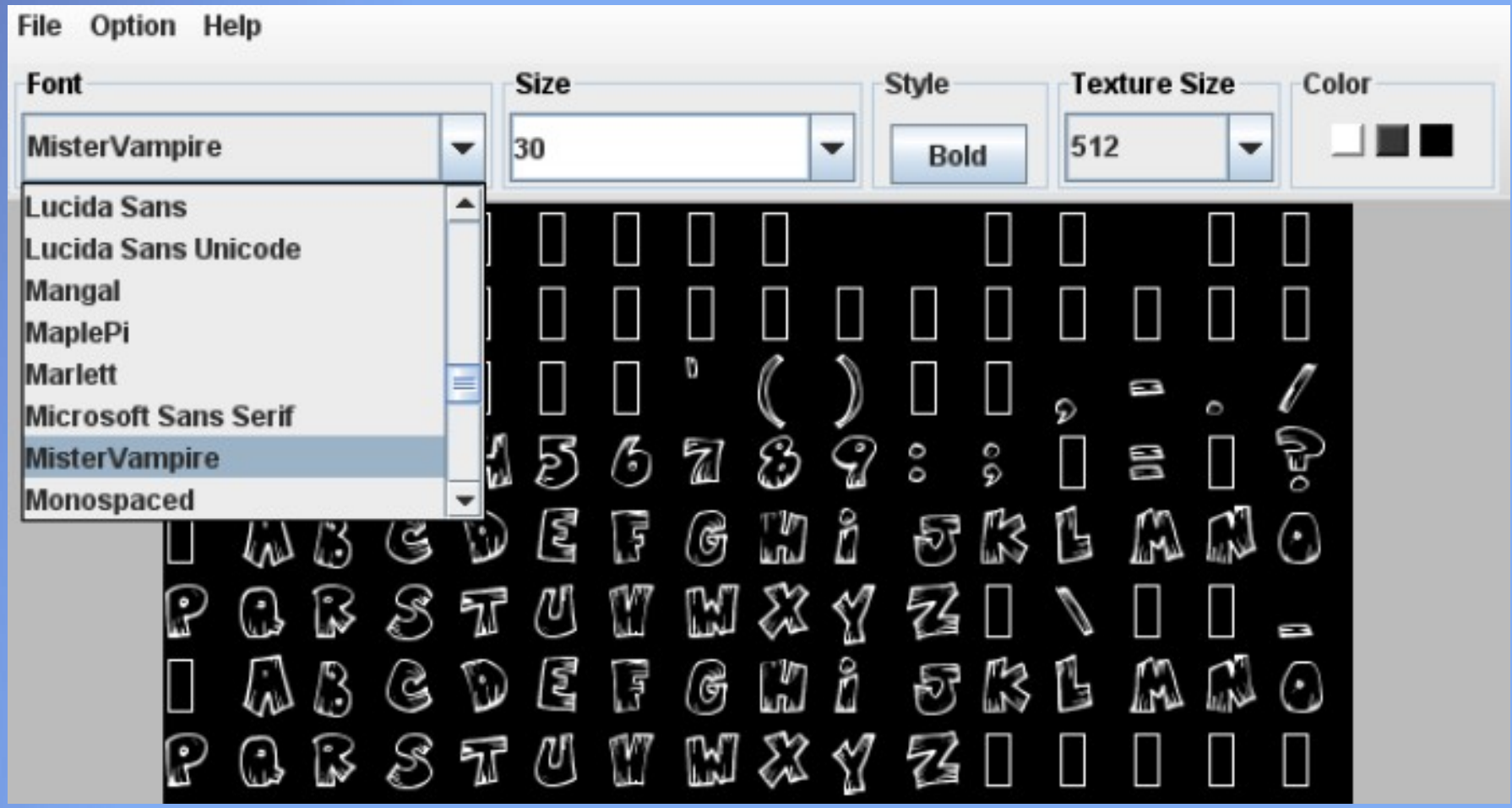
- Exige a transferência de um bloco de imagem para o dispositivo de video a partir da memória RAM ou VIDEO
  - Algumas placas antigas podem perder desempenho

Etapas de desenvolvimento
Renderização de fontes

- As imagens para fontes podem ser geradas pela aplicação ou virem de outra ferramenta
  - Foi utilizado o programa F2IBuilder para geração de fontes para o jogo
  - Foi utilizado o leitor de arquivos .png (libpng)

Etapas de desenvolvimento
Renderização de fontes

- FWBuilder para geração de fontes para o jogo



Etapas de desenvolvimento
Renderização de fontes

- A controladora de renderização de fonte foi criada com as seguintes funcionalidades:
  - Permite o desenho de uma fonte em qualquer posição da tela
  - Permite a configuração de vários modos de alinhamento
    - O alinhamento foi importante para a impressão das fontes na GUI

Etapas de desenvolvimento
Sistema de simulação

Etapas de desenvolvimento
Sistema de simulação

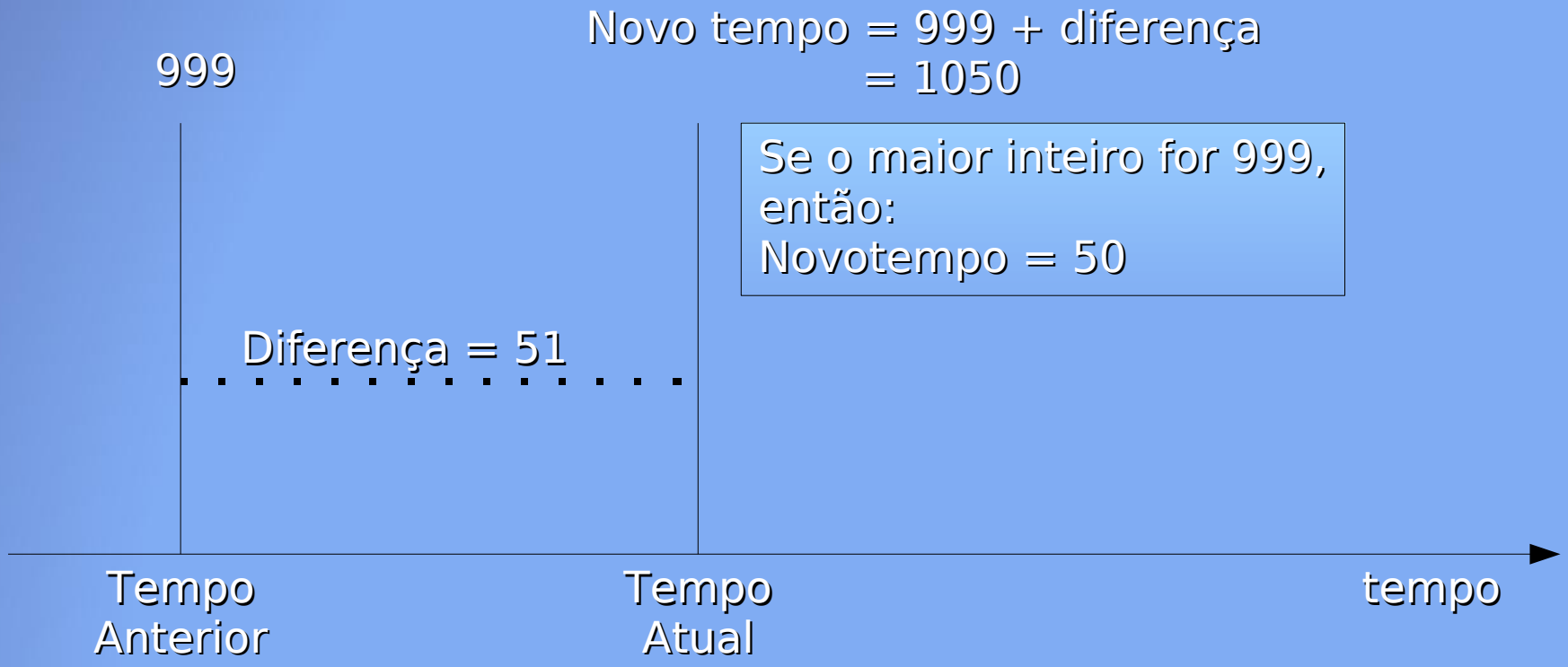
- O processamento de uma simulação em um computador ocorre de maneira discreta/amostral
  - O controlador de simulação que irá efetivamente realizar os "passos" do jogo e faze-lo interativo
  - O ajuste de velocidade pode ser implementado neste controlador
    - Pense naquele jogo que roda mais rapido em um PC que em outro

Etapas de desenvolvimento
Sistema de simulação

- Lembram-se do problema do Ponto Flutuante?
  - Ele apareceu aqui
    - Para resolver o problema a simulação é feita totalmente baseada em inteiros sem sinal
    - Foi implementada uma função para determinação da diferência entre 2 tempos que podem acarregar inclusive em uma diferença negativa de tempos consecutivos

Etapas de desenvolvimento
Sistema de simulação

- Um tempo atual pode ser menor que o tempo anterior

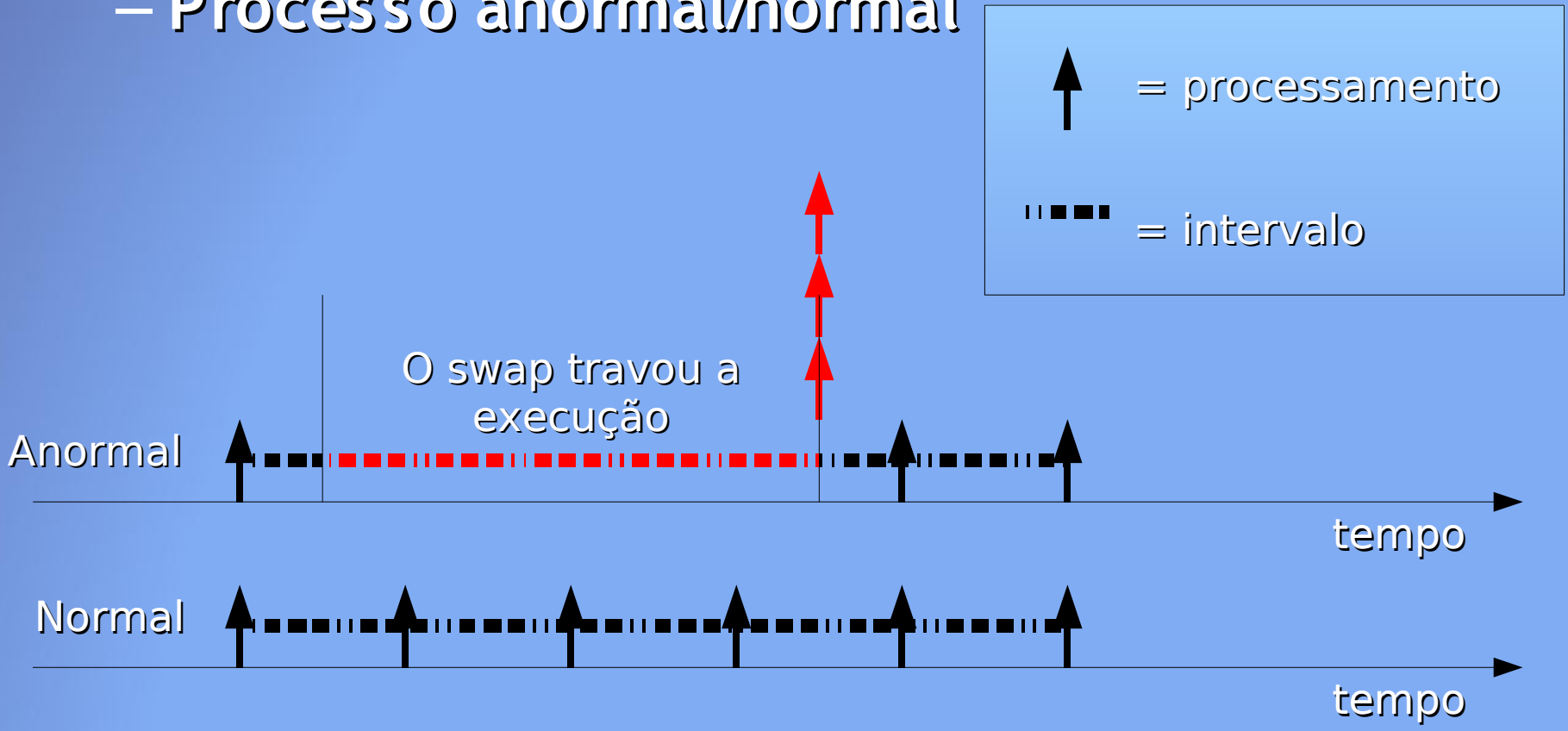


Etapas de desenvolvimento
Sistema de simulação

- Este sistema foi baseado em intervalos fixados pelo programador no qual deve existir um passo de simulação a cada intervalo.
  - Se for bem implementado pode permitir efeito slow motion automático!!!

Etapas de desenvolvimento
Sistema de simulação

- Linha de tempo
  - Processo anormal/normal



Etapas de desenvolvimento
Sistema de simulação

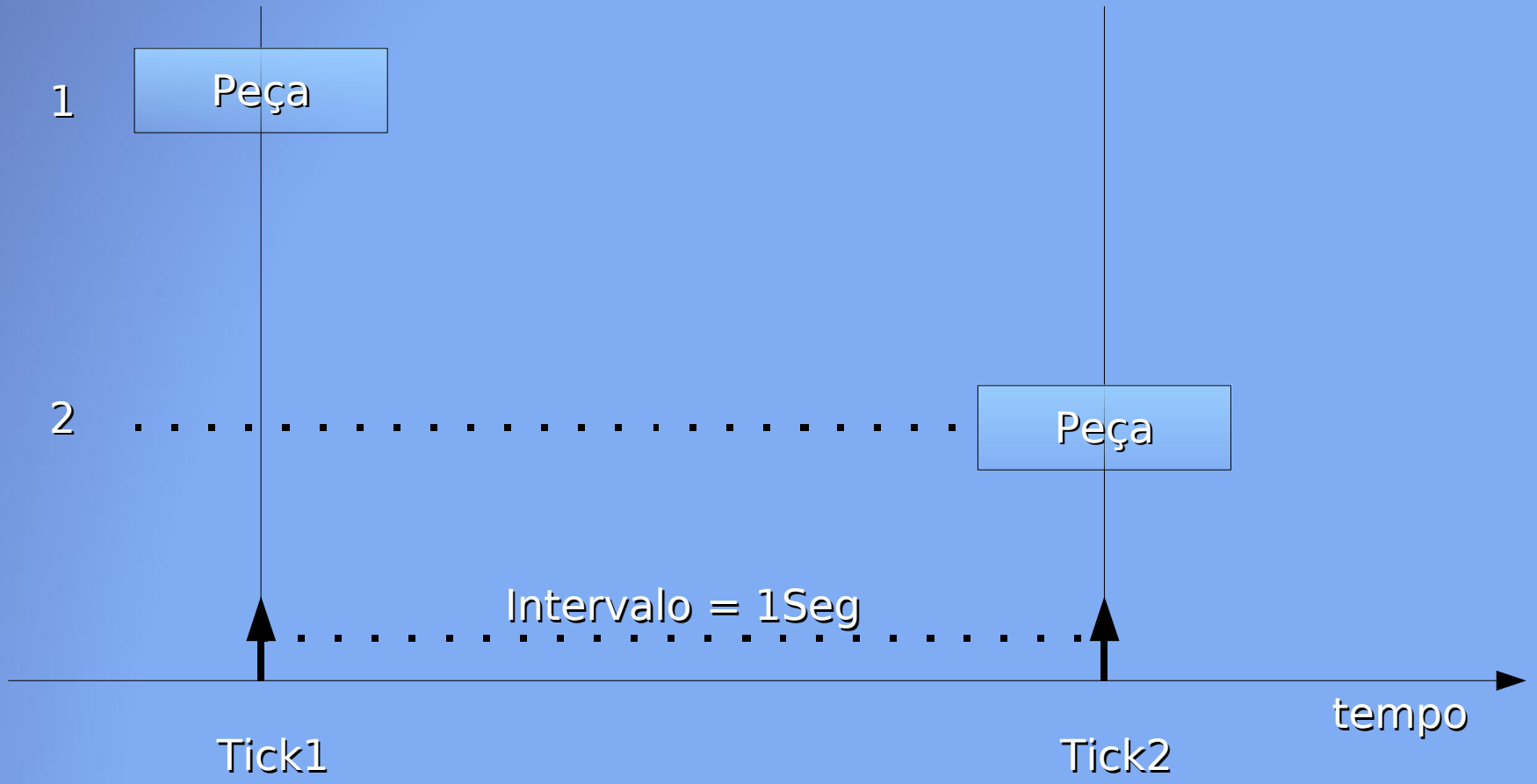
- A simulação pode ocorrer por exemplo de 500 em 500 milissegundos, mas e a renderização?
  - Será que vai ficar tudo parado até dar o "tick" da proxima simulação?
  - Desta forma serão renderizados muitos quadros iguais até o estado do jogo ser alterado?

Etapas de desenvolvimento
Sistema de simulação

- Pode-se utilizar métodos de interpolação entre as simulações para suavizar a animação
  - Um ganho com esta abordagem é a adaptabilidade da animação ao framerate que cada pessoa consegue ter em seu PC
    - Uma placa de video pode conseguir 200 FPS enquanto outra 2000 FPS
  - Quanto maior o framerate, mais lisa parecerá a animação!!!

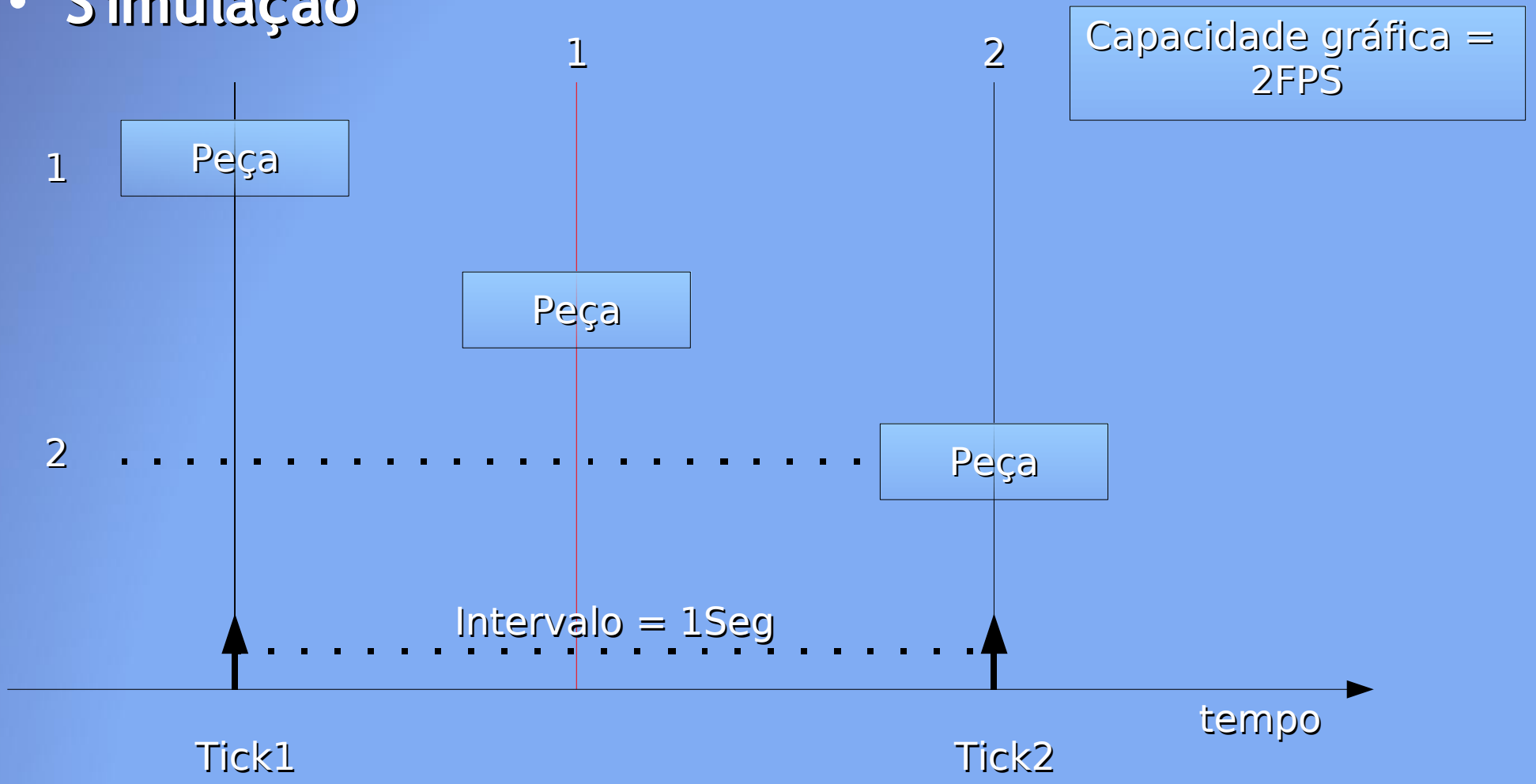
Etapas de desenvolvimento
Sistema de simulação

- Simulação



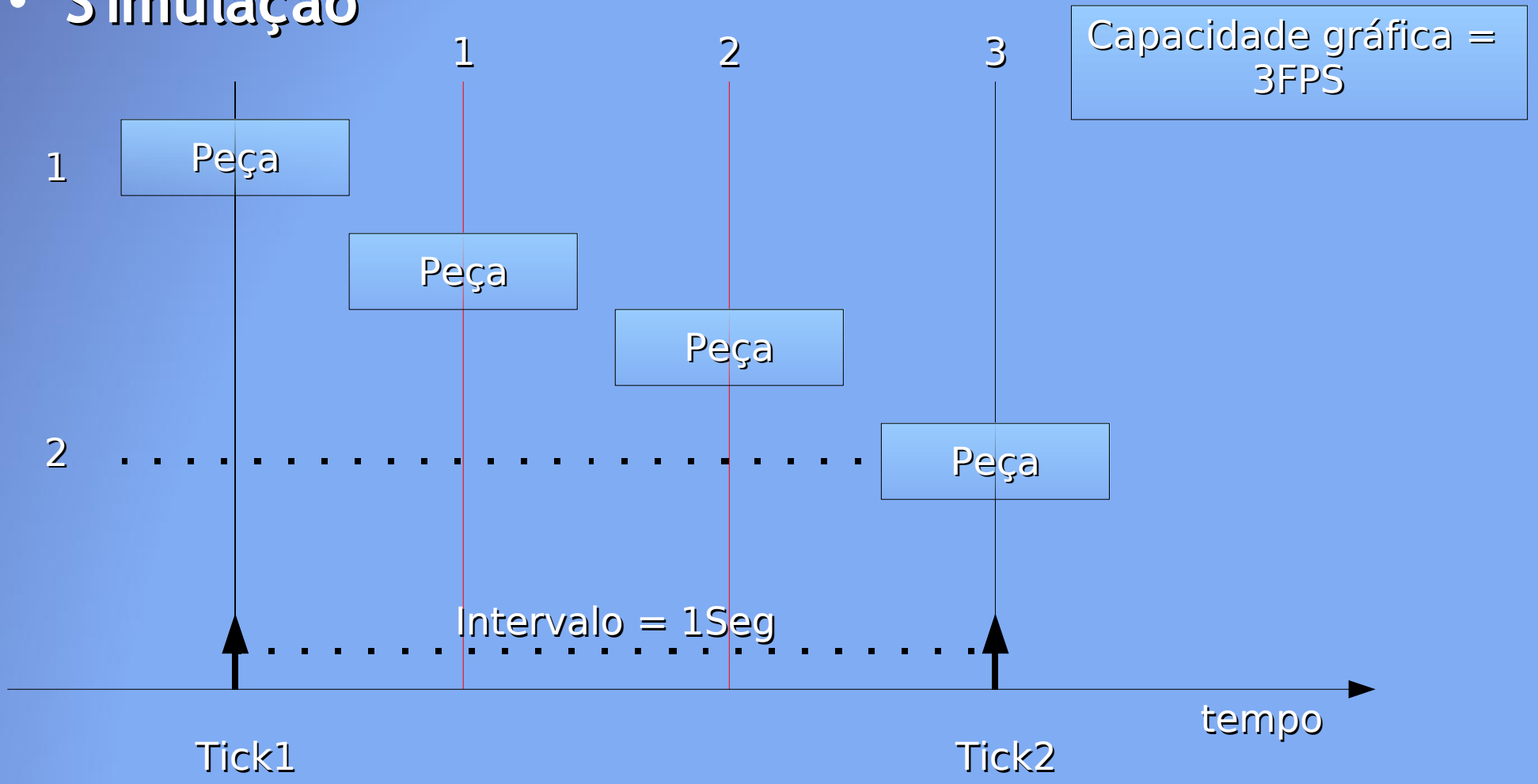
Etapas de desenvolvimento
Sistema de simulação

- Simulação



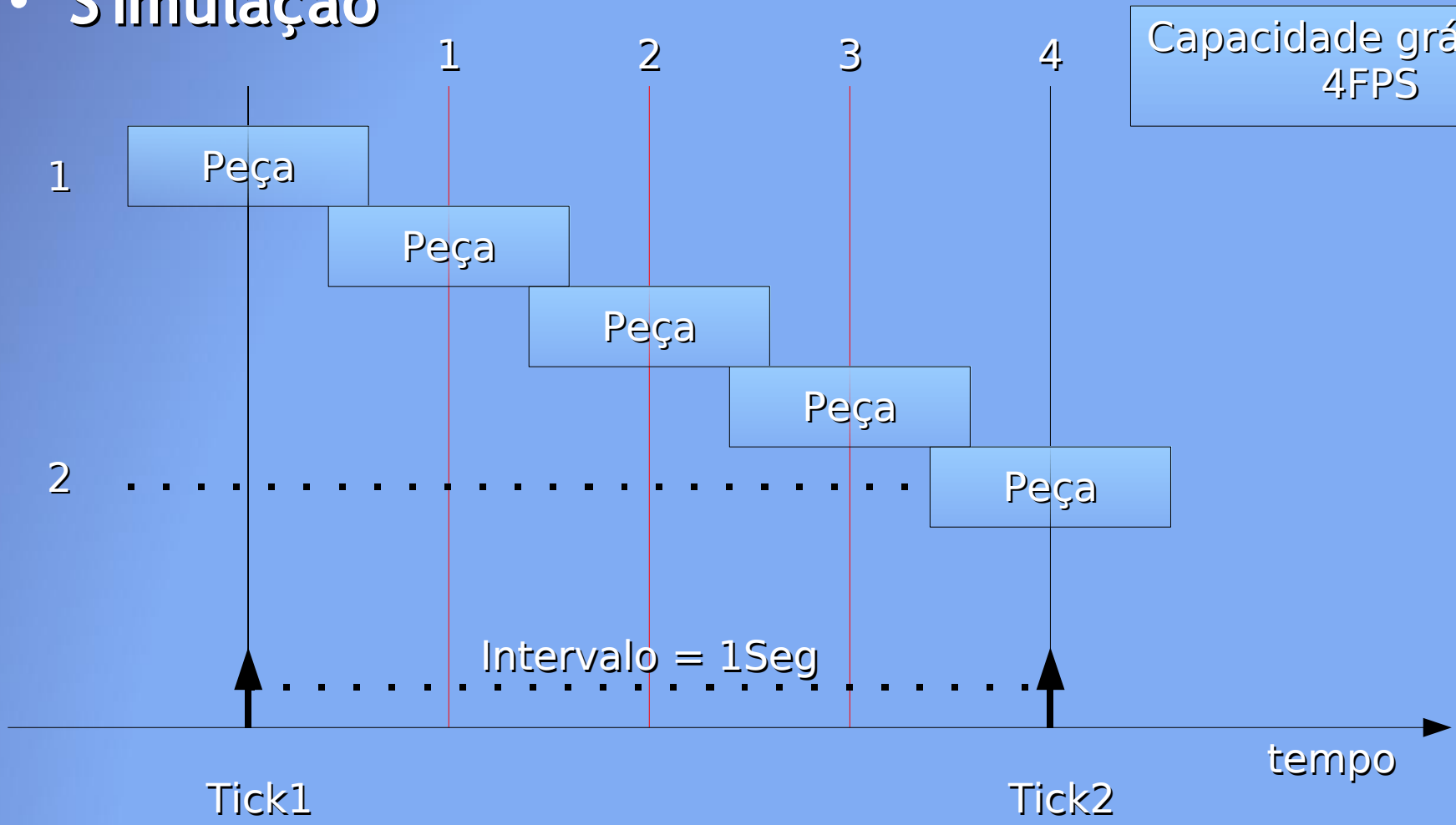
Etapas de desenvolvimento
Sistema de simulação

- Simulação



Etapas de desenvolvimento
Sistema de simulação

- Simulação



Etapas de desenvolvimento
Sistema de simulação

```
class GenericSimulation{
  protected:
  GenericSimulation(const unsigned int interval, const bool paused);
  virtual ~GenericSimulation();
  void setCounter(const unsigned int counter);
  public:
//deve ser chamado de dentro do processador em idle
  void makeSimulation(const unsigned int time_ms);
  unsigned int getSimulationInterval_ms()const ;
  void setSimulationInterval_ms(const unsigned int time);
  unsigned int getLastTimeSimulation_ms()const ;
  void setLastTimeSimulation_ms(const unsigned int time);
  bool getPausedSimulation()const ;
  void setPausedSimulation(const bool v);
// auxiliar para renderização
  float lerpCalc(const unsigned int time_ms)const ;
//método que deve ser implementado
  virtual void processSimulation(const unsigned int counter,
                                 const unsigned int time_ms) = 0;
};
```

Etapas de desenvolvimento
Sistema de simulação

```
void GenericSimulation::makeSimulation(const unsigned int time_ms){
  if(pausedSimulation){
     ...
    return;
  }
  lastDiffTimeSimulation_ms = diffSystime(time_ms,
                                          lastTimeSimulation_ms);
  //processamento de troca de intervalo
     ...
  while ( lastDiffTimeSimulation_ms >= simulationInterval_ms ){
    //simulation tick
    processSimulation(counter,time_ms);//simulação da aplicação
    lastTimeSimulation_ms += simulationInterval_ms;
    counter++;
    lastDiffTimeSimulation_ms = diffSystime(time_ms,
                                            lastTimeSimulation_ms);
    //processamento de troca de intervalo
     ...
  }
}
```

Etapas de desenvolvimento
Sistema de simulação

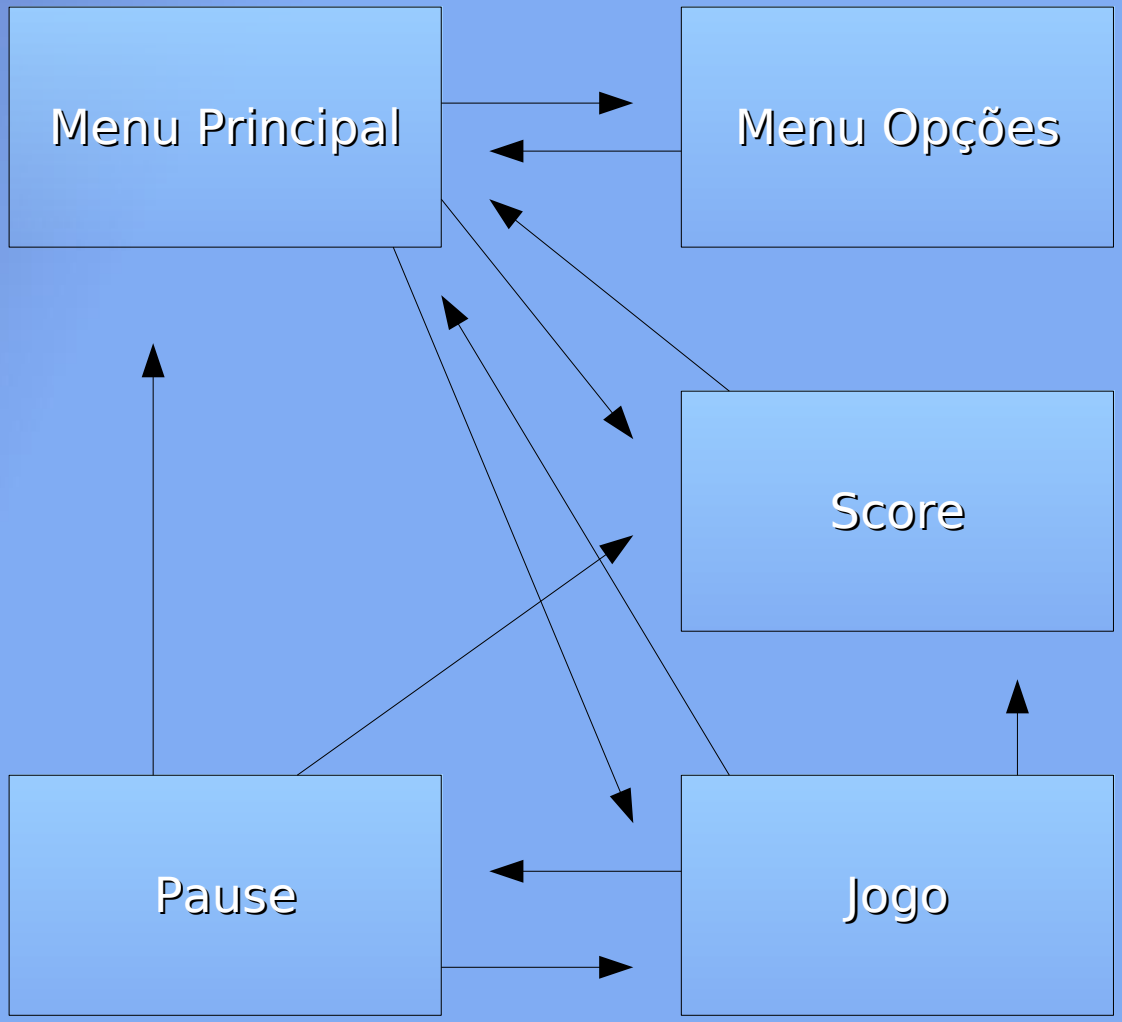
- Mostrar tetris com e sem interpolação
- Cuidado ao generalizar as interpolações
  - O desenvolvimento da simulação pode ficar muito complexo
  - Mostrar o Breakout não terminado

Etapas de desenvolvimento
Criação da GUI

Etapas de desenvolvimento
Criação da GUI

- Primeira coisa importante para se fazer uma GUI
  - Definir um diagrama de sequencia ao qual todo o jogo deve estar dentro

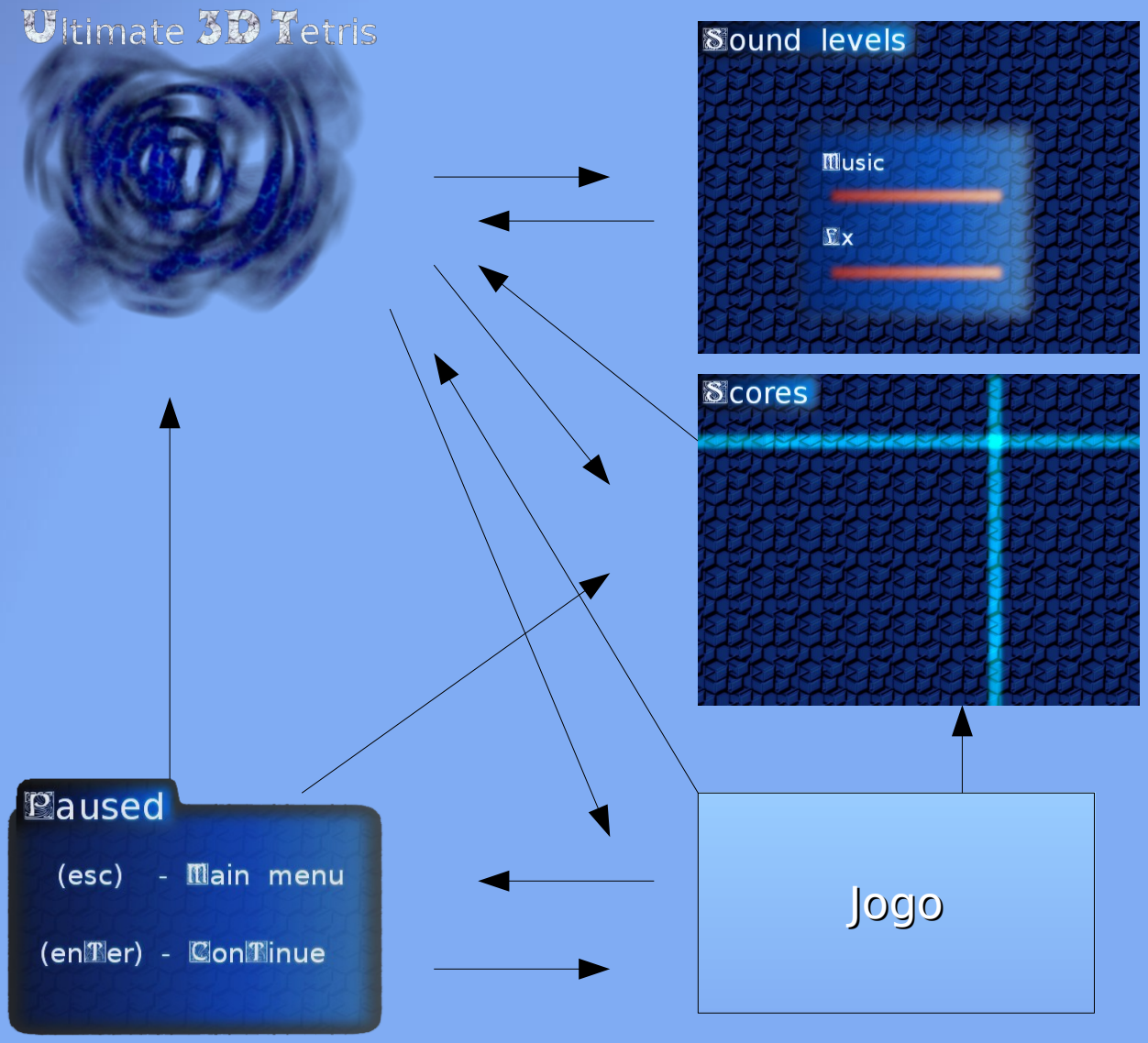
Etapas de desenvolvimento
Criação da GUI



Etapas de desenvolvimento
Criação da GUI

- Como disse anteriormente sou programador, mas quanto de programação deve compor a GUI?
  - Pode ser feita totalmente programada
    - O que poderia demorar
  - Pode ser feita com a ajuda de um artista

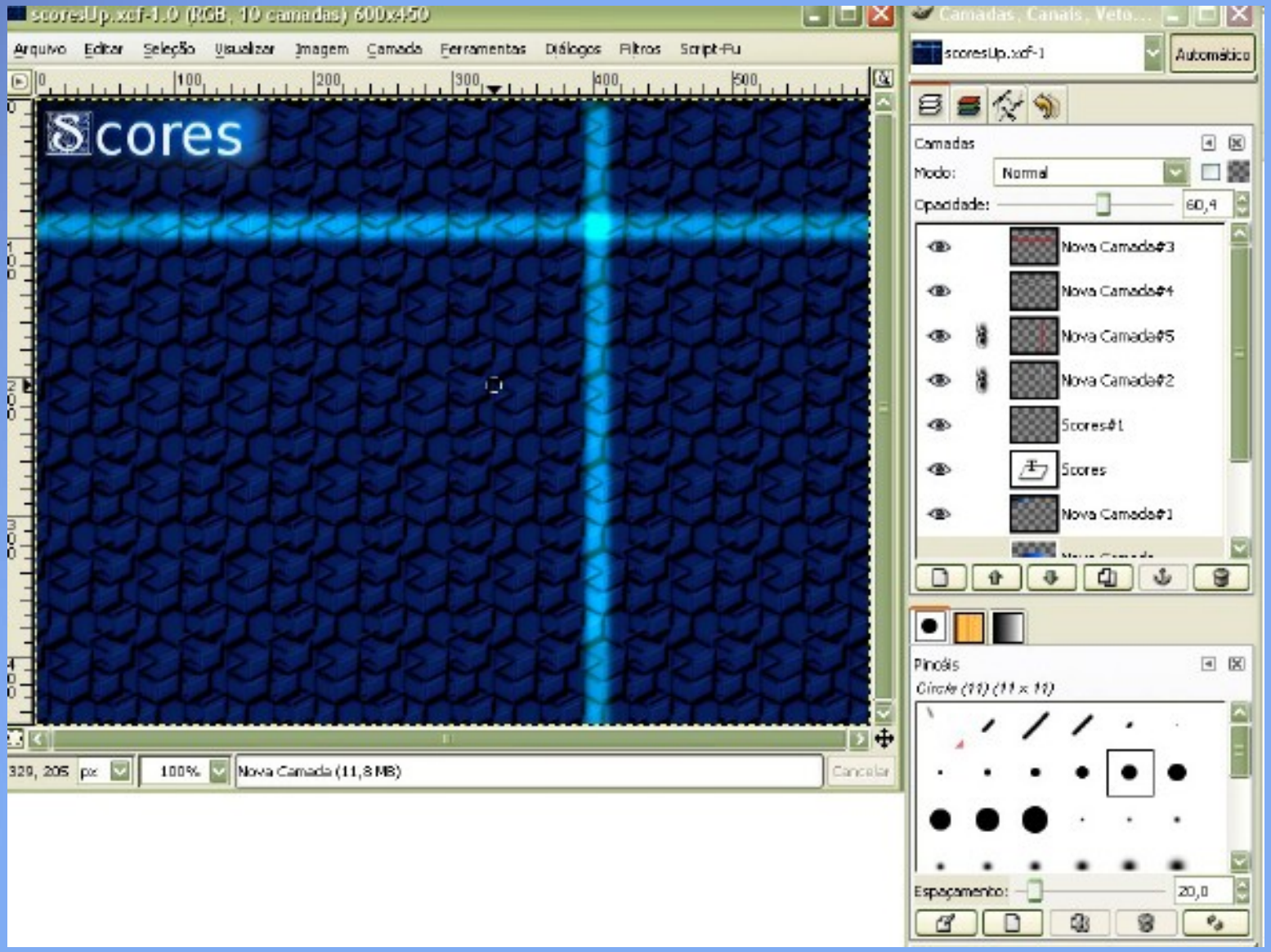
Etapas de desenvolvimento
Criação da GUI



Etapas de desenvolvimento
Criação da GUI

- As interfaces podem ser feitas em softwares como:
  - *GIMP
  - Photoshop
  - InkScape
  - etc...
- Eu tive ajuda do meu irmão para fazer as interfaces
  - Ele fez tudo xD

Etapas de desenvolvimento
Criação da GUI



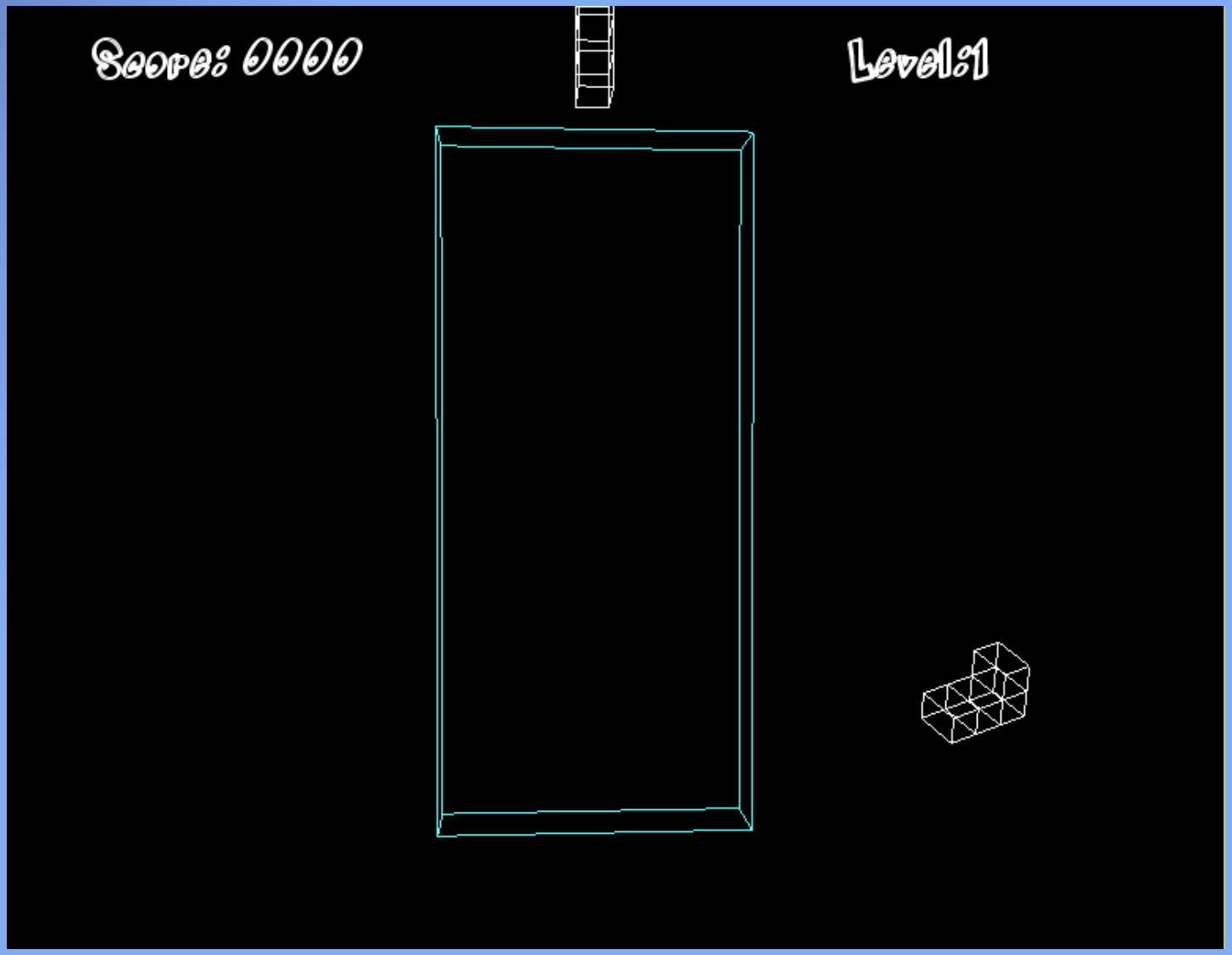
Etapas de desenvolvimento
Criação da GUI

- As GUIs foram implementadas de modo que elas contenham as informações sobre os recursos utilizados
  - Pode existir somente um objeto de UI criado (para evitar de gerenciar recursos não utilizados, que entram em cache, etc...)
  - As ações das UIs foram implementadas com base em identificadores que podem ser retornados

Etapas de desenvolvimento
Criação da GUI

- A tela do jogo também é uma UI
  - Devem ser posicionados as informações interessantes para o jogador como:
    - Pontuação
    - Nível
    - Próxima peça

Etapas de desenvolvimento
Criação da GUI



Etapas de desenvolvimento
Som

Etapas de desenvolvimento
Som (tema da última palestra do evento)

- Não entrarei em detalhes sobre a implementação pois este é o tema da última palestra do grupo deste semestre
  - Mas basicamente o que tem que ser feito é a escolha de bons sons de efeitos e músicas que transpareçam sua intenção sobre o jogo
  - Saber equalizar os sons para reprodução simultânea sem saturar o som, entre outras coisas

Etapas de desenvolvimento
Som

- Um jogo com som é outro jogo!!!!
  - Você pode colocar intenções de alegria ou tristeza apenas trocando entre músicas de fundo

Etapas de desenvolvimento
Som

- Saiba escolher a API certa para som
  - Deve-se pesar os objetivos ao serem alcançados com a API
    - OpenAL NÃO é boa para sincronização de lábios, pois ela é mais alto nível que o SDL_sound por exemplo
  - Foi utilizada a API OpenAL por oferecer reprodução de som de forma fácil
    - Só carregar o som e dar play!!!

Etapas de desenvolvimento
Som

- Boas fontes de sons, samples e músicas para aplicações não comerciais:
  - Google: “free samples”
- Para as músicas do tema foram utilizadas algumas mp3 encontradas no google
  - Tema tetris remixado dance
  - Tema tetris tocado por uma orquestra japonesa
  - Tema tetris original
  - Tema tetris versão trance

Etapas de desenvolvimento
Som

- Sons de efeitos que queremos
  - Aplicação de rotação sobre a peça
    - Pode ser um barulho como se alguma coisa estivesse passando rapido perto do ouvido
  - Quebrar uma linha do muro
    - Quero colocar o som de vidro sendo quebrado
  - Encaixar uma peça sobre o muro
    - Dar um “plink!” << to bom para fazer onomatopéias também
- Reproduzir sons!!!

Etapas de desenvolvimento
Som

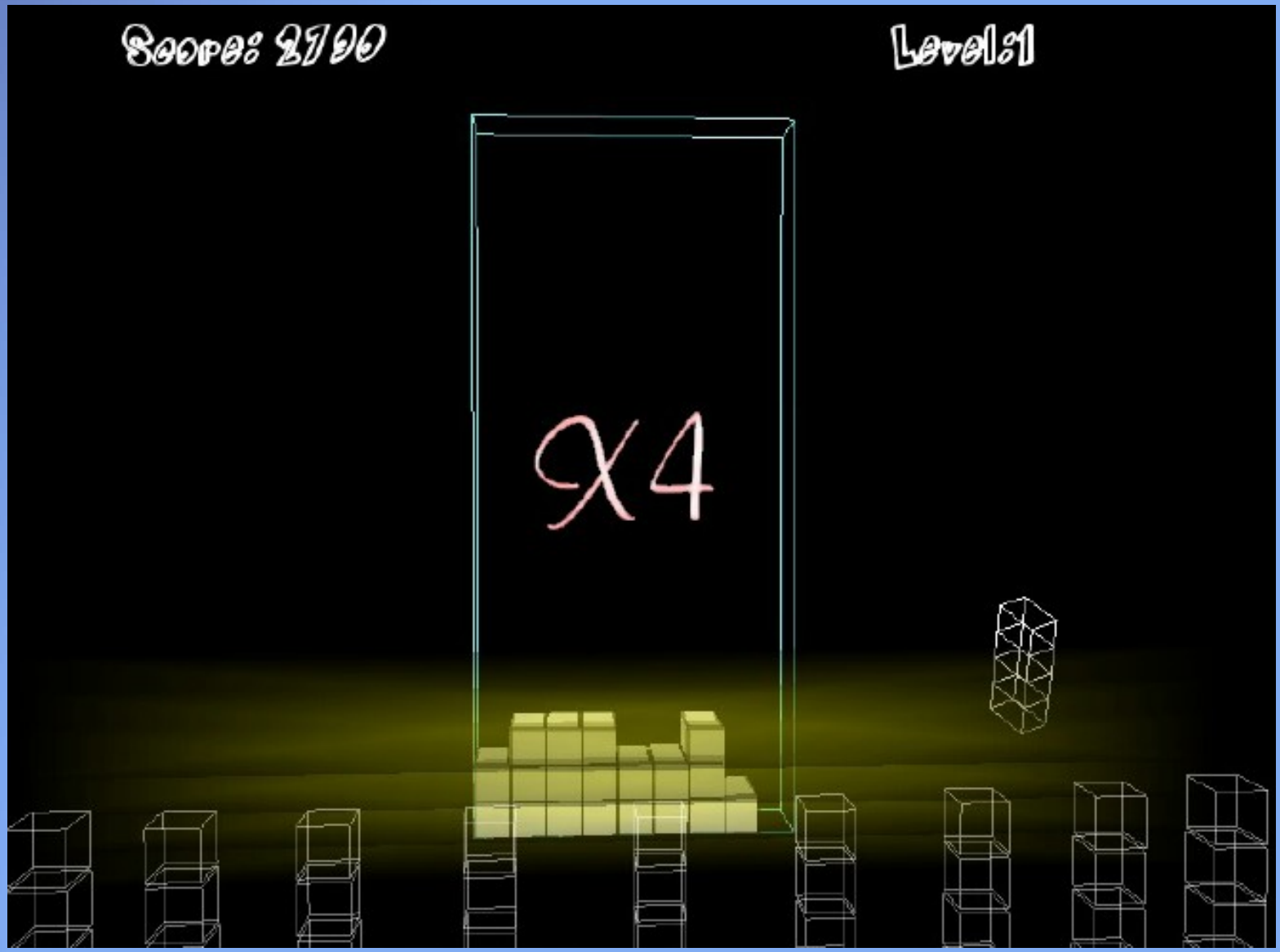
- Os objetos de recursos como imagens, sons, texturas, etc... devem ser alocados de acordo com a sua UI
  - Desta forma os sons também são objetos e "sabem" como ser criados, reproduzidos, mixados e apagados

Etapas de desenvolvimento
Efeitos visuais

Etapas de desenvolvimento
Efeitos visuais

- Depois do jogo estar jogável é possível se melhorar algumas coisas:
  - Mostrar os combos de acordo com o número de linhas que foram quebradas
  - Fazer os quadrados quebrados voarem em direção a tela
  - Efeito visual de explosão sobre a linha quebrada

Etapas de desenvolvimento
Efeitos visuais



Etapas de desenvolvimento
Efeitos visuais

- Os efeitos visuais podem ser pensados em qualquer hora do desenvolvimento
  - Tenha em mente que escolher efeitos complexos pode acarretar em maior tempo para implementação, ainda mais sem saber como a aplicação realmente irá funcionar

Etapas de desenvolvimento
Estados persistentes do jogo

Etapas de desenvolvimento
Estados persistentes do jogo

- Que graça teria um jogo que você consegue fazer 99.999 pontos sem poder provar para seu colega sua façanha.

- Foi criado um sistema de persistencia baseado em XML justamente para armazenar os scores dos jogadores.

- Além disto também se implementou um controlador que permite 2 níveis de hierarquia dentro do XML

Etapas de desenvolvimento
Estados persistentes do jogo

- Com o sistema de persistencia foi possível salvar as informações
  - Aplicação
    - Som
      - Nível música
      - Nível efeitos
    - Pontuação
      - Pontuação e nome player 0
      - ...
      - Pontuação e nome player 4

Etapas de desenvolvimento
Estados persistentes do jogo

- Biblioteca: TinyXML
  - Facil de utilizar
  - Apenas precisa se colocar as fontes e compilar
  - Implementado em DOM(Document Object Model)
- Mostrar config.xml

Etapas de desenvolvimento
Possíveis melhorias

Etapas de desenvolvimento
Possíveis melhorias

- Características que ficaram para a próxima release do jogo:
  - Colorir peças com cores diferenciadas por peça
  - Colocar shaders (palestra de shaders!!!)
  - Fazer algum modo multi-player
  - Fazer porte para XNA e rodar o tetris num XBOX360 << gostei desse

Teste da versão beta

- Se o computador não tiver problemas com opengl e reprodução de som, testar o proprio jogo
  - Caso contrário, utilizar o video gravado

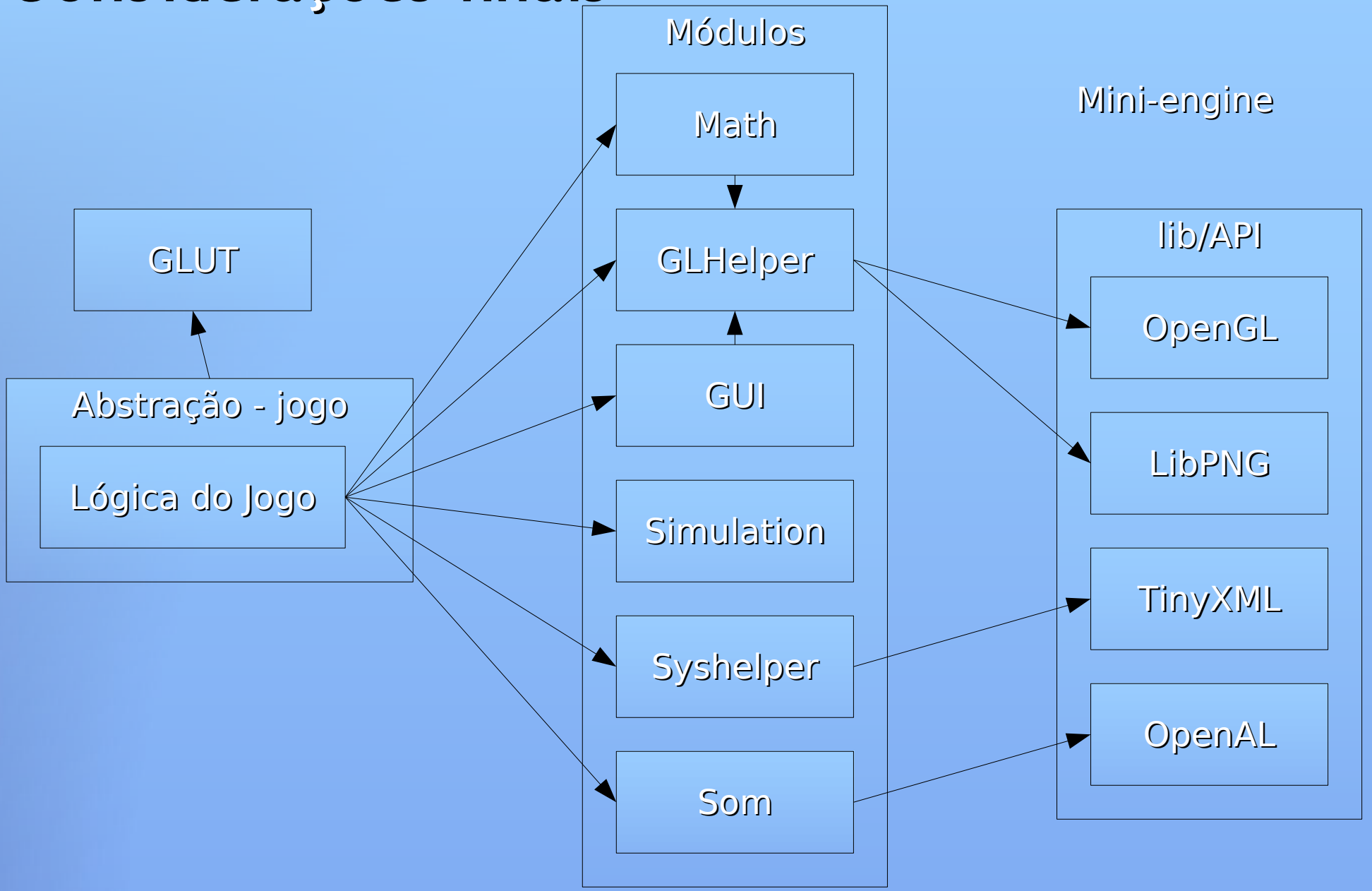
Considerações finais

- Reflexão sobre todo o processo
  - Definição do objetivo da aplicação
    - Requisitos
  - Definir plataforma alvo
    - Design
  - Projeto de interface
  - Implementação e teste de módulos (prototipação)
  - Finalização e criação de pasta com binários e recursos (deploy)

Considerações finais

- Reflexão sobre todo o processo
  - Pergunta:
    - Isto lembra parte do processo de desenvolvimento de alguma aplicação que você ja fez? (levantem a mão)

Considerações finais



Perguntas?

Alessandro Ribeiro da Silva
Site: www.alessandrosilva.com
Email: alessandro.ribeiro.silva@gmail.com
asilva@dcc.ufmg.br