Motor de Jogos e Arquitetura

Arquitetura por trás de uma *game engine*

Slides por:
Gustavo Ferreira Ceccon (gustavo.ceccon@usp.br)

Este material é uma criação do
Time de Ensino de Desenvolvimento de Jogos Eletrônicos (TEDJE)
Filiado ao grupo de cultura e extensão Fellowship of the Game (FoG), vinculado ao ICMC - USP

Objetivos

- Conceitos básicos e as partes de um motor de jogos
  - Núcleo, Física, Gráfico, Áudio e Redes
- Arquitetura de um jogos e seus objetos
  - Mundo e cena
  - Modelo herança e composição
- Linguagens de programação e paradigmas
- Núcleo da engine e jogo
  - Adicionar cenas e objetos e controle de fluxo

Objetivos

- *Game loop* e o que tem dentro dele
  - Tipos de *game loops*
  - Interpolação e extrapolação
  - Métodos de integração: Euler, Verlet e RK4
  - Programação concorrente
- Padrões de projeto
  - Estrutural e criacional
- Formas de *input*

Índice

1. Introdução

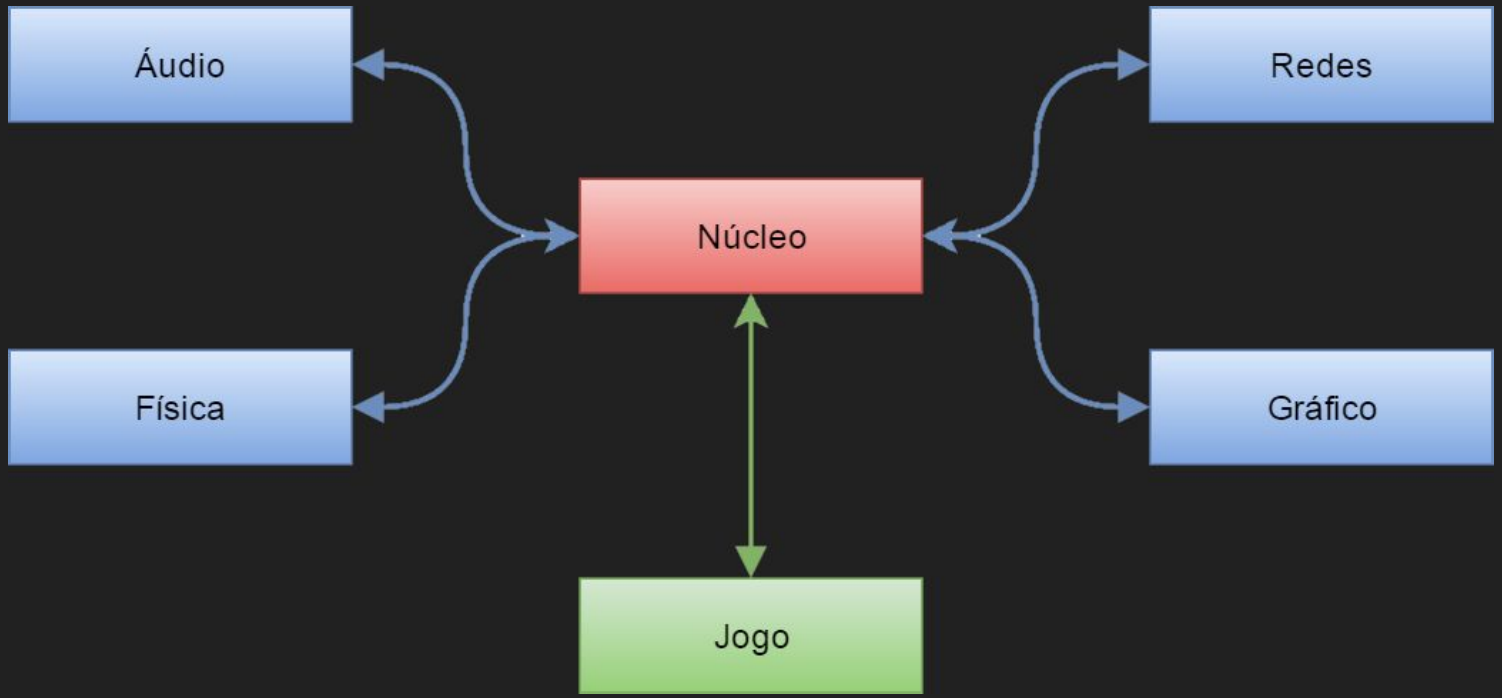
1. Introdução

- O que é um motor de jogos?
  - Estrutura fundamental
  - Módulos
- Por que estudar?
  - Funcionamento do hardware e software

1. Introdução

- O que oferece?
  - Interface com o programador
  - Funções básicas
  - Estrutura básica
  - Exportação para múltiplas plataformas (geralmente)

1. Introdução



1. Introdução

Exemplo completo

2. Estrutura

2. Estrutura

- Estrutura geral
  - Jogo
  - Cenas
  - Objetos
- Modelo hierárquico (árvore)

2. Estrutura

- Jogo
  - Janela
    - Jogo em si
    - Editor
  - Engloba as cenas, podendo rodar uma ou mais ao mesmo tempo

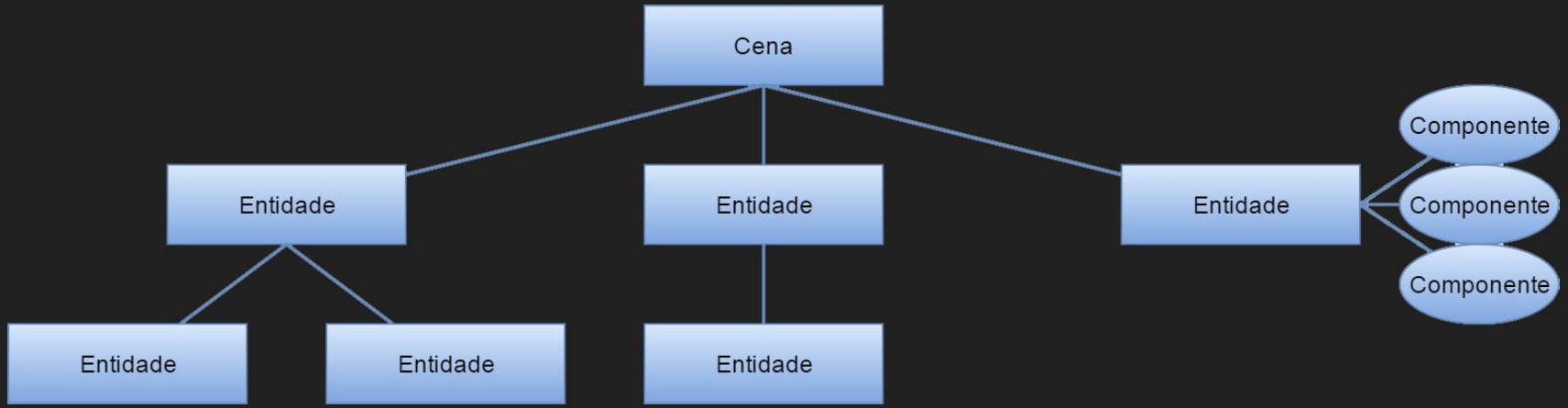
2. Estrutura

- Cena
  - Tela
    - Menu
    - Fase 0, 1, 2, ...
  - Engloba os objetos
  - Cuida do funcionamento geral dos objetos e da hierarquia da estrutura

2. Estrutura

- Objeto
  - Atores ou objetos estáticos
  - Possuem ***n*** comportamentos
  - Objetos tem seu próprio comportamento e interagem com outros objetos usando a física ou referência à outras instância de objetos

2. Estrutura


Cena
Entidade
Entidade
Entidade
Componente
Componente
Componente
Entidade
Entidade
Entidade

2. Estrutura

- Hierarquia em árvore
  - Transformações relativas e globais
  - Um (mal) exemplo: o Sol, a Terra e a Lua

2. Estrutura

Exemplo Haze

3. Arquitetura

3. Arquitetura

- ➔ *Game objects* (GO)
  - Todo o resto
  - Tem atributos, comportamentos, física, áudio, comunicam entre si etc.

3. Arquitetura

- Programação imperativa
  - Simples, rápido, *hardware-friendly* (até certo ponto na história) etc.

3. Arquitetura

- Programação orientada a objeto
  - Herança, polimorfismo, classes, simulação do mundo físico, abstração etc.

3. Arquitetura

- ➔ Programação orientada a objeto
  - ◆ Bom para jogos
    - Instâncias representam estados de objetos
    - Reaproveitamento de código, herança e polimorfismo
  - ◆ Ruim pelos mesmos motivos
    - Jai

3. Arquitetura

➔ Herança vs. Composição

3. Arquitetura

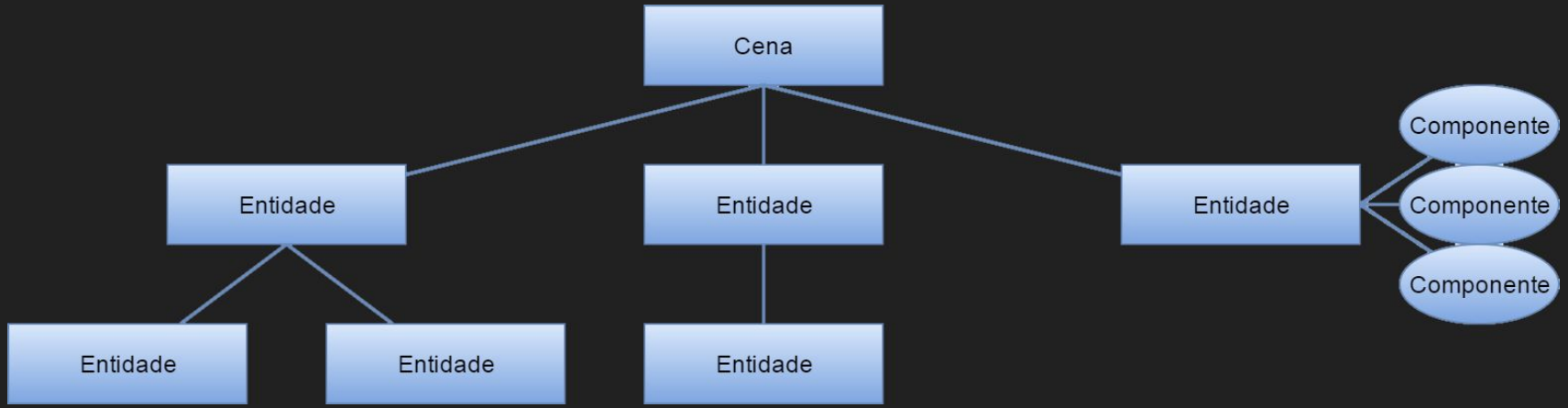
- Herança
  - Pros: usa POO, é bem direto na implementação
  - Cons: difícil manutenção, não funciona bem com jogos, *buble-up effect*

3. Arquitetura

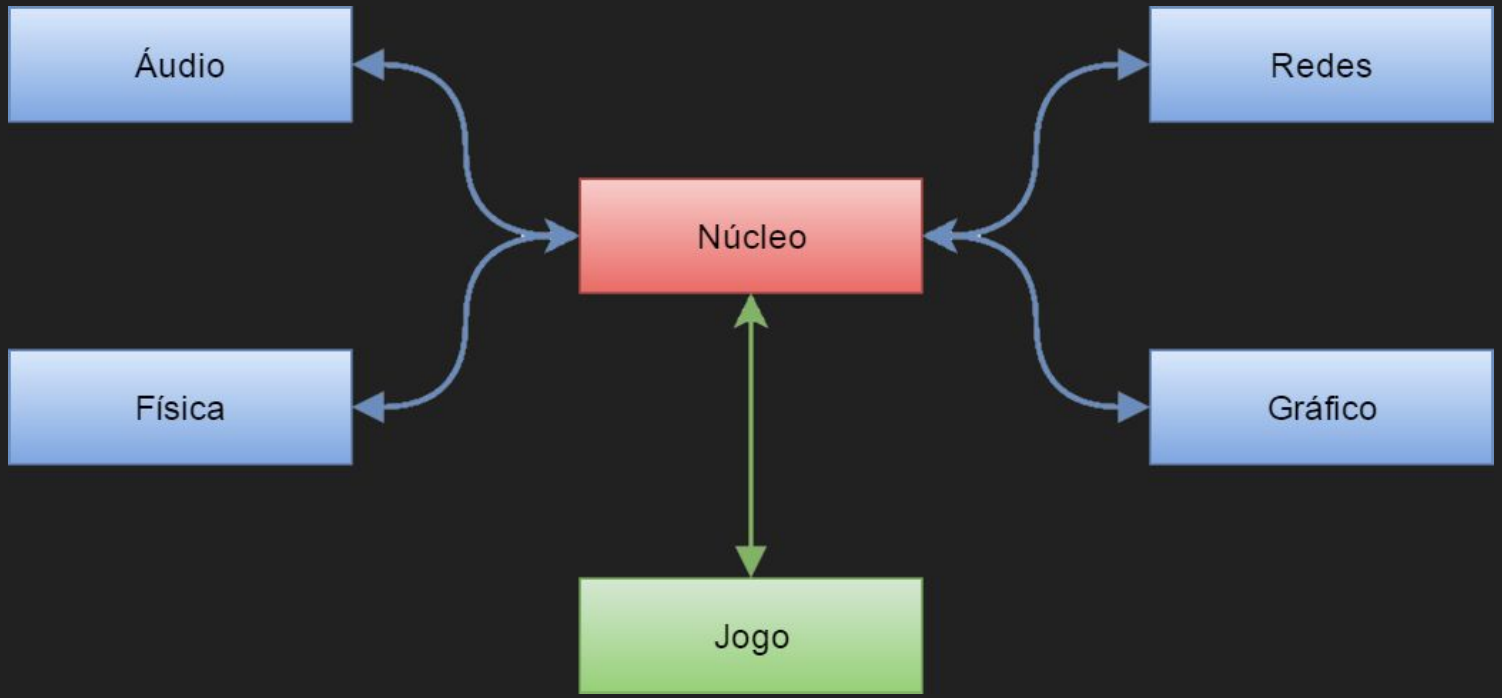
- ➔ Composição
- ➔ Pros: fácil manutenção, funcionalidade podem ser adcionadas em tempo de execução, flexível e robusto
- ➔ Cons: difícil modelar classes pequenas, *overhead,* ruim para jogos de pequeno porte

4. Jogo

4. Jogo


Cena
Entidade
Entidade
Entidade
Entidade
Entidade
Entidade
Componente
Componente
Componente

4. Jogo



4. Jogo

- Núcleo
  - *World* - Mundo, jogo, janela, editor
  - *Scene* - Cena, tela, fase, menu
  - *Entity* - Entidade, classe, *game object*, *player*
  - *Behaviour* - Comportamento, componente

4. Jogo

1. Herdar um jogo
    a. Definir parâmetros de plataforma, tela, entrada, saída, gráficos, áudios, física etc.
    b. Adicionar ao jogo uma cena inicial
    c. Adicionar fluxo de controle de cenas (opcional)

4. Jogo

2. Herdar uma cena
    a. Criar objetos dentro da cena
    b. Definir a hierarquia dos objetos (árvore)
    c. Ativar fluxo de controle (opcional)
    d. Controlar o fluxo de cena (opcional)

4. Jogo

3. Herdar uma entidade
    a. Criar atributos e métodos para essa entidade
    b. Adicionar comportamentos gerais
    c. Definir propriedades dos comportamentos

4. Jogo

4. Herdar um comportamento
    a. Criar um comportamento específico para o objeto
    b. Usar as funções de inicialização e *game loop* para criar a lógica / comportamento do objeto

4. Jogo

➔ Exemplo: Pong

5. Game Loop

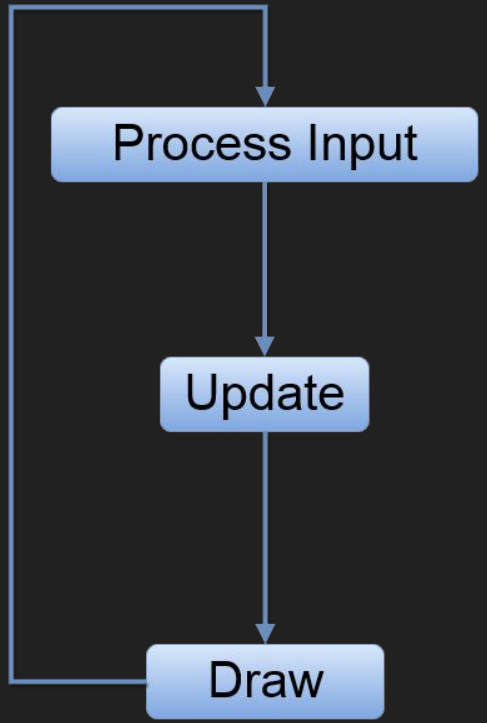
5. Game Loop

- ➔ Básico do jogo
- ➔ Desde sempre presente em jogos
- ➔ Laço “infinito” que cuida do funcionamento do jogo
- ➔ Roda métodos de integração

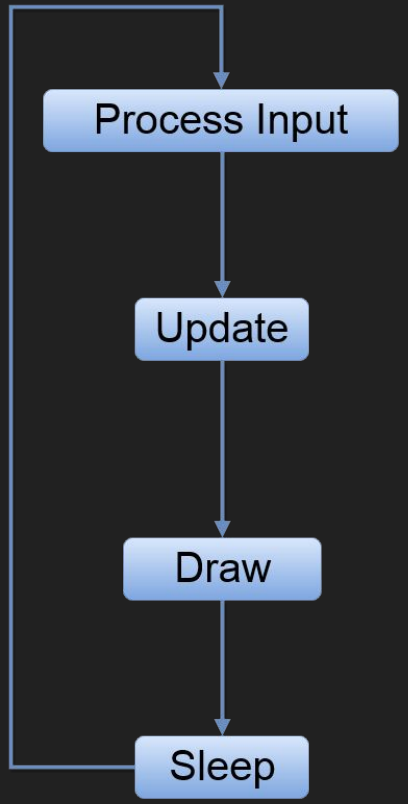
5. Game Loop

- Tipos
  - Simples: CPU-*dependent*
  - Simples com dt: *CPU-independent*
  - Simples com dt fixo: CPU rápida simulando CPU-*dependent*
  - *Catch-up* simples: atualiza de acordo com o tempo de *render*
  - *Catch-up* com extrapolação: atualiza de acordo com o tempo de *render* e extrapola o restante

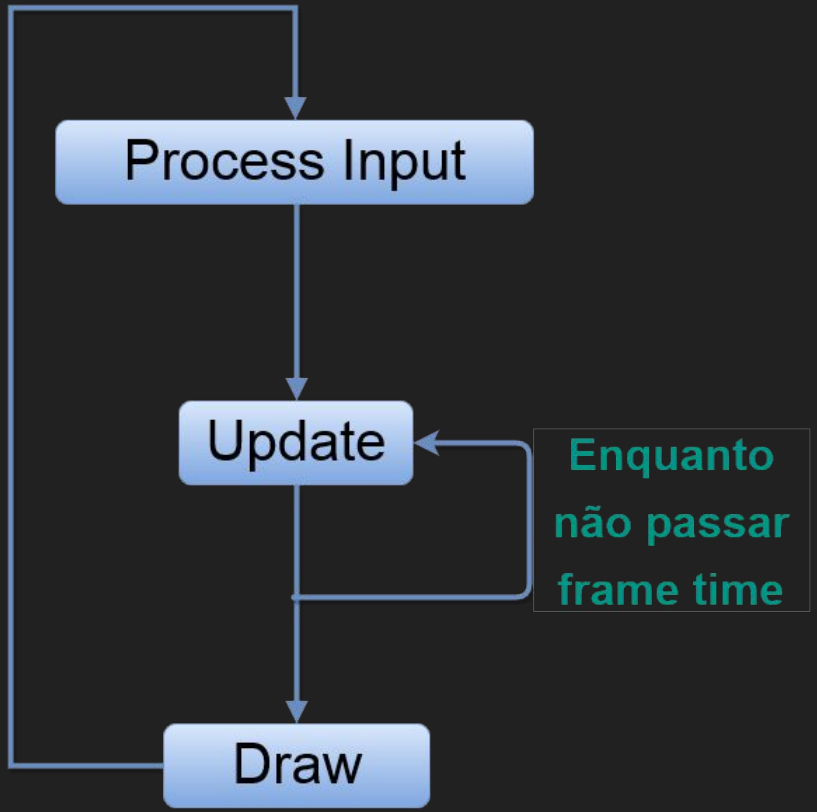
5. Game Loop



5. Game Loop



5. Game Loop



5. Game Loop

Exemplo Unity

5. Game Loop

- Programação concorrente
  - CPUs atuais chegaram num limite de clock
  - CPUs com menos clock e com mais núcleos trabalhando em paralelo
  - O *game loop* como está não aproveita a arquitetura atual de processadores

5. Game Loop

- Programação concorrente
  - CPU passa maior parte esperando a GPU e a placa de vídeo renderizar o mundo
    - Solução: separar os processos de *render* e de *update*
    - Já é feito assim

5. Game Loop

➔ Por que não colocar tudo numa *thread* separada?

5. Game Loop

- ➔ Problemas
  - Comunicação entre *threads*
  - *Thread safety*
  - Consumo de energia
  - Programação
    - *Job control*
    - *Thread pool*

5. Game Loop

- Física
  - Simulação, colisão e resposta
- Gráfico
  - GPU já faz isso
- Matemática
  - Matriz, vetores e *arrays*

5. Game Loop

- Inteligência artificial
  - Algoritmos optimizados em paralelo
- *Update*
  - CUIDADO: ordem de execução e dependência de dados

6. Padrões de Projeto

6. Padrões de Projeto

- Problemas comuns resolvidos
- Não forçar um padrão sobre um problema
  - Ter conhecimento dos padrões
- Geralmente voltados a problemas de POO
- Tipos: criação, estrutural, comportamental, concorrente

6. Padrões de Projeto

- Padrões de projetos
  - *Singleton* (criação)
  - *Observer* (comportamental)
  - *Flyweight* (estrutural)
  - *Prototype* (criação)
  - Composite (estrutural) já foi visto!

6. Padrões de Projeto

- Singletone
  - Exemplo: um *game manager* que contém informações sensíveis e que devem ser únicas. Ele controla coisas dentro de um jogo e múltiplas instâncias podem causar problemas

6. Padrões de Projeto

- ➔ Solução
  - Deixar o construtor como privado
  - Criar uma função global/estática (acessível) que retorna a instância única
    - A instância pode ser criada sempre no início
    - A instância pode ser criada na primeira chamada
      - *Lazy initialization* (outro padrão!)

6. Padrões de Projeto

- ➔ *Observer*
  - Exemplo: uma HUD precisa saber se o jogador perdeu o não para mostrar a tela de *game over*, porém não é uma boa ideia chamar uma função a partir do *player*

6. Padrões de Projeto

- ➔ Solução
  - Criar um evento de *callback* na HUD
  - Adcionar um *listener* do script da HUD ao *observer* do *game manager*
  - Quando o jogador perder, o *game manager* ativa o *observer* / evento, chamando a função de *callback*

6. Padrões de Projeto

- *Flyweight*
  - Exemplo: precisamos renderizar um conjunto de objetos iguais (árvores) com o mesmo modelo e em posições diferentes

6. Padrões de Projeto

- ➔ Solução
  - Compartilhamos o mesmo objeto de modelo e textura para a árvore e criamos apenas objetos com a referência ao mesmo
  - Criamos novas instâncias apenas para a posição da árvore

6. Padrões de Projeto

- *Prototype*
  - Exemplo: precisamos criar um *spawner* de monstros que tem o mesmo comportamento e atributos

6. Padrões de Projeto

- ➔ Solução
  - Criamos um *prefab*, ou seja, um modelo do monstro
  - Criamos uma classe *spawner* que recria vários clones desse objeto
    - *Shallow cloning*
    - *Deep cloning*

6. Padrões de Projeto

- *Object-centric architecture*
  - Instâncias de objetos => propriedades
- *Property-centric architecture*
  - Propriedades => Instâncias de objetos

7. Input

7. Input

- ➔ Como receber eventos?
  - Sistema operacional
  - *Callbacks*
  - *Event polling*

7. Input

- ➔ *Callback*
  - Implementação de funções especializadas em receber certos parâmetros
    - Tecla, mouse, controle
  - Adicionado referências a essas funções

7. Input

Key input

If you wish to be notified when a physical key is pressed or released or when it repeats, set a key callback.

```
glfwSetKeyCallback(window, key_callback);
```

The callback function receives the **keyboard key**, platform-specific scancode, key action and **modifier bits**.

```
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    if (key == GLFW_KEY_E && action == GLFW_PRESS)
        activate_airship();
}
```

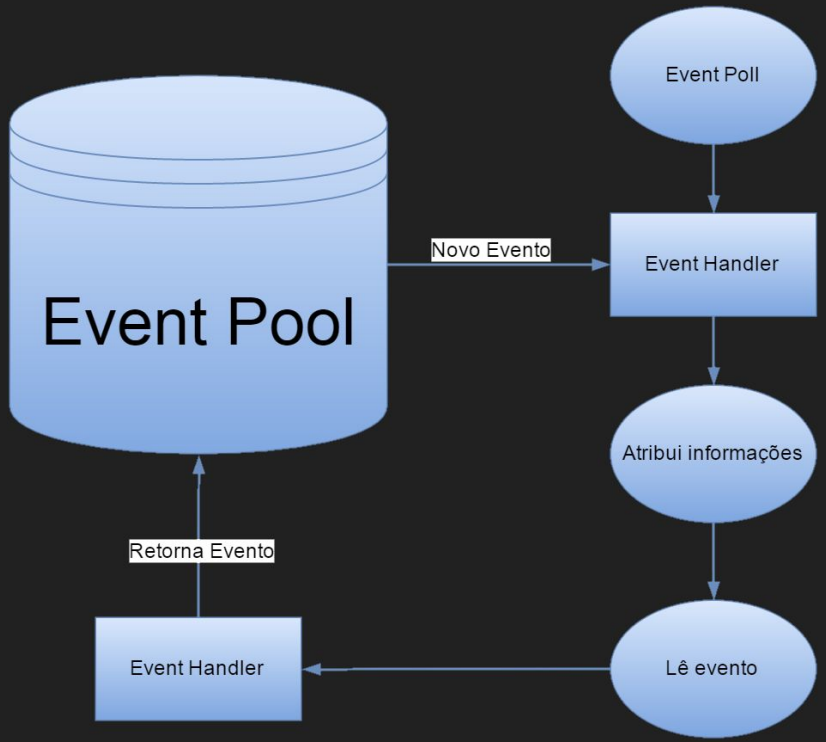
7. Input

- *Event Polling*
  - Mensagens ou eventos
  - Armazenados temporariamente
  - Laço passa capturando todos os eventos

7. Input

```
int main( int argc, char* args[] )
{
    if( !init() )
    {
        printf( "Failed to initialize!\n" );
    }
    else
    {
        bool quit = false;

        SDL_Event e;

        while( !quit )
        {
            while( SDL_PollEvent( &e ) != 0 )
            {
                if( e.type == SDL_QUIT )
                    quit = true;
            }
            SDL_UpdateWindowSurface( gWindow );
        }
    }

    close();

    return 0;
}
```

7. Input


Event Poll
Event Pool
Novo Evento
Event Handler
Atribui informações
Retorna Evento
Event Handler
Lê evento

Dúvidas?

Referências

Referências

[1] Jason Gregory-Game Engine Architecture-A K Peters (2009)
[2] Game Coding Complete, Fourth Edition (2012) - Mike McShaffry, David Graham
[3] David H. Eberly 3D Game Engine Architecture Engineering Real-Time Applications with Wild Magic The Morgan Kaufmann Series in Interactive 3D Technology 2004
[4] http://gameprogrammingpatterns.com/
[5] http://gafferongames.com/
[6] http://docs.unity3d.com/Manual/index.html
[7] http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/
[8] https://en.wikipedia.org/wiki/Software_design_pattern
[9] https://www.youtube.com/user/BSVino/videos
[10] https://www.youtube.com/user/thebennybox/videos
[11] https://www.youtube.com/user/GameEngineArchitects/videos
[12] https://www.youtube.com/user/Cercopithecan/videos
[13] http://www.glfw.org/docs/latest/input_guide.html
[14] http://lazyfoo.net/tutorials/SDL/index.php
[15]
[16]
[17]