

Motor de Jogos e Arquitetura

Arquitetura e game loop

Slides por:

Gustavo Ferreira Ceccon (gustavo.ceccon@usp.br)

Adaptados por:

Fabrício Guedes Faria (fabricao.guedes.faria@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

Este material possui licença CC By-NC-SA. Mais informações em:
<https://creativecommons.org/licenses/by-nc-sa/4.0/>



Objetivos

- Conceitos básicos de um motor de jogos
- Arquitetura e estruturas de um jogo
- Game loop
- Modelos de programação
- Mostrar exemplos na Unity



Sneak peek - Unity reel

Unity GDC 2016 Game Showreel:

<https://www.youtube.com/watch?v=8lWpnvNxs8k>

Unity at GDC 2018 Sizzle Reel:

<https://www.youtube.com/watch?v=QCqTn3SbfA0>

Unity - Book of the Dead:

https://www.youtube.com/watch?v=DDsRfbfnC_A

Unity - Adam (Playlist):

<https://www.youtube.com/watch?v=GXI0l3yqBrA&list=PLX2vGYjWbI0Relv3zkQu-zbqYwdTxqZpS>



Índice

1. Introdução
2. Arquitetura e Estrutura
3. Game Loop
4. Game Object



1. Introdução



1. Introdução

O que é um jogo (*videogame*)?



1. Introdução

“Soft Real-Time Interactive Agent-Based Computer Simulation”

Jason Gregory, Arquitetura de Motor de Jogos



1. Introdução

→ Computer Simulation

- ◆ Simulação de um mundo virtual
- ◆ Modelos matemáticos e físicos do mundo real





Tom Clancy's The Division (2016)





Need for Speed (2015)



1. Introdução

→ Interactive Agent-Based

- ◆ Orientado a objetos, ou seja, tem características e comportamentos definidos
- ◆ Jogo interativo, que reage à entrada do jogador





Detroit: Become Human (2018)



1. Introdução

→ Real-Time

◆ 60 FPS = 16.666666 ms

◆ 30 FPS = 33.333333 ms

◆ 24 FPS = 41.666666 ms

◆ Tick vs Frame

→ Por que é melhor ter mais FPS?





The Witcher 3 (2015)



1. Introdução

→ Soft System

- ◆ Recuperável no caso de fps drop, por exemplo
- ◆ Ao contrário de Hard Systems, que podem ser sistemas críticos



1. Introdução

- O que é um motor de jogos (game engine)?
 - ◆ Estrutura fundamental, base de todo jogo
 - Pode conter partes específicas de gêneros de jogos
 - ◆ Contém os módulos essenciais, como gráficos, física, detecção de input e áudio



1. Introdução

→ História breve

- ◆ Arcades eram *hardware-specific* (1 jogo = 1 máquina)
- ◆ Os primeiros consoles tinham um cartucho para cada jogo
- ◆ Evoluiu para CDs, com maior capacidade de armazenamento
- ◆ Ao poucos foram aproveitando código comum entre jogos similares (Quake e outros FPS, RAGE)
- ◆ O mercado de engines começou a crescer (Unreal e Source)
- ◆ Unity, Unreal 4, CryEngine - novos modelos de negócio (open-source, porcentagem de lucro, licença mensal)



1. Introdução

→ O que oferece?

- ◆ Interface com o programador e designer (editor)
- ◆ Funções básicas como renderizar *mesh*, tocar sons, aplicar transformações etc., além de estruturar básicas que representam os objetos
- ◆ Exportação para múltiplas plataformas (geralmente), além de editores (alguns casos)



1. Introdução

→ Por que estudar?

- ◆ Funcionamento do hardware e software, além do conhecimento de como funciona por trás do game design
- ◆ Aplicação de diversas áreas da computação, aprendidas num curso de Ciências da Computação
- ◆ Interessante para empresas de jogos AAA



1. Introdução

→ Vantagens

- ◆ Modularização, um código mais organizado e independente
- ◆ Reaproveitamento, podendo usar em múltiplos jogos
- ◆ Flexível, fácil mudança do código do jogo e adaptação
- ◆ Atender múltiplas plataformas, útil hoje em dia já que temos um grande número de usuários jogando em diferentes consoles, sistemas operacionais etc.



1. Introdução

→ Desvantagens

- ◆ Ficar preso à engine e o que ela oferece, levando à gambiarras muitas vezes
- ◆ Necessário se familiarizar com o funcionamento de uma engine nova (difícil se não houver documentação boa)
- ◆ Difícil lidar com bugs da engine (exceção: open-source)



1. Introdução

→ Por onde começar?

- ◆ Escolha plataformas, tanto do editor (se existir) e de exportação
- ◆ Escolha de paradigma e de linguagem, além de quais bibliotecas externas e ferramentas de desenvolvimento (version control, IDE)
- ◆ Estruturação e arquitetura da engine, além de que área cobre a sua engine
- ◆ Bottom-up development vs. Top-down development



1. Introdução

→ Exemplos

- ◆ [Quake Family](#) (Doom, Quake, Medal of Honor)
- ◆ [Unreal Family](#) (Unreal Tournament e Gears of War)
 - Atualmente uma das mais usadas pelas AAA
- ◆ [Source Engine](#) (muitos jogos da Valve)
- ◆ [Unity](#) (muitos jogos indies)
- ◆ [CryEngine](#) (Crysis, Far Cry)



2. Arquitetura e Estrutura



2. Arquitetura e Estrutura

Tudo o que a engine já oferece a você:

[Exemplo completo](#)



2. Arquitetura e Estrutura

- Estruturação geral da interface de uma engine
 - ◆ Game/World/Window - Simulação do jogo como seria na build final
 - ◆ Scene/Level - Área principal de montagem do jogo (níveis inteiros ou pedaços de mapa) e edição de objetos
 - ◆ Entity/Actor/Game Object - Qualquer elemento do jogo que possui um comportamento ativo

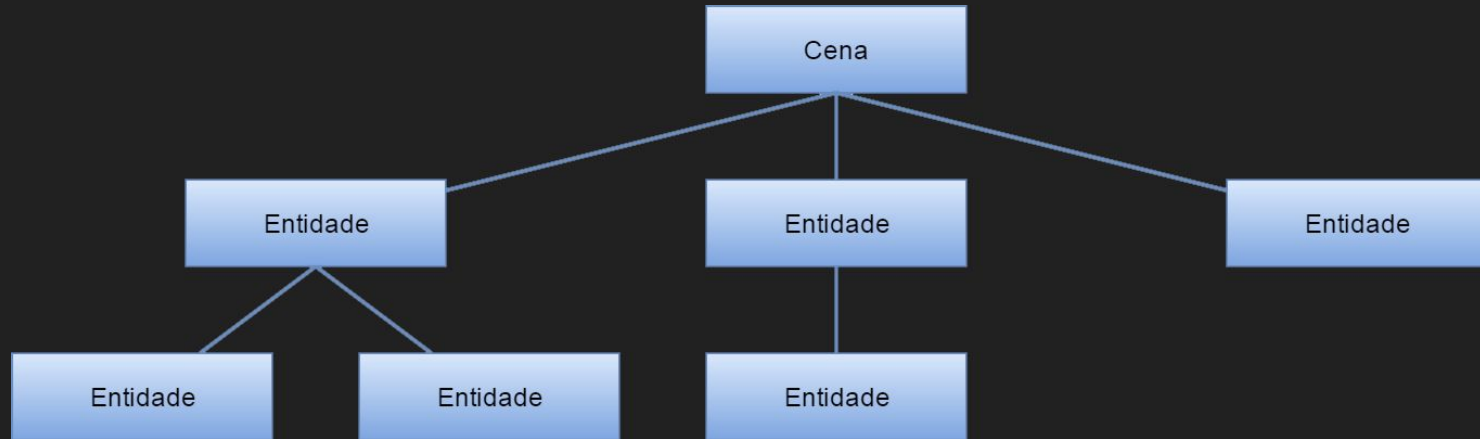


2. Arquitetura e Estrutura

- Hierarquia de uma cena em árvore
 - ◆ Útil para aplicar transformações relativas e globais
 - ◆ Usado principalmente na construção do level, pois facilita o posicionamento e interação
 - ◆ Jeito intuitivo de mexer com objetos



2. Arquitetura e Estrutura



UNITY TIME !!!! - Hierarquia



3. Game Loop



3. Game Loop

- Jogos eletrônicos são simulações de um mundo virtual, além disso sabemos que eles são programas de tempo real
 - ◆ Portanto, jogos estão diretamente entrelaçados com a noção de tempo
- Frames Per Second (FPS) é uma medida de quantos quadros conseguimos renderizar por segundo, mas por baixo é muito mais que isso



3. Game Loop

- Temos que mostrar pelo menos 24 frames por segundo, porém também temos que lidar com várias outras coisas
 - ◆ Audio, Input, AI, Networking etc.
- Todo frame temos iterações de processamento dessas coisas e o laço dessas iterações se chama Game Loop
 - ◆ A ordem e quantidade de processamento dedicado depende da escolha do game loop e da arquitetura do jogo.



3. Game Loop

- Vamos tentar montar o game loop:
- Objetivo:
 - ◆ Renderizar frames, que atendam expectativas do jogador
- Problemas:
 - ◆ O que processar?
 - ◆ Quanto processar?
 - ◆ Em que ordem processar?



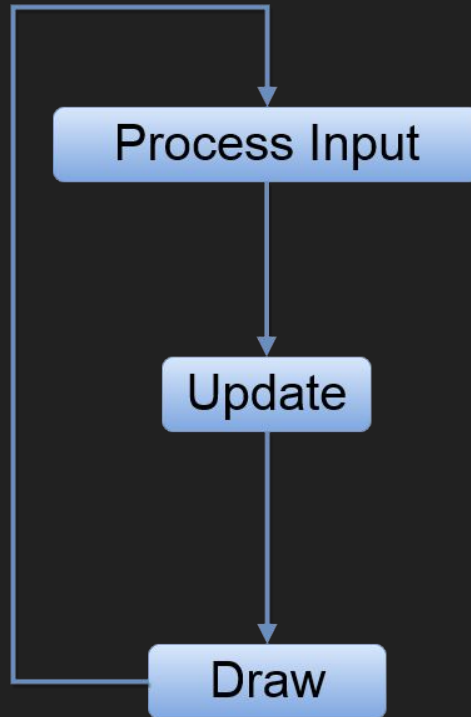
3. Game Loop

→ 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*
- ◆ *Frame skipping*



Game Loop - Simple



Game Loop - Simples

```
while (!done)
{
    input(); //atualiza estados
    update(); //sem param.
    draw(); //sem param.
}
```



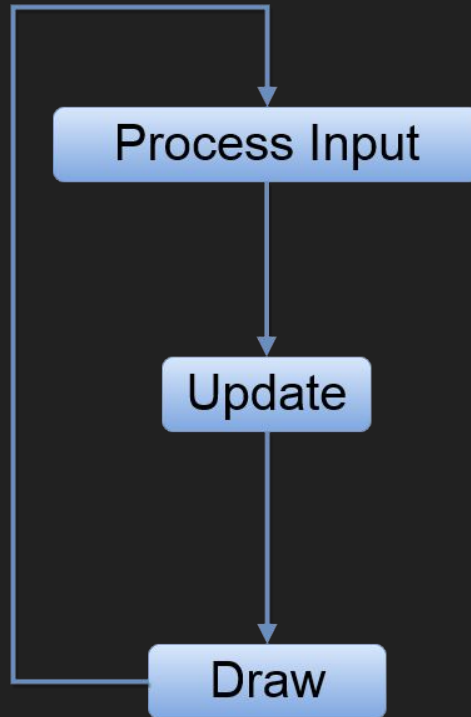
3. Game Loop

→ 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*
- ◆ *Frame skipping*



Game Loop - Simples com dt



Game Loop - Simples com *dt*

```
lastTime = now();  
while (!done)  
{  
    current = now();  
    dt = current - last;  
    last = current;  
    input(); //atualiza estados  
    update(dt); //passa param. Física baseada em dt  
    //Método de integração  
    draw(); //sem param.  
}
```



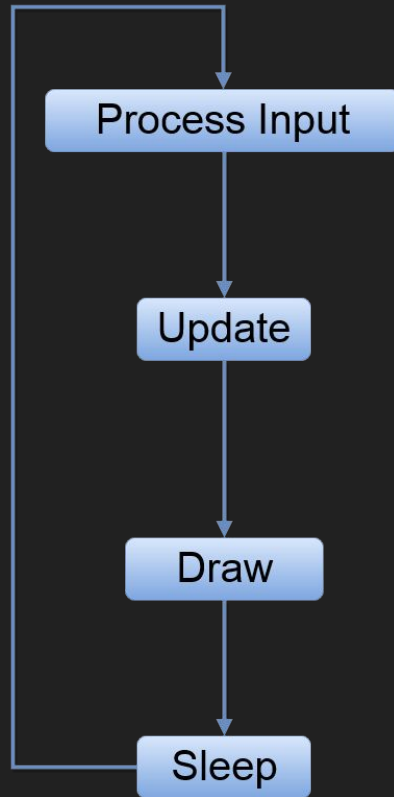
3. Game Loop

→ 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*
- ◆ *Frame skipping*



Game Loop - Simples com *dt* fixo

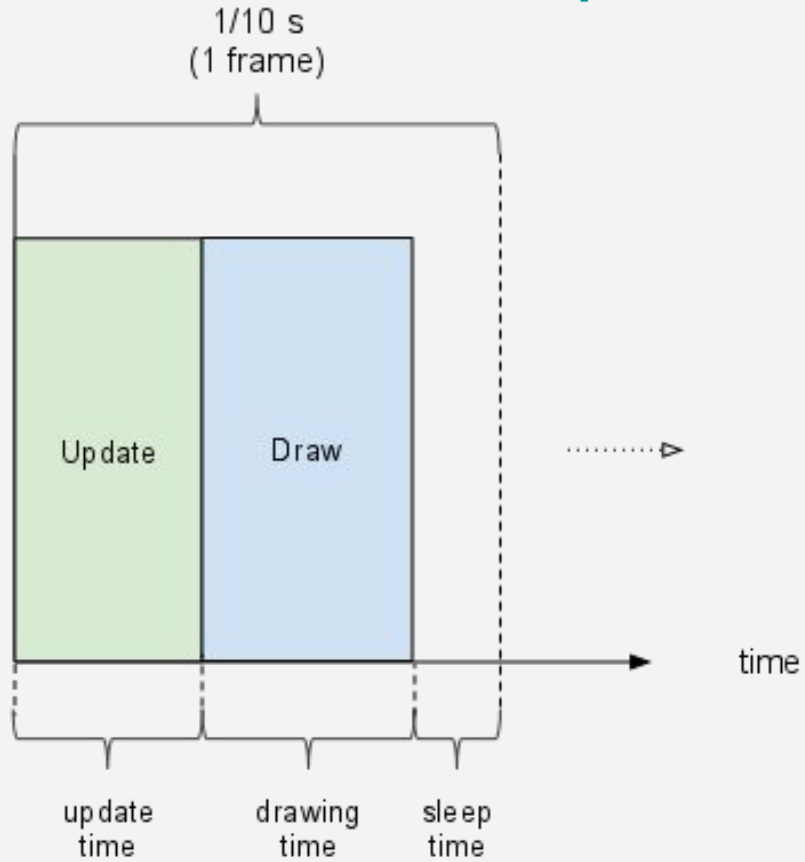


Game Loop - Simples com *dt* fixo

```
while (!done)
{
    start = now();
    input();
    update();
    draw();
    sleep(dt - (now() - start));
    //dt é fixo. now-start é o tempo do loop.
}
```



3. Game Loop



3. 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*
- ◆ *Frame skipping*

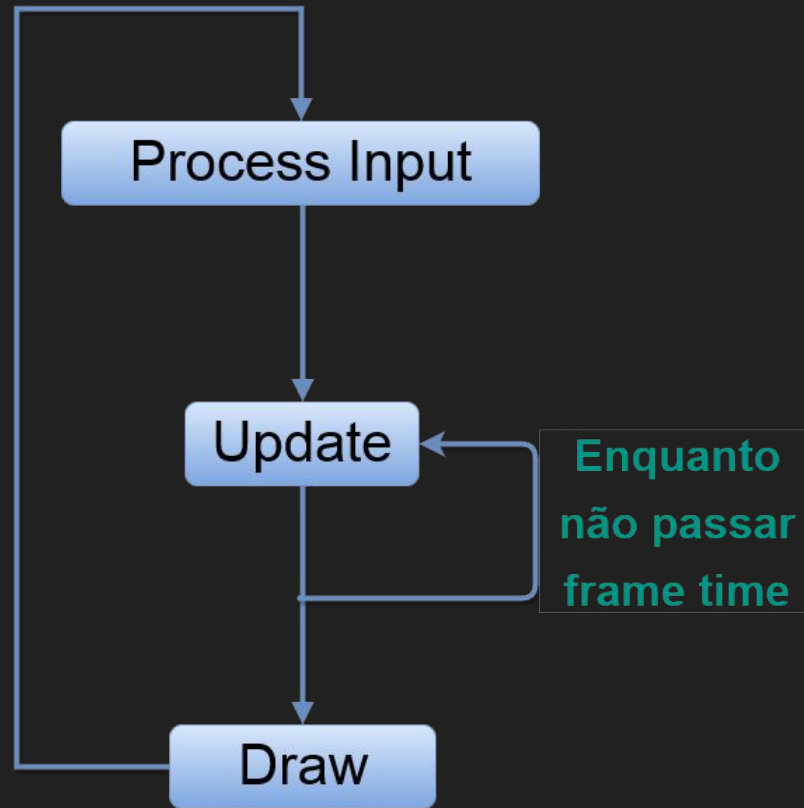


3. Game Loop

- Em alguns casos, podemos ter CPUs mais rápidas que GPUs.
- Neste caso, o Update será mais rápido que o Draw.
- Frame time > Update time
 - ◆ UPS ≠ FPS
- Para solucionar o problema, utilizamos catch-up



Game Loop - Catch-up simples



Game Loop - Catch-up simples

```
lastTime = now()
while (!done)
{
    currentTime = now()
    frameTime = currentTime - lastTime;
    lastTime = currentTime;
    while(frameTime > 0) \\Catch-up
    {
        delta = min(frameTime, dt);\\ Menor entre fixo e o restante
        update(delta);
        frameTime -= delta;
    }
    draw();
}
```



3. 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*
- ◆ *Frame skipping*

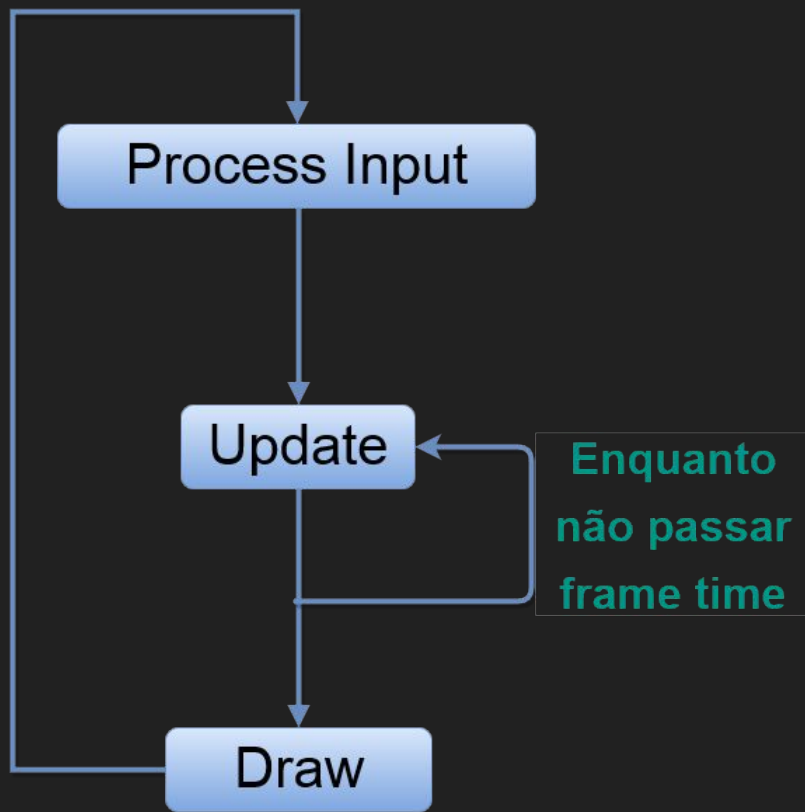


3. Game Loop

- *Catch-up* com extrapolação: atualiza de acordo com o tempo de *render* e extrapola o restante
 - ◆ Se um *draw* precisa ocorrer antes de um *update* terminar
 - O resultado entre os *updates* é extrapolado
- Interpolação: um ponto entre dois pontos conhecidos
 - ◆ $P' = (1 - \alpha) * P_0 + \alpha * P \quad = 0 \leq \alpha \leq 1$
- Extrapolação: interpolação entre um ponto conhecido e uma previsão



Game Loop - Catch-up com extrapolação



Game Loop - Catch-up com extrapolação

```
lastTime = now()
accumulator = 0;
while (!done)
{
    currentTime = now()
    frameTime = currentTime - lastTime;
    lastTime = currentTime;
    accumulator += frameTime;
    while(accumulator >= dt) \\Catch-up
    {
        update(dt);\\Fixo
        accumulator -= dt;
    }
    alpha = accumulator/dt;
    draw(alpha);
    //state = (1-alpha)*previous + alpha*current;
}
```



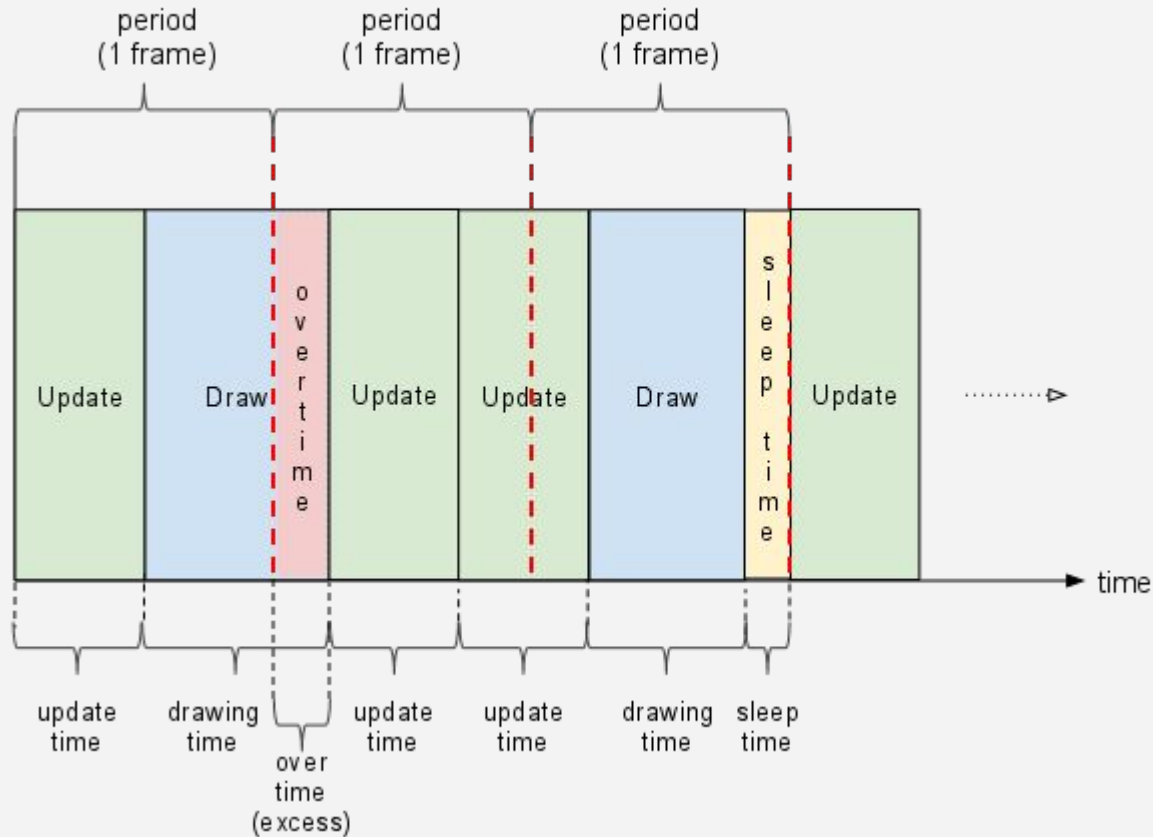
3. Game Loop

→ 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*
- ◆ **Frame skipping**



3. Game Loop

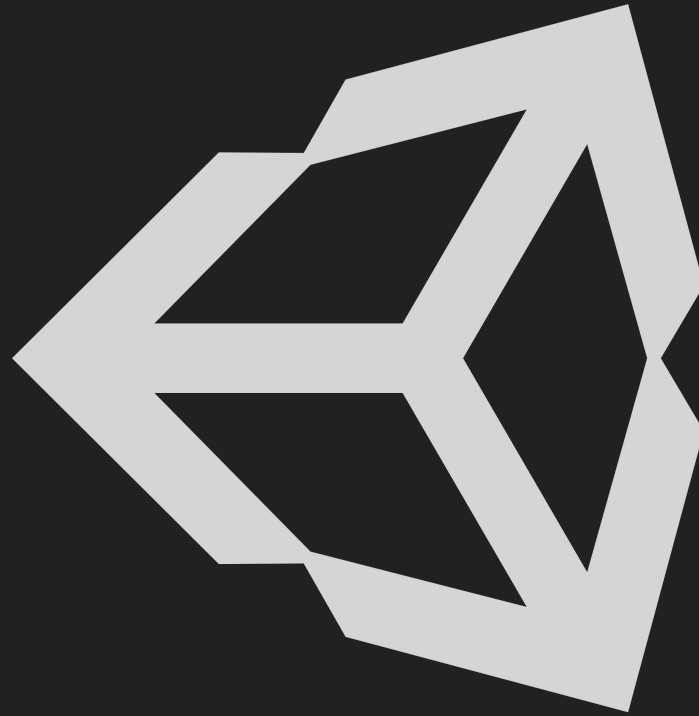


3. Game Loop

Exemplo Unity



UNITY TIME !!!! - Game Loop



4. Game Object



4. Game Object

→ Programação imperativa

- ◆ Simples e direto, sem muito problema na implementação
- ◆ Eficiente, porque é mais próxima de linguagem de máquina
- ◆ Uso de ponteiro de funções pode levar a bugs



4. Game Object

- Programação orientada a objeto
 - ◆ Classes cobrem tanto dados quanto comportamento
 - ◆ Pode se fazer uso de herança, polimorfismo etc.
 - ◆ Bom reaproveitamento de código e extensível
 - ◆ Número de classes pode subir exponencialmente, muita generalização pode aumentar a carga de trabalho



4. Game Object

→ Composição (Componentes)

- ◆ Adicionar pequenos comportamentos e atributos comuns em cada objeto invés de herdá-los
- ◆ Cada script representa um componente e cada objeto contém um vetor de componentes
- ◆ É possível representar todos scripts como uma matriz também



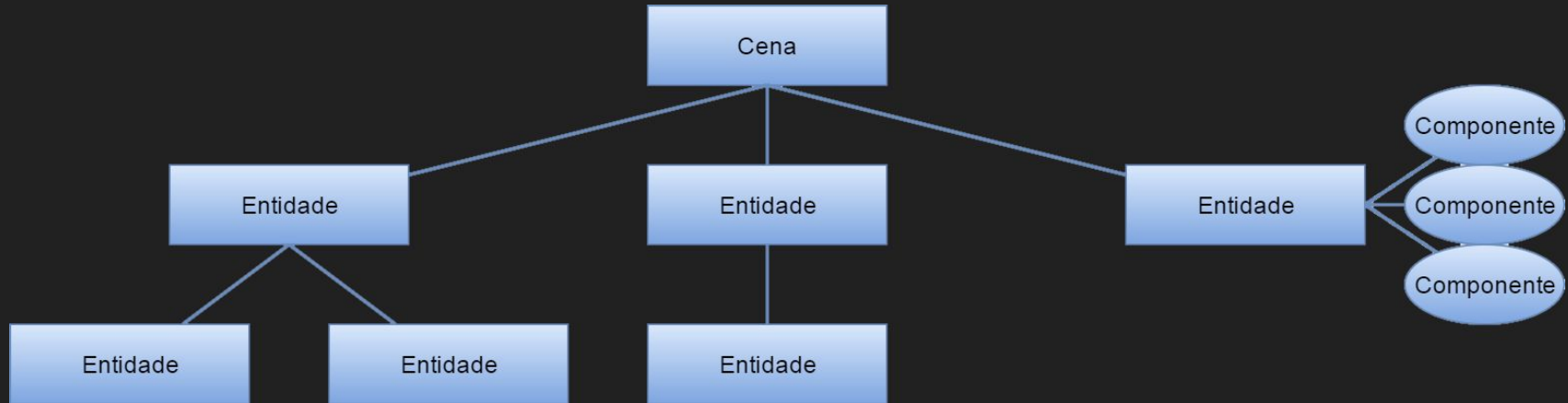
4. Game Object

→ Composição (Componentes)

- ◆ A dependência entre componentes e objetos pode complicar a execução dos scripts
 - Se um script depende de outro script, isso pode quebrar o paralelismo, uma das vantagens de usar composição
 - A comunicação entre objetos e scripts fica pesada
- ◆ Nem sempre é trivial separar as funcionalidades
- ◆ Pode ser overkill para jogos pequenos o suficiente



4. Game Object

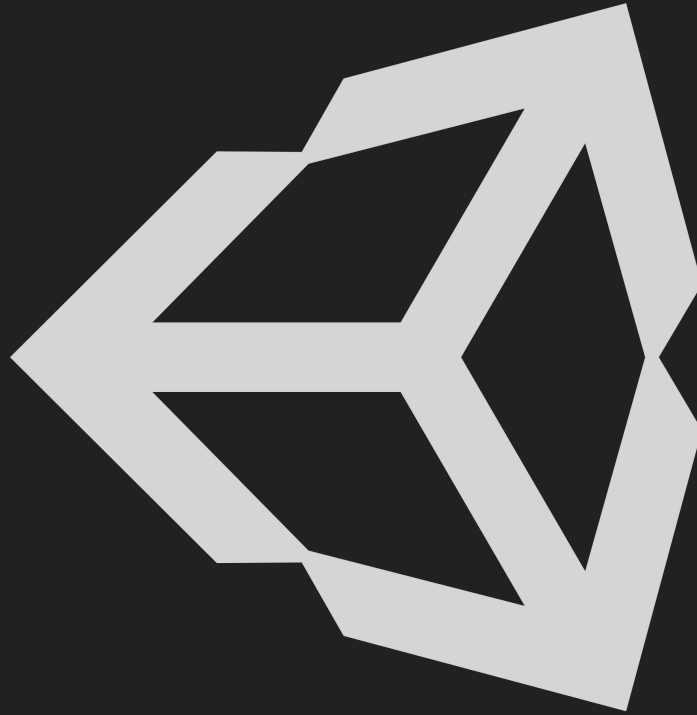


4. Game Object

- Melhor de dois mundos (híbrido)
 - ◆ Usar pouca herança (árvore pequena) e o suficiente de composição (para as funcionalidades) para facilitar o desenvolvimento
 - ◆ Maior parte das engines usam



UNITY TIME !!!! - Componentes



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/Cercopithecian/videos>
- [13] http://www.glfw.org/docs/latest/input_guide.html
- [14] <http://lazyfoo.net/tutorials/SDL/index.php>
- [15]
- [16]
- [17]

