Motor de Jogos e Arquitetura

Arquitetura e game loop de uma game engine

Slides por:
Gustavo Ferreira Ceccon (TEDJE - FoG - ICMC, 2017)

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
- Por onde começar
- Arquitetura e estruturas de um jogo
- Game loop
- Modelos de programação
- Básico de Unity

Índice

1. Introdução

1. Introdução

O que é um jogo?

1. Introdução

Soft Real-Time Interactive Agent-Based Computer Simulation

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 à 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
  - Tempo limitado para processar, atualizar e mostrar o resultado para o jogador

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

Bug Report (RuneScape e League of Legends

1. Introdução

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

1. Introdução

- ➔ História breve
  - Arcades e consoles eram hardware specific
  - Ao poucos foram aproveitando código comum entre jogos similares (Quake e outros FPS)
  - O mercado de engines começou a crescer (Unreal e Source)
  - Unreal 4, CryEngine e novos modelos de negócio, open-source e porcentagem de lucro

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

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
  - Não extensível, podendo não atender todas as necessidades, tornando difícil desenvolvimento

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

Exemplo completo

2. Arquitetura e Estrutura

- ➔ Muitas vezes separadas em módulos ou camadas
  - Dificuldade de separar os módulos: existe uma grande quantidade de intersecções
    - Camadas de dependência: nível de abstração e de proximidade com hardware
  - Genericamente falando: graphics, audio, physics, networking, além do core, que é o fundamento para o jogo em si
  - Core pode conter coisas específicas de plataforma

2. Arquitetura e Estrutura

- Estruturação geral de uma engine
  - Game/World/Window
  - Scene/Level
  - Entity/Actor/Game Object

2. Arquitetura e Estrutura

- Game/World/Window
  - Cuida da inicialização e término, em algumas plataformas e bibliotecas existe um trabalho exaustivo
  - Engloba as cenas (uma ou mais delas) e oferece diversas funções para manipulamento delas e da janela
  - Pode cuidar do input system (keyboard, mouse, joystick)

2. Arquitetura e Estrutura

Exemplo: Win32 API

2. Arquitetura e Estrutura

- ➔ Scene/Level
  - ◆ Cuida dos objetos, execução dos scripts e do level em si
  - ◆ Representa o level e depende do game design
    - Pode ser parte dele, quando o universo é muito grande (GTA V) ou ele inteiro (fase do Mario)
  - ◆ Implementa alguma estrutura de dados para guardar os objetos, podendo ser uma árvore, por exemplo

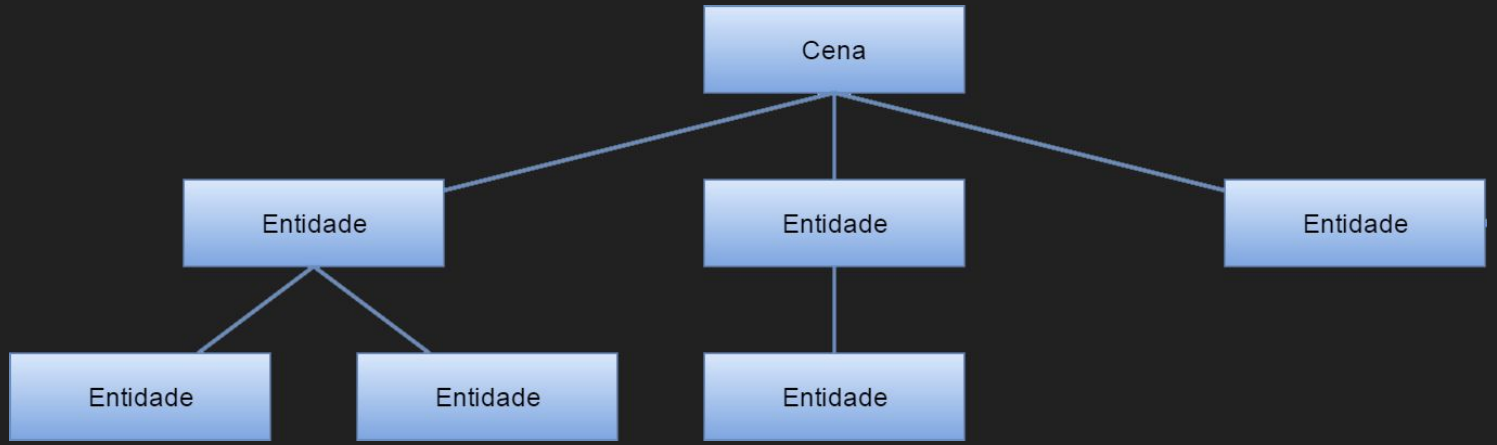
2. Arquitetura e Estrutura

- ➔ Entity/Actor/Game Object
  - ◆ Representam os objetos do jogo (desde props até jogador)
  - ◆ Tem atributos, mecânicas e comportamentos

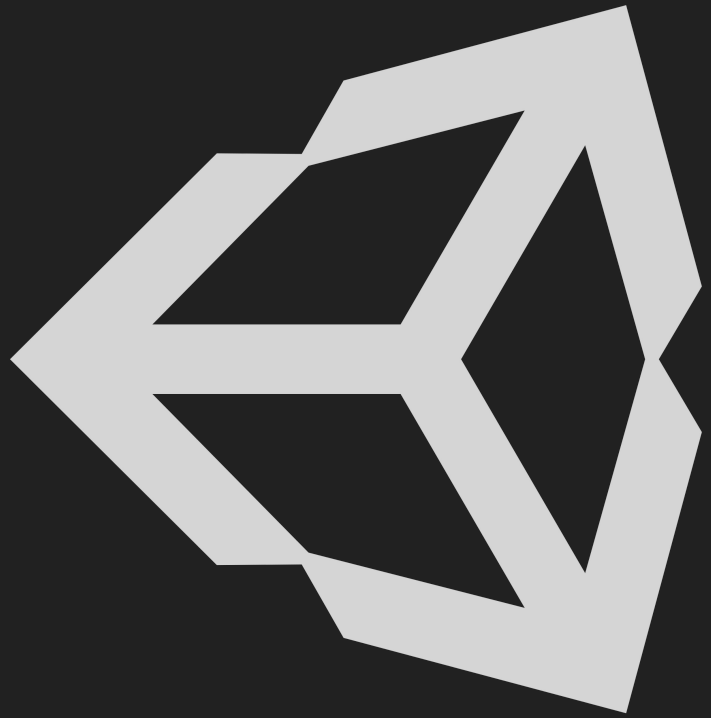
2. Arquitetura e Estrutura

- 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 !!!! - Tetris



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 monte de 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

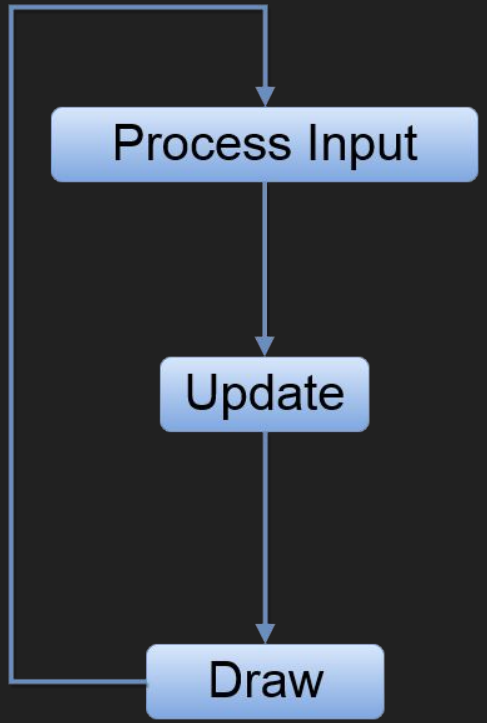
3. Game Loop

- 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 - Simples



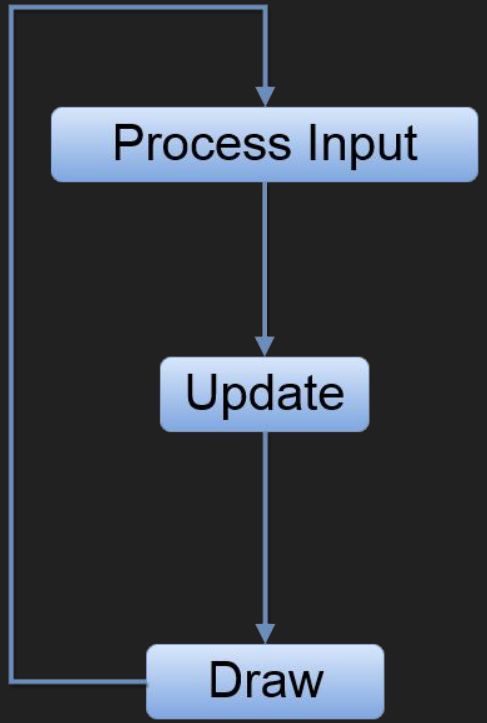
Game Loop - Simples

```
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*



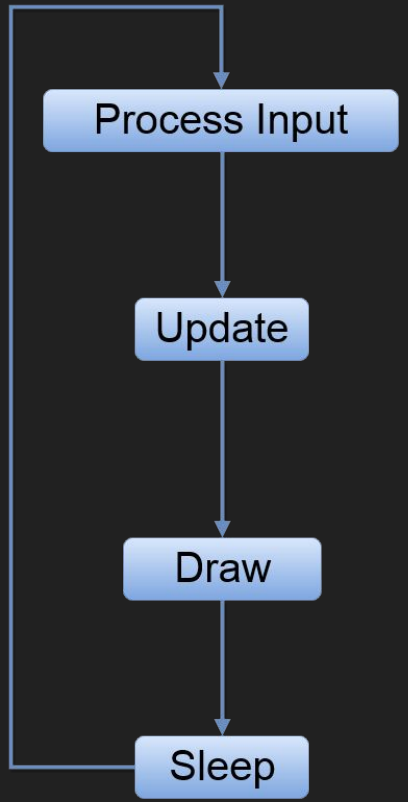
Game Loop - Simples com *dt*

```
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

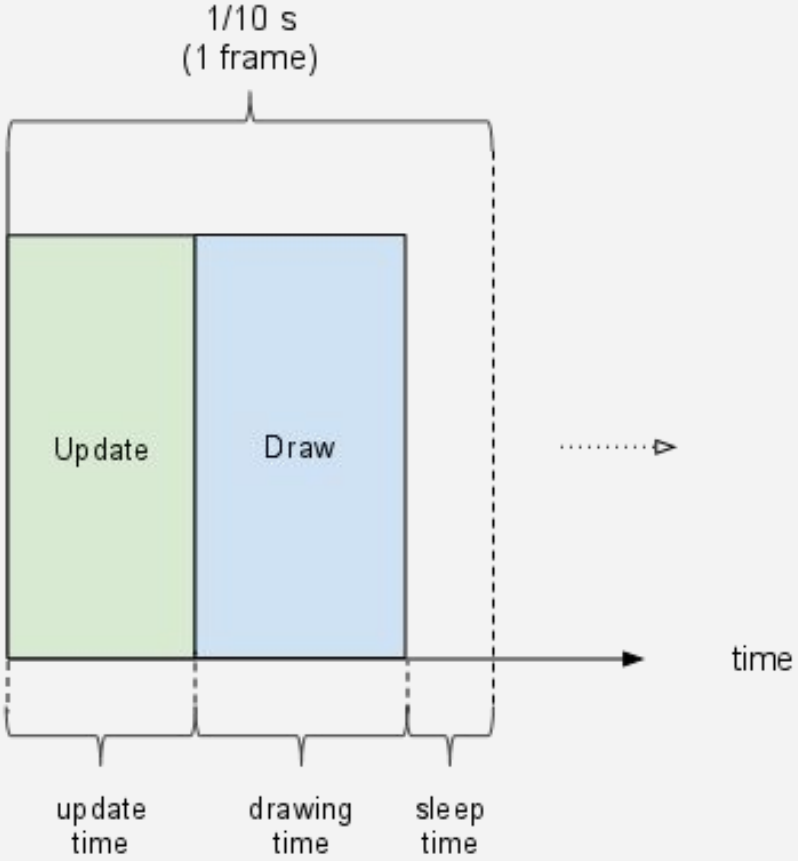
Game Loop - Simples com *dt* fixo



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


1/10 s
(1 frame)
Update
Draw
time
update time
drawing time
sleep time

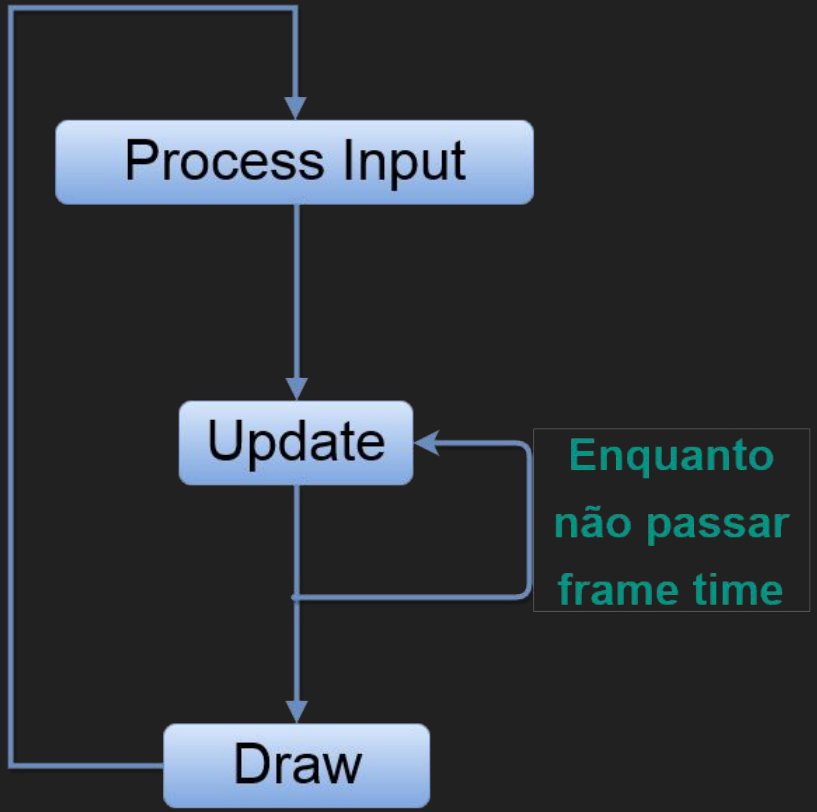
3. Game Loop

- 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();
}
```

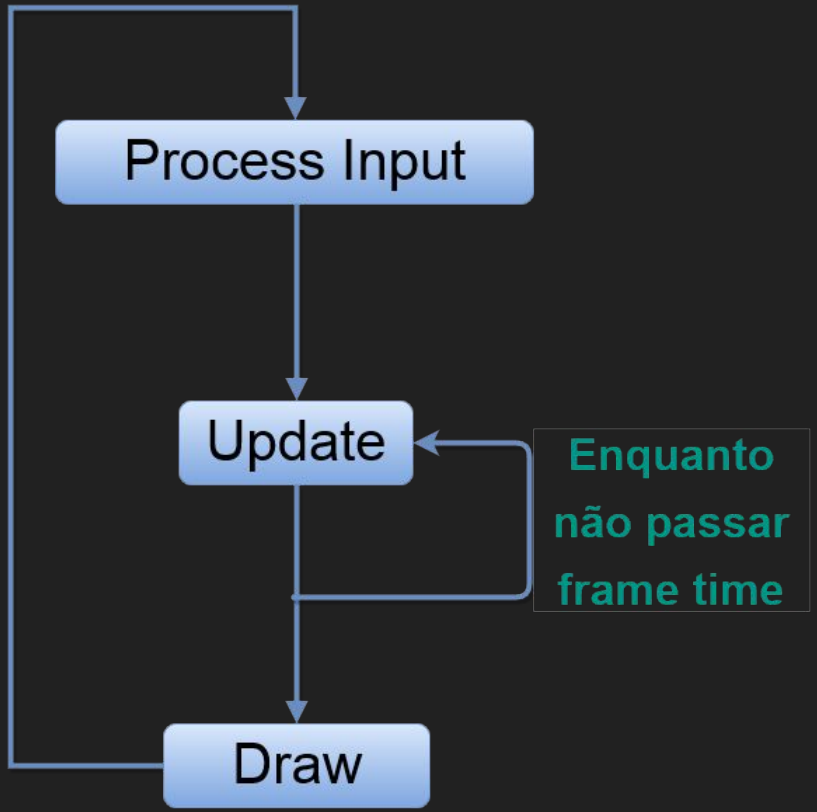
3. Game Loop

- 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
  - $P' = (1 - \alpha)*P_0 + \alpha*P \quad =0 <= \alpha <= 1$
- Extrapolação: interpolação entre um ponto conhecido e uma previsão

Game Loop - Catch-up com extrapolação



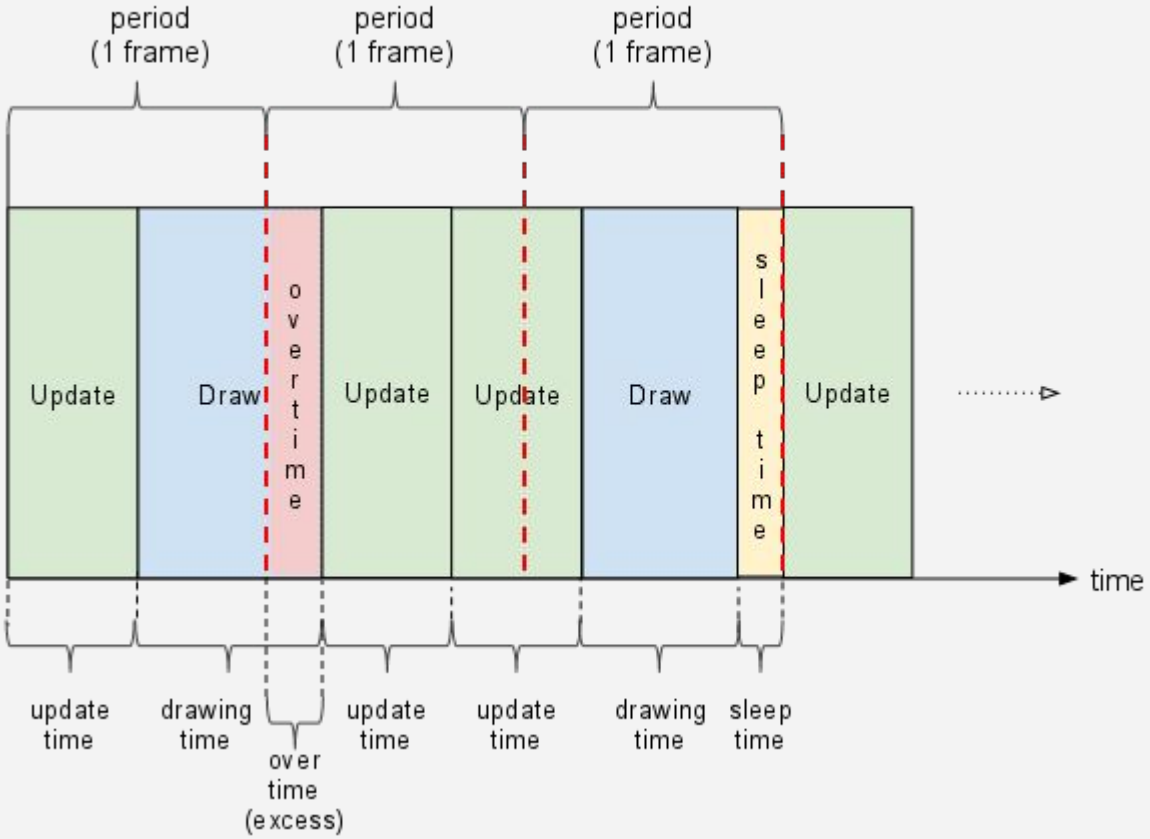
Game Loop - Catch-up com extrapolação

```
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

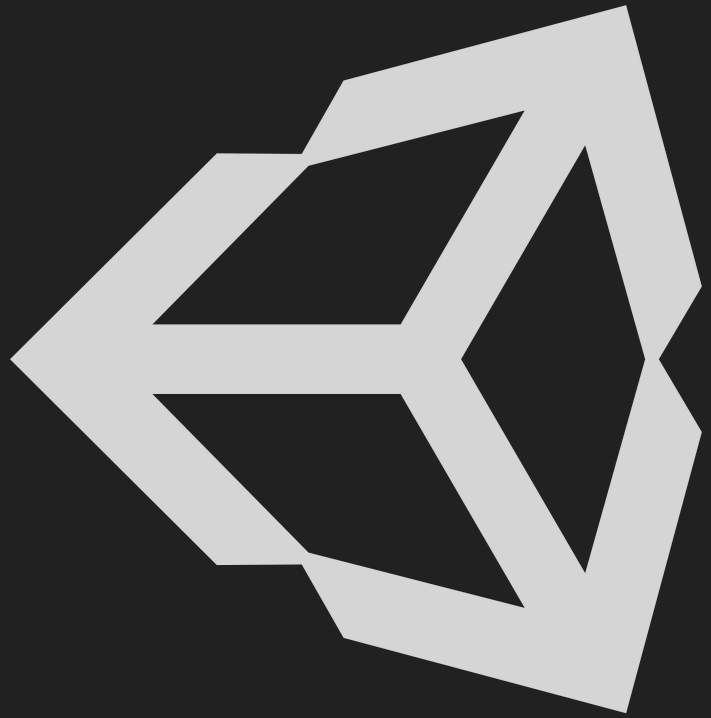
3. Game Loop


period
(1 frame)
period
(1 frame)
period
(1 frame)
Update
Draw
overtime
Update
Update
Draw
sleep time
Update
time
update time
drawing time
over time (excess)
update time
update time
drawing time
sleep time

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

- Herança
  - Útil quando é possível generalizar o objeto, para compartilhar atributos e métodos
  - Pode ficar complicado separar as diferenças e juntar as semelhanças nos nós da árvore de herança

4. Game Object

- Herança
  - Quando há a necessidade de juntar as semelhanças, as funções sobem na árvore, sobrecarregando as classes pai
    - Classes filho começam a ficar irrelevantes
  - Quando há necessidade de separar as diferenças, as funções descem na árvore, tornando o código esparso
    - Classes pai começam a ficar irrelevantes

4. Game Object

- Exemplos de herança
  - Móvel e Colecionável
  - Raça e Classe

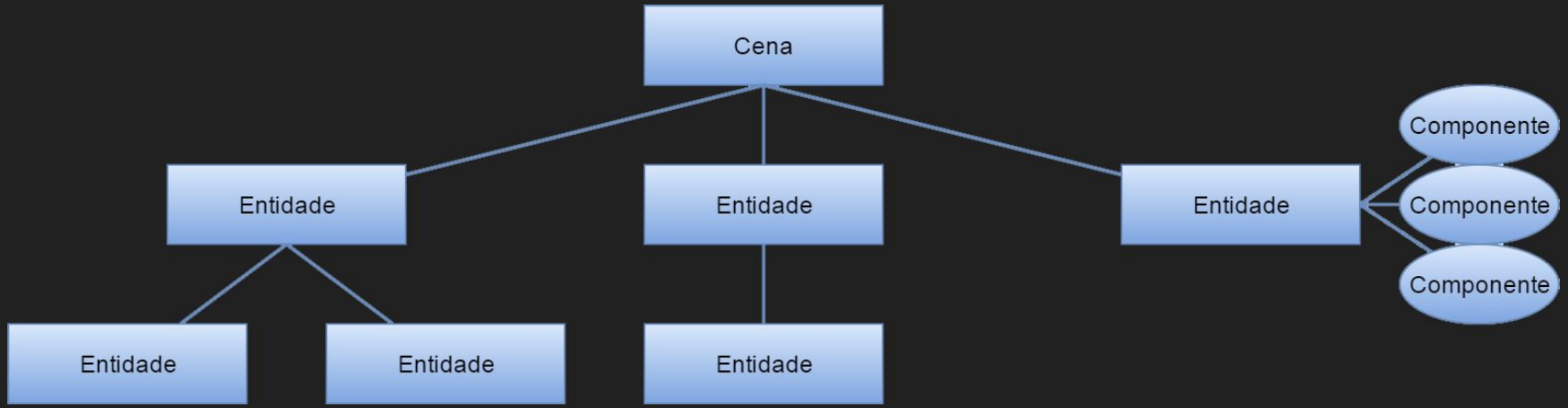
4. Game Object

- Composição
  - 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
  - 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



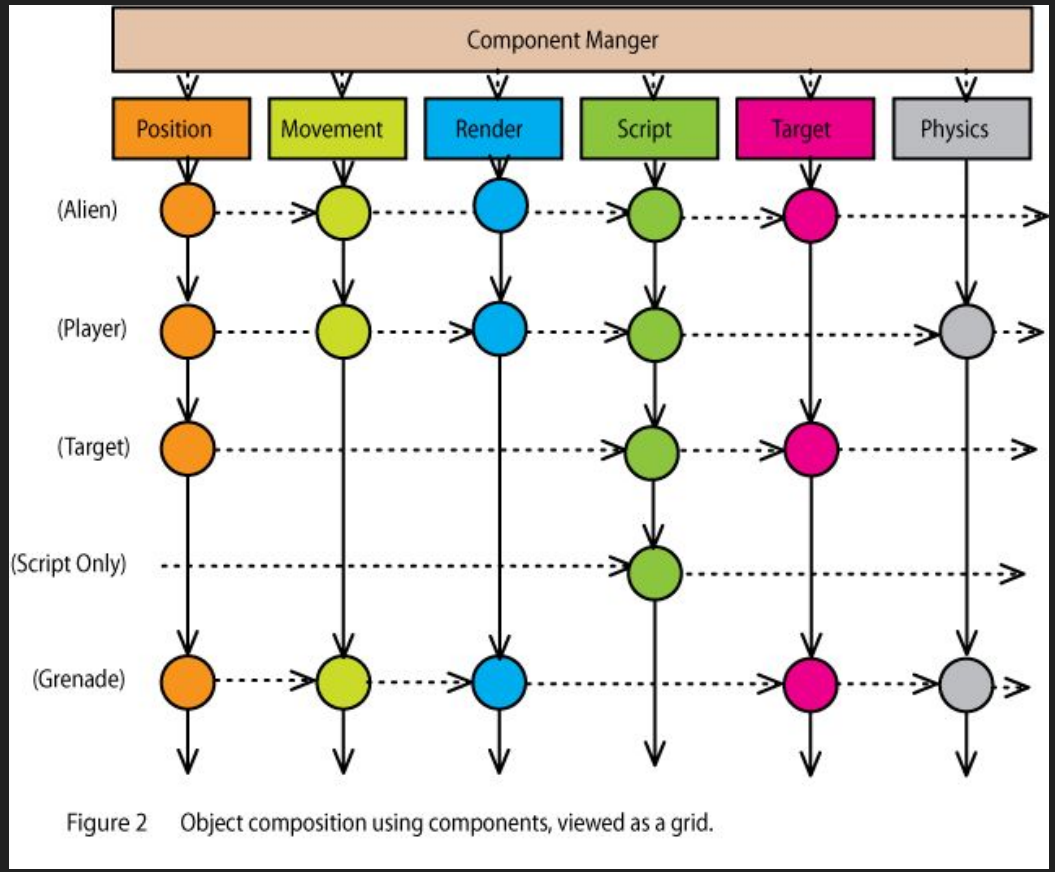


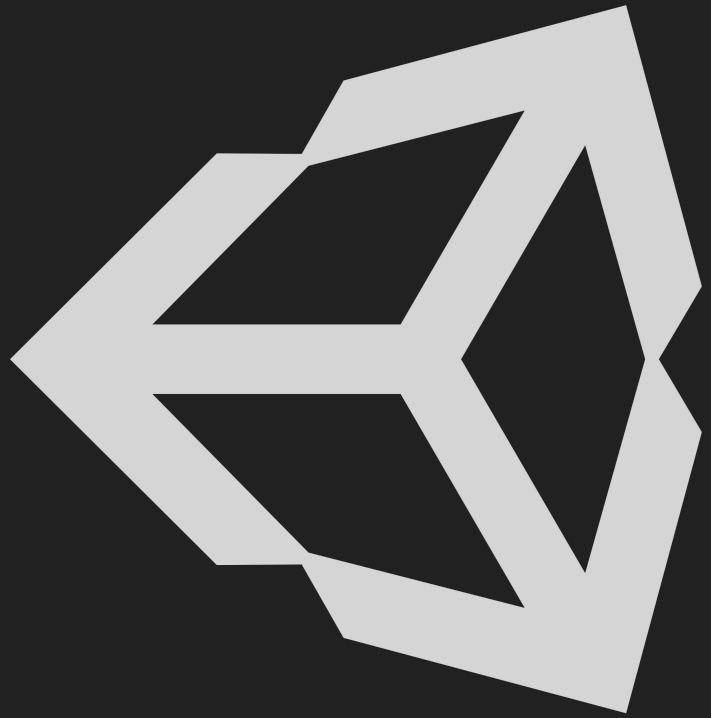
Figure 2 Object composition using components, viewed as a grid.

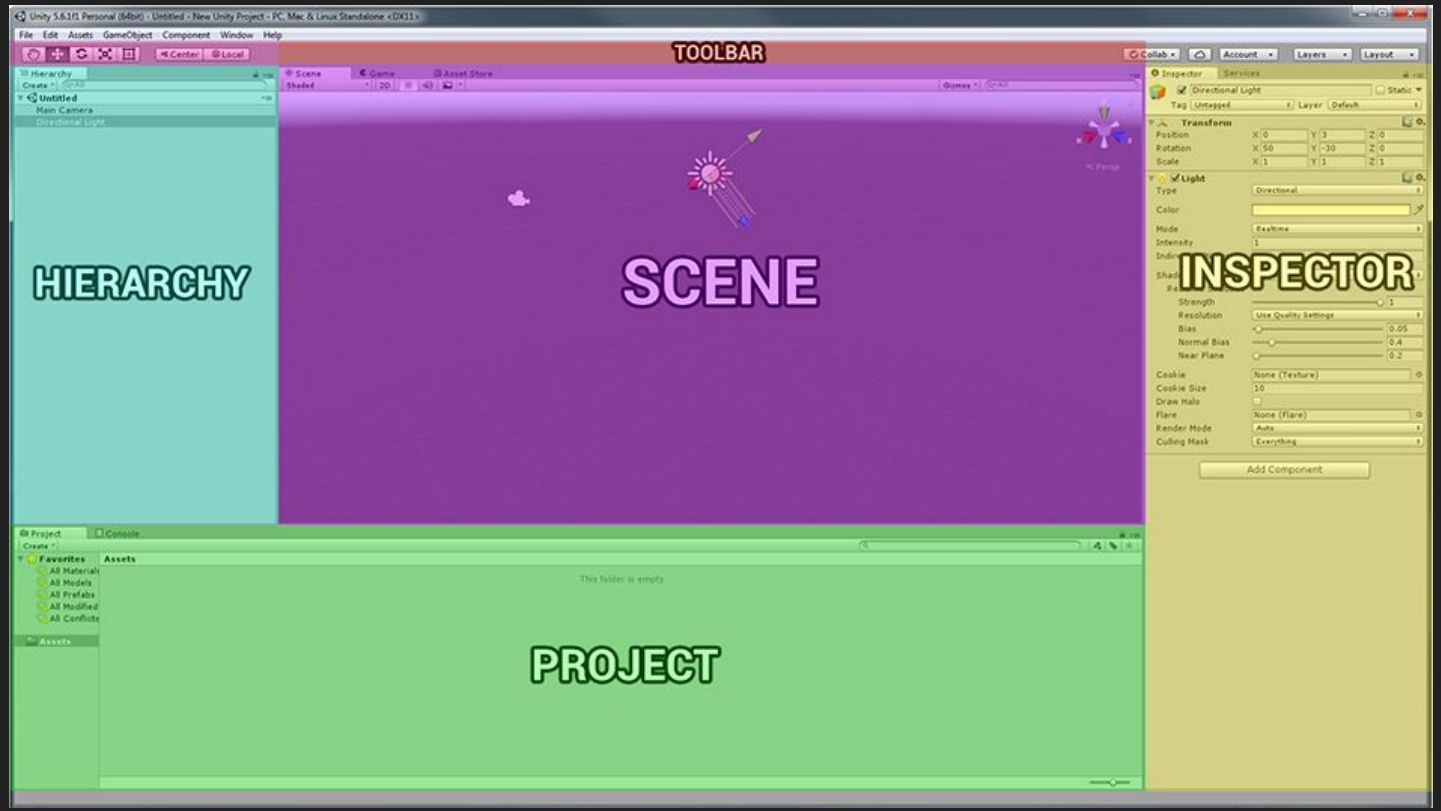
Matriz esparsa de componentes

4. Game Object

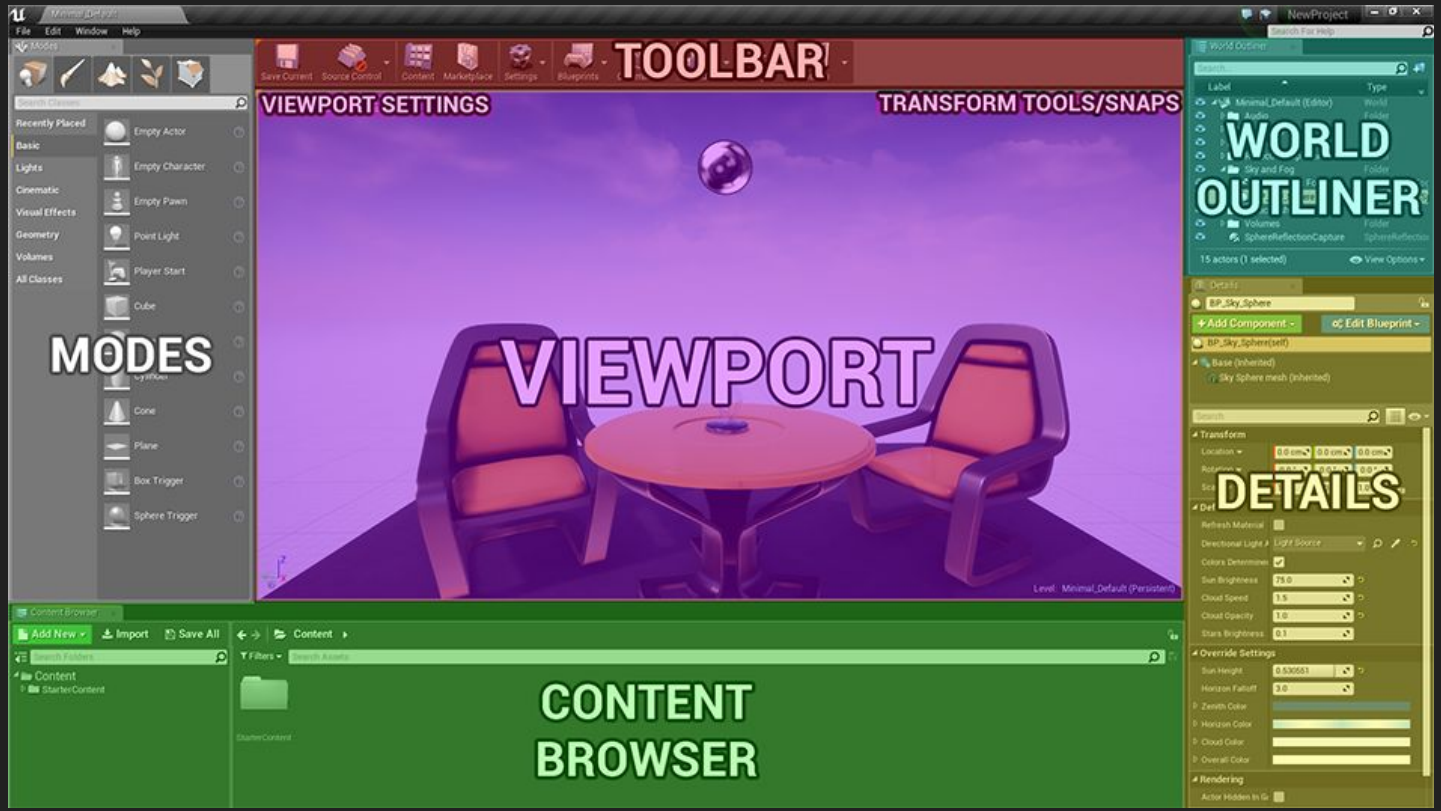
- ➔ Melhor de dois mundos (híbrido)
  - Usar pouca herança (árvore pequena) e o suficiente de composição (para as funcionalidades) para facilitar o desenvolvimento
  - Maior parte das engines usam

UNITY TIME !!!! - Planetas

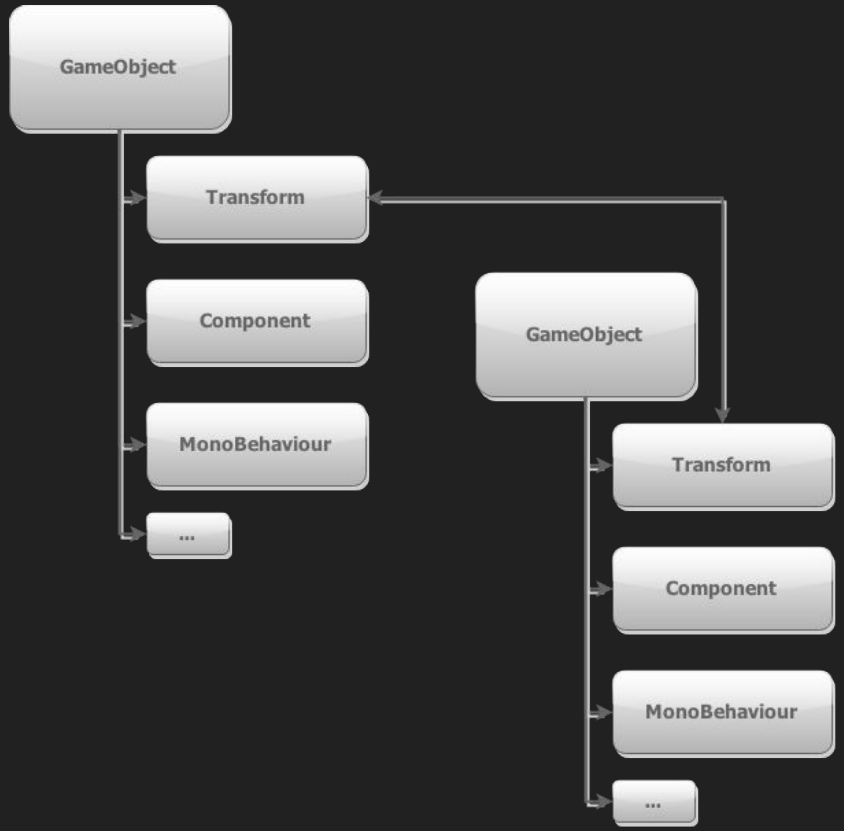




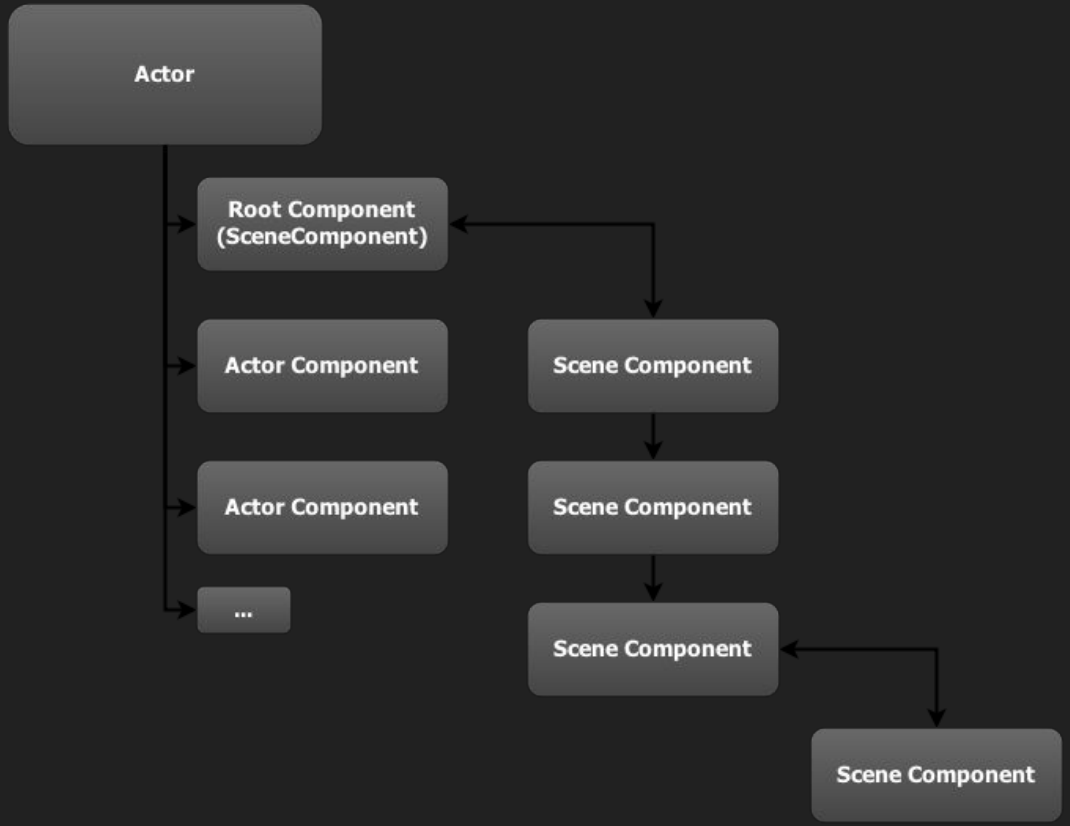
Unity



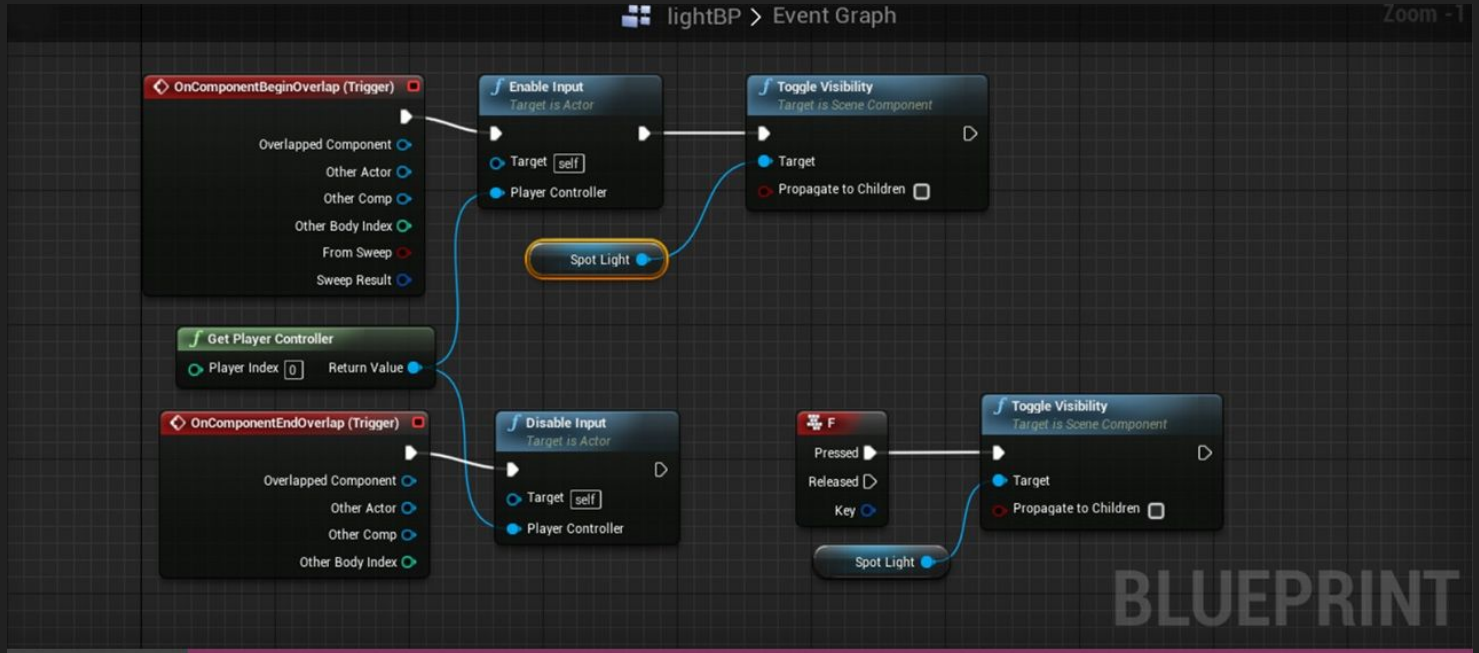
[Unreal Engine]



Game Object na Unity

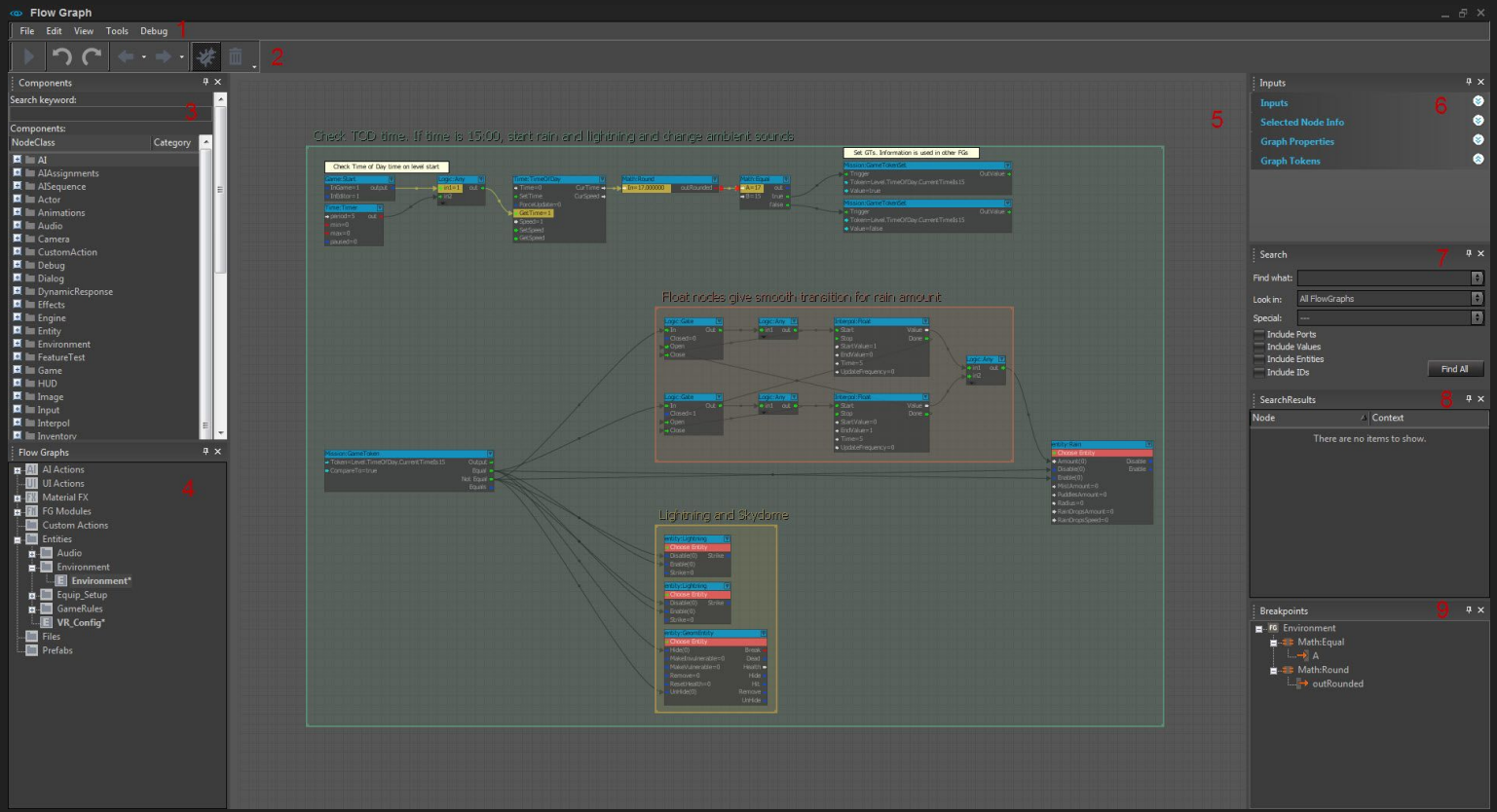


Actor na Unreal



TUTORIAL SERIES

Blueprint na Unreal Engine



Flow Graph na Cry Engine

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]