DCC192

2025/2

UFMG

Desenvolvimento de Jogos Digitais

A6: Game Objects

Prof. Lucas N. Ferreira

Logística

Avisos

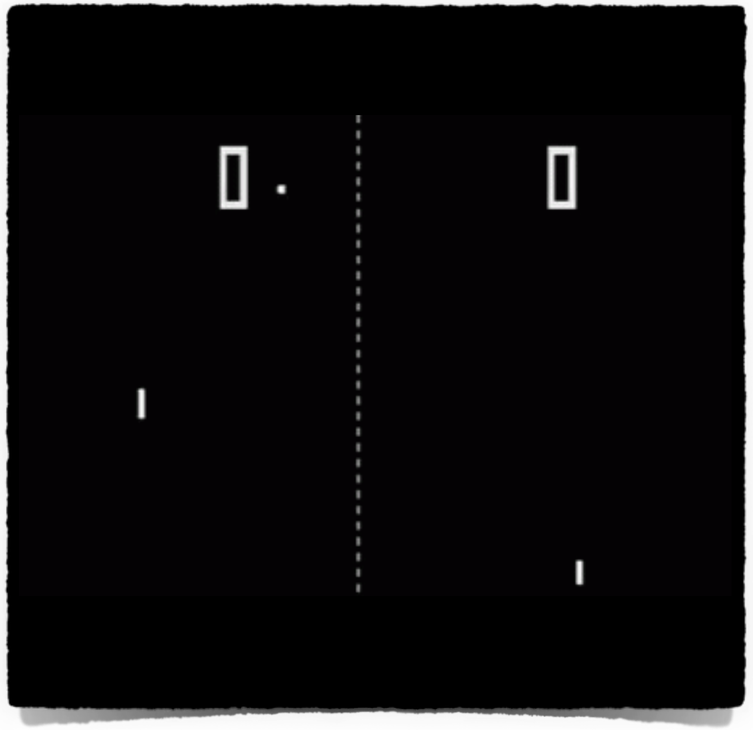
- O **TP1: Pong** foi publicado!
  - Nota: 10% da nota dos TPs!
  - Entrega: 15/09 (11:59h)

Plano de Aula

- Game Update: Atualizando Objetos do Jogo
- Objetos Dinâmicos , Estáticos e Gatilhos
- Modelagem de Objetos
  - Modelo de hierarquia de classes
  - Modelo de componentes
  - Modelo híbrido

Game Loop

Na última aula vimos que um jogo é um laço (*loop*) que repete as seguintes três funções: **`ProcessInput()`**, **`Update()`** e **`GenerateOutput()`**

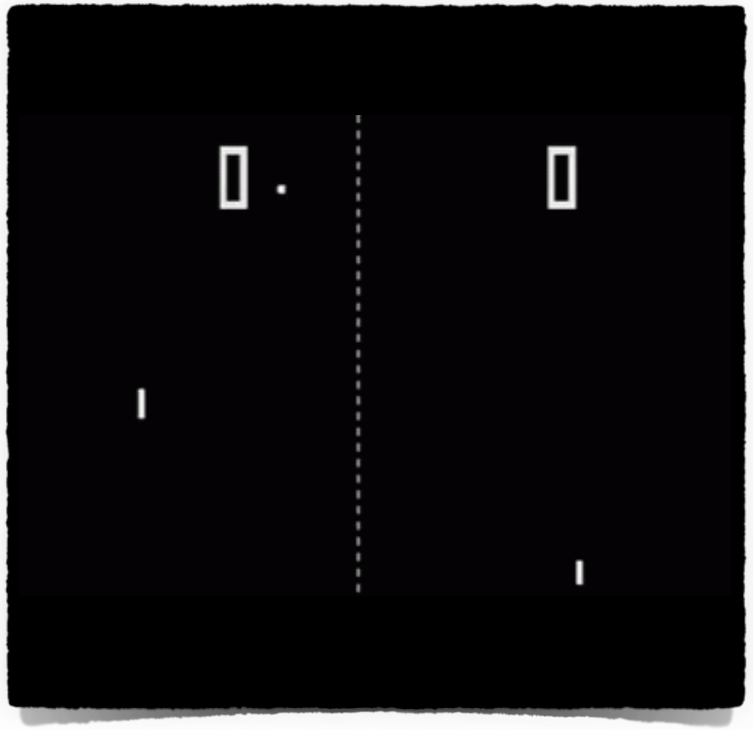


```
1. while Game is running
2.     ProcessInput()
3.     UpdateGame()
4.     GenerateOutput()
```

A etapa de **`Update`** implementa toda a "lógica" do jogo (mecânicas, regras, objetivos, ...)

Game Update

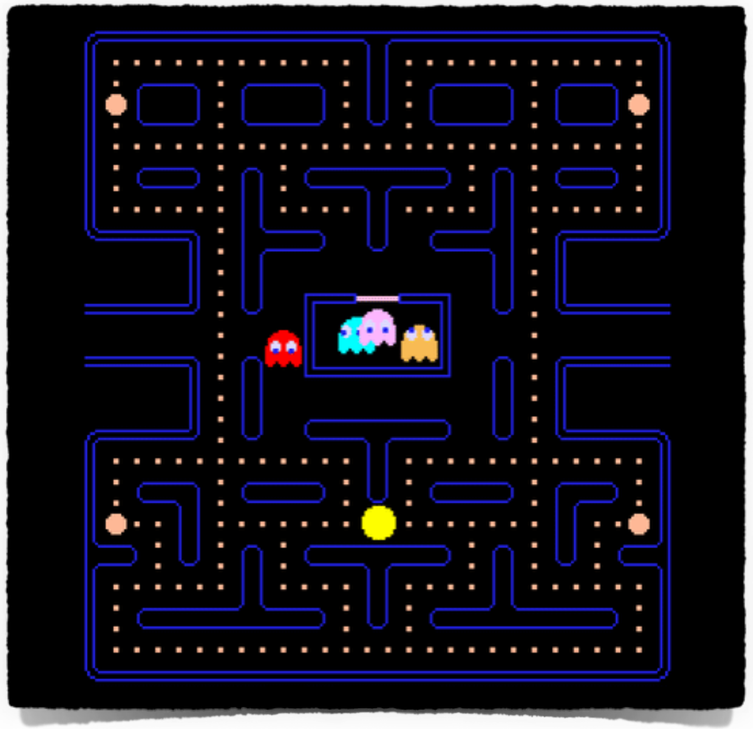
Em jogos simples, como o Pong, é razoável implementar o **`Update`** do jogo em uma única função, pois a sua lógica de atualização é relativamente simples :



```
while !goal:
  // Atualizar posição das raquetes
  update player1.position based on input1
  update player2.position based on input2

  // Atualizar posição da bola
  update ball.velocity based on colision
  ball.position += ball.velocity

  // Verificar se houve gol
  if ball.position > LEFT_BOUND or
     ball.position < RIGHT_BOUND
        goal = true

        // Atualizar placar
        ...
```

Game Update

Para jogos um pouco mais complexos, como o PacMan, isso já se torna inviável, pois a lógica do jogo é complexa demais para ser escrita em uma única função:

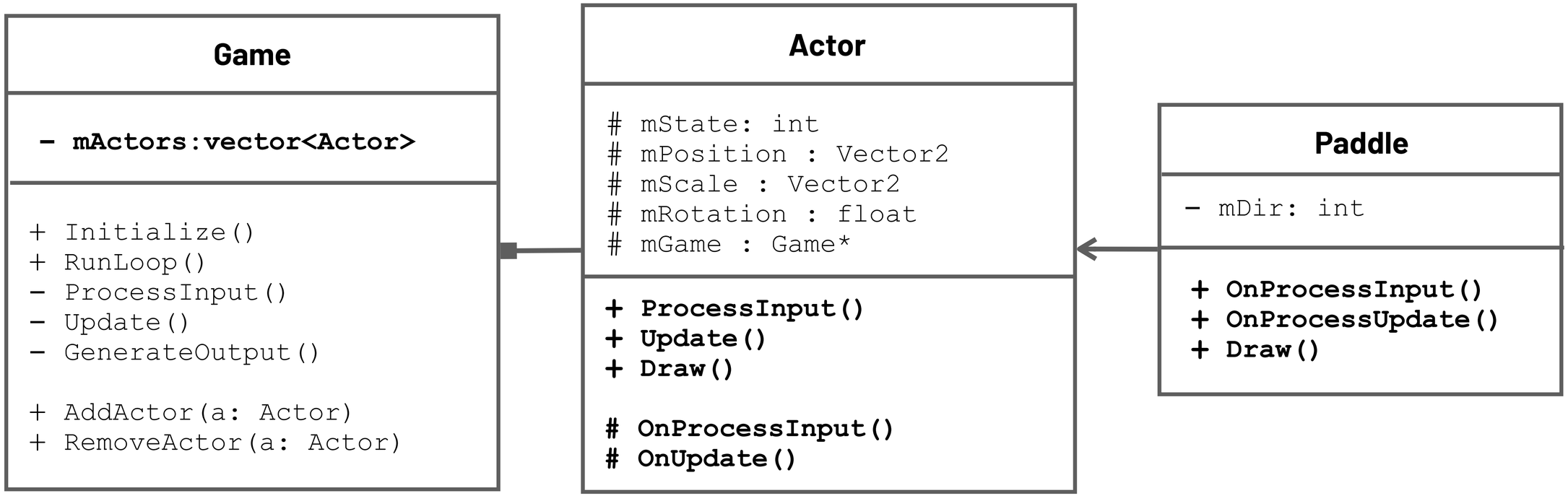


```
while player.lives > 0
  // Atualizar posição
  update player.position based on input

  // Atualizar posição
  foreach Ghost g in world
    if player collides with g
      kill either player or g
    else
      update AI for g

  // Comer as pastilhas/power ups
  ...
```

- Verificação/Resolução de colisão não é trivial
- Cada fantasma tem sua IA, também não trivial

Game Update

Em engines modernas, o jogo geralmente é um objeto que contém uma lista de *Game Objects (ou Actors)* onde cada objeto implementa seu próprio **`ProcessInput()`**, **`Update()`** e **`Draw()`**



- `+ProcessInput()` e `+Update()` são chamados pelo `Game`
- `#OnProcessInput()` e `#OnUpdate()` são estendidos pela classe filha, dependendo do estado (ativo ou pausado) `mState` do `Actor`

Game Update

Em engines modernas, o jogo geralmente é um objeto que contém uma lista de *Game Objects (ou Actors)* onde cada objeto implementa seu próprio **`ProcessInput()`**, **`Update()`** e **`Draw()`**

```
class Game {
public:
  void AddActor(Actor *obj);
  void RemoveActor(Actor *obj);

private:
  void ProcessInput();
  void Update(float deltaTime);
  void GenerateOutput();

  vector<Actor *> mActors;
};
```

```
void Game::ProcessInput() {
   for (auto actor: mActors)
      actor->ProcessInput(deltaTime);
}
```

```
void Game::Update(float deltaTime) {
   for (auto actor: mActors)
      actor->Update(deltaTime);
}
```

```
void Game::Draw() {
   for (auto actor: mActors)
      actor->Draw(deltaTime);
}
```

Game Update

Adicionar/Remover game objects durante o update de um objeto pode ser um problema:

- **Adicionar** um elemento ao final da lista faz com que ele seja atualizado antes mesmo dele ser visualizado
- **Remover** um elemento da lista que está antes de i pula o update de um objeto

```
void Game::Update(float deltaTime) {
    for (auto actor: mActors)
        actor->Update(deltaTime);
}
```

Game Update

Adicionar/Remover game objects durante o update de um objeto pode ser um problema:

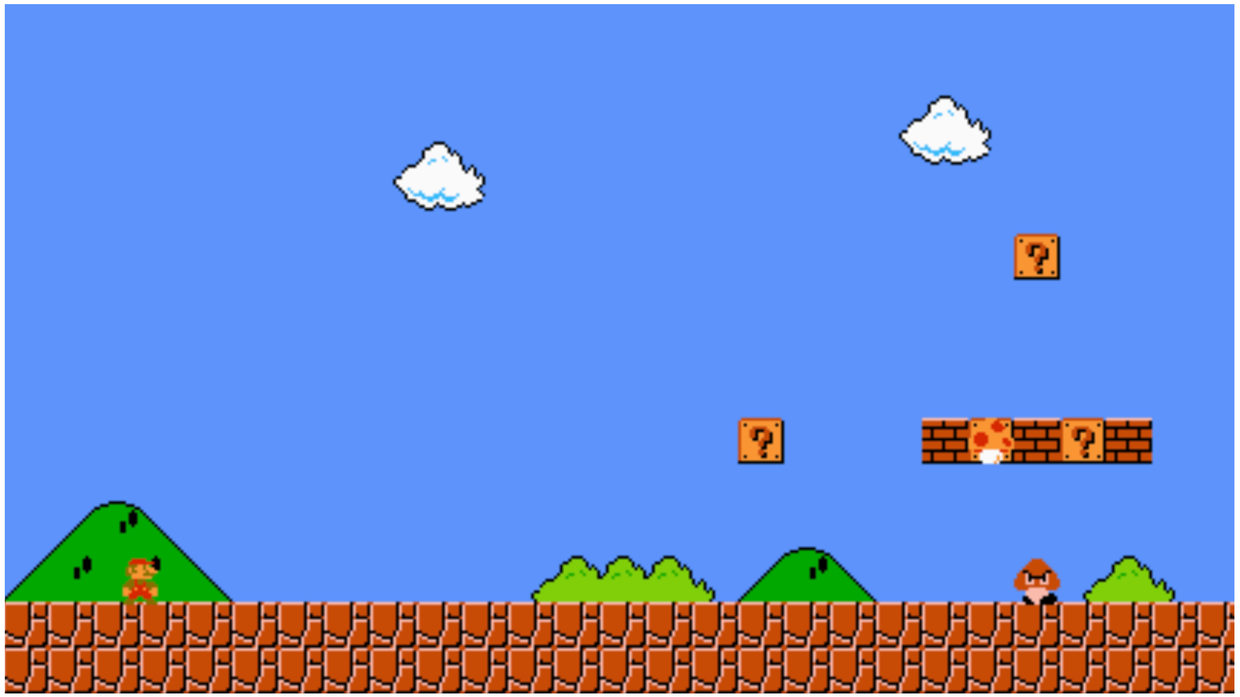
- **Adicionar** um elemento ao final da lista faz com que ele seja atualizado antes mesmo dele ser visualizado
- **Remover** um elemento da lista que está antes de i pula o update de um objeto
- **Solução**: adicionar uma lista auxiliar de "*atores pendentes*" para armazenar objetos adicionados durante os updates dos objetos e adicioná-los à lista de actors no final do update

```
void Game::Update(float deltaTime) {
    mUpdatingActors = true;
    for (auto actor: mActors)
        actor->Update(deltaTime);
    mUpdatingActors = false;

    for (auto pendActor: mPendingActors)
        mActors->Add(pendActor);
    mPendingActors.clear();
}
```

```
void Game::AddActor(Actor *actor) {
    if (mUpdatingActors)
        mPendingActors->Add(actor);
    else
        mActors->Add(actor);
}
```

O que são game objects?

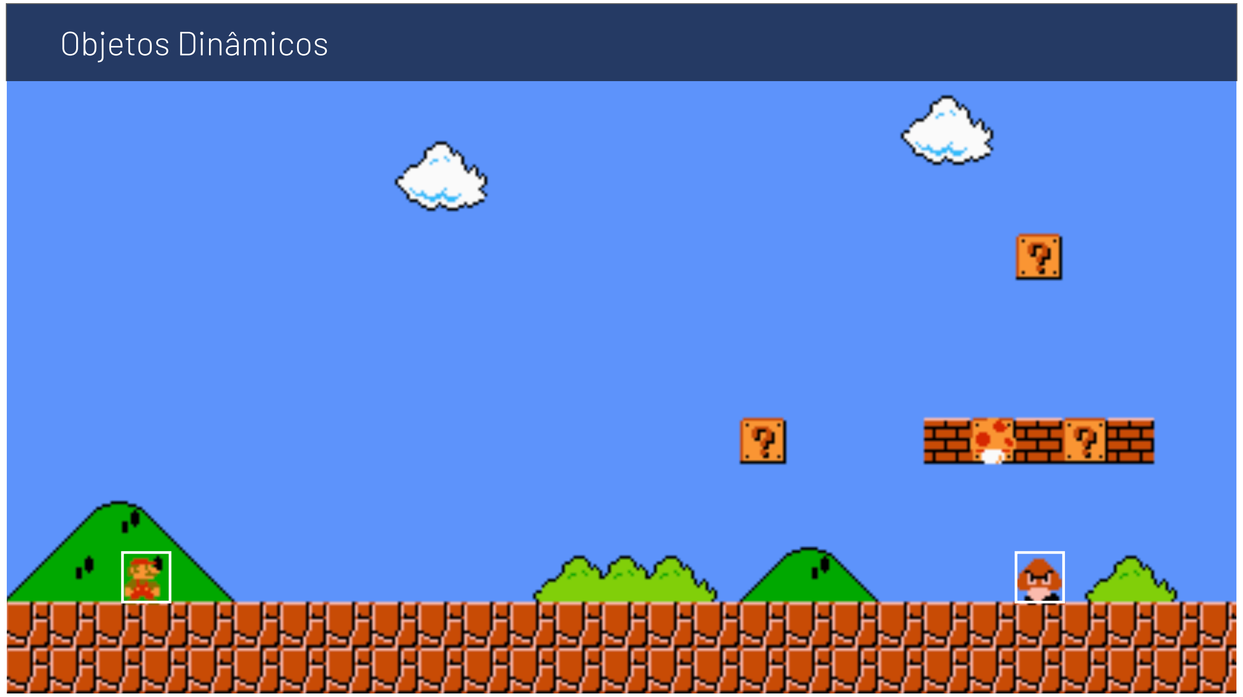
Até agora temos como adicionar objetos no jogo, mas o que é um *game object* em si? quais atributos e métodos eles devem ter?



```
class Actor {
public:
  Actor();
  ????

private:
  ????
};
```

Game Objects Dinâmicos

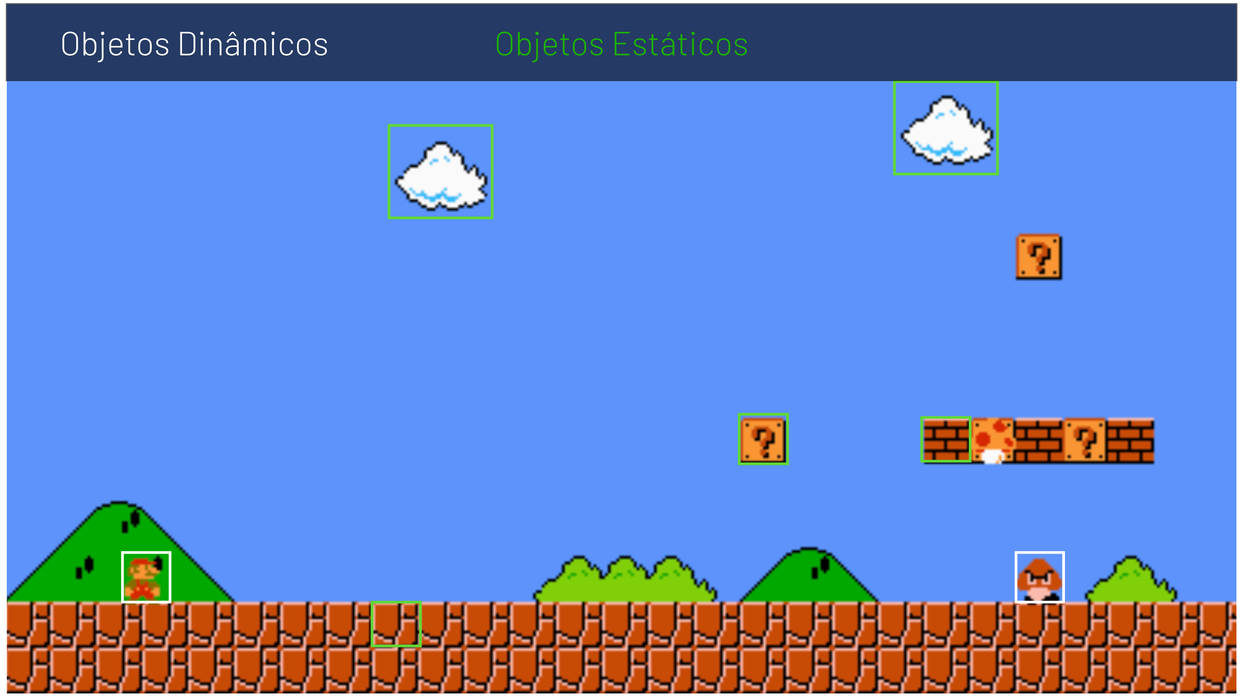
Objetos dinâmicos possuem gráficos e se movem no mundo, portando devem possuir atributos físicos (velocidade, aceleração, colisor...) e métodos para desenho:




```
class Actor {
public:
  void ProcessInput();
  void Update();
  void Draw();
  void Move();

private:
  Vector2 mVelocity;
  Vector2 mAcceleration;
  Rect mCollider;
};
```

Game Objects Estáticos

Objetos estáticos também possuem gráficos, mas não se movem, apesar de que alguns possuem colisores (chão, plataformas) e outros não (nuvem):




```
class Actor {
public:
  void ProcessInput(); ???
  void Update(); ???
  void Draw();
  void Move(); ???

private:
  Vector2 mVelocity; ???
  Vector2 mAcceleration; ???
  Rect mCollider; ???
};
```

Game Objects Gatilhos (*Triggers*)

Objetos gatilhos utilizam colisão para chamar um determinado evento no jogo (ex., spawn do Goomba), mas não possuem gráficos:


```
class Actor {
public:
  void ProcessInput(); ???
  void Update(); ???
  void Draw(); ???
  void Move(); ???

private:
  Vector2 mVelocity; ???
  Vector2 mAcceleration; ???
  Rect mCollider; ???
};
```

Problema: os objetos de jogo geralmente são muito diferentes entre si!

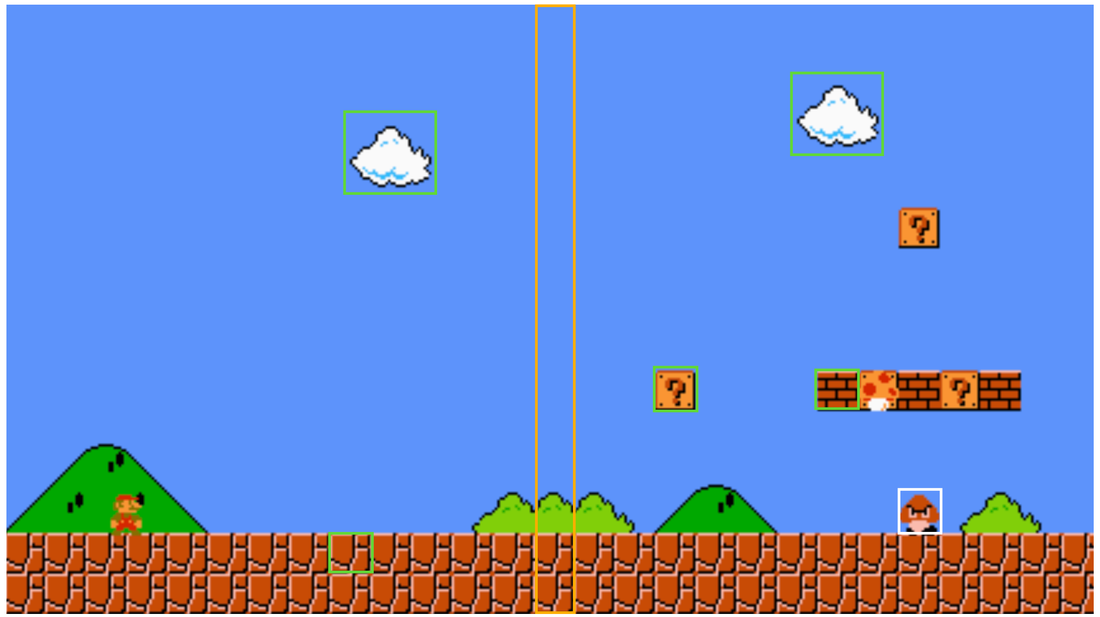
Modelagem de Objetos

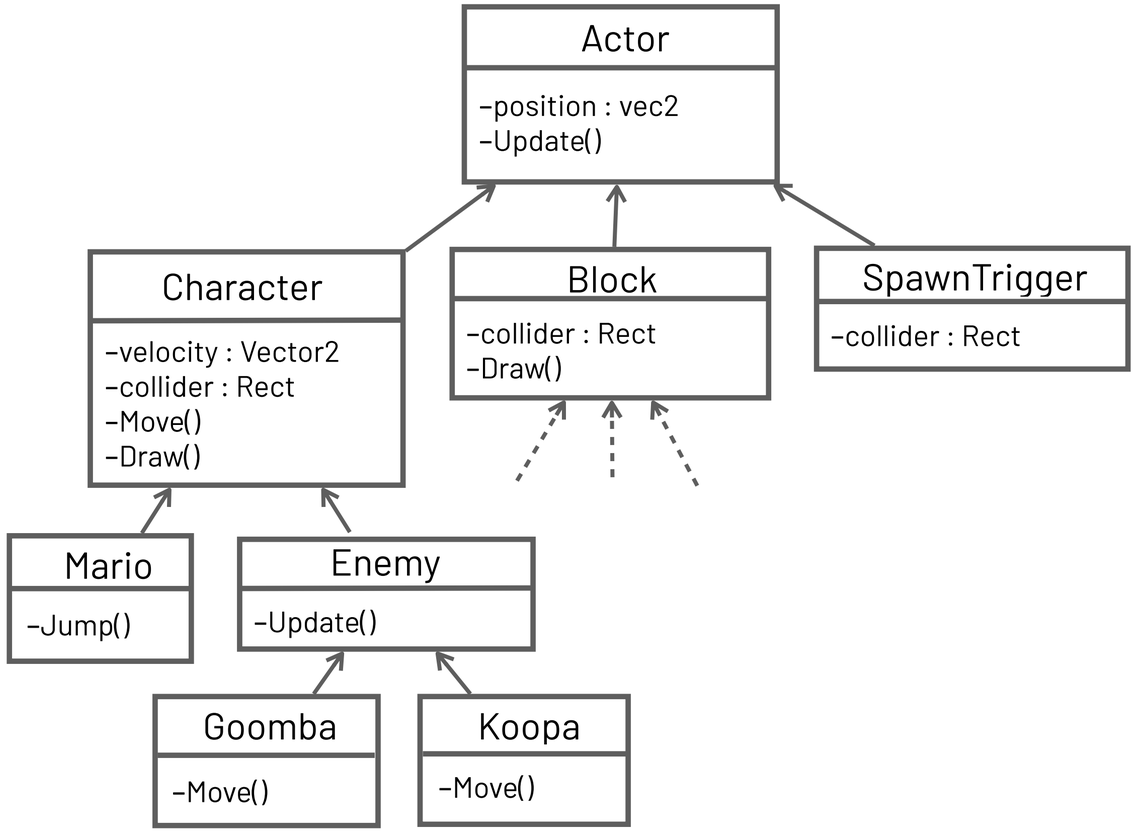
Existem três técnicas principais para modelor game objects:

- Modelo de hierarquia de classes
- Modelo de componentes
- Modelo híbrido

Modelo de Hierarquia de Classes

No **modelo de hierarquia de classes**, o comportamento dos objetos do jogo é definito e compartilhado utilizando uma hierarquia de classes, com a raíz em um classe base (Actor)

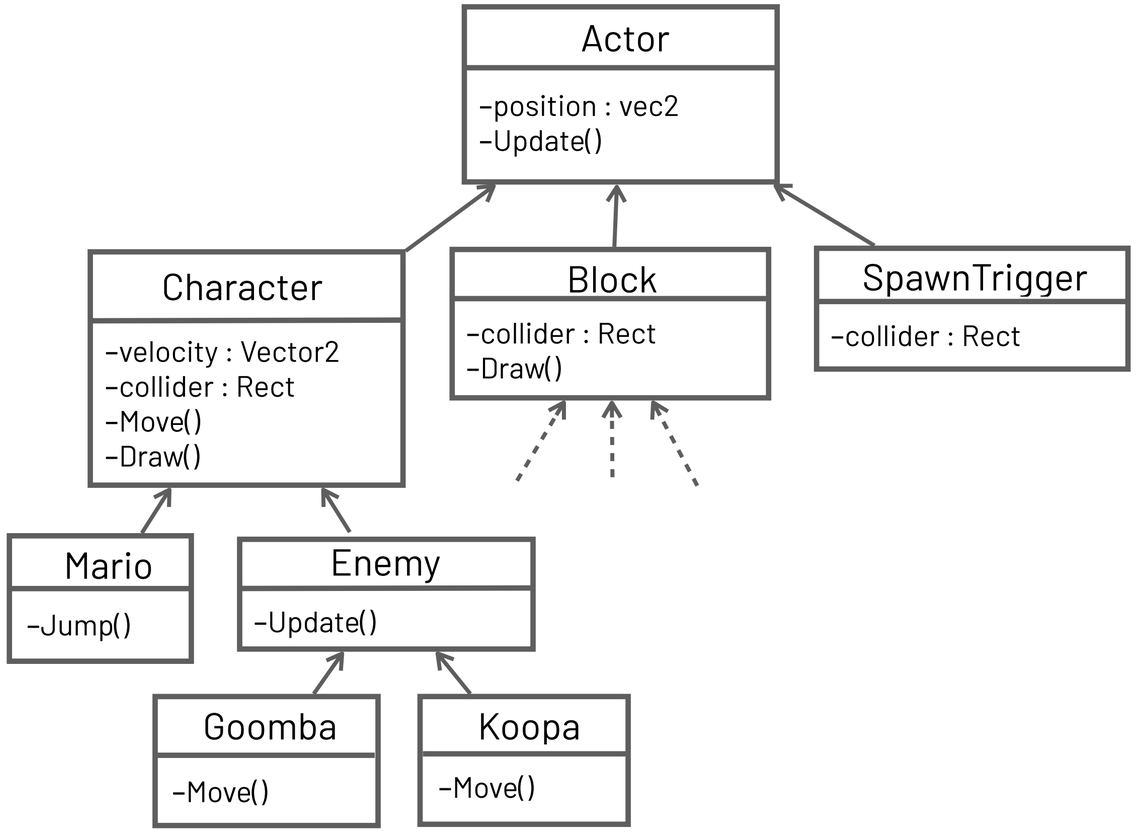




Problema do Modelo de Hierarquia de Classes

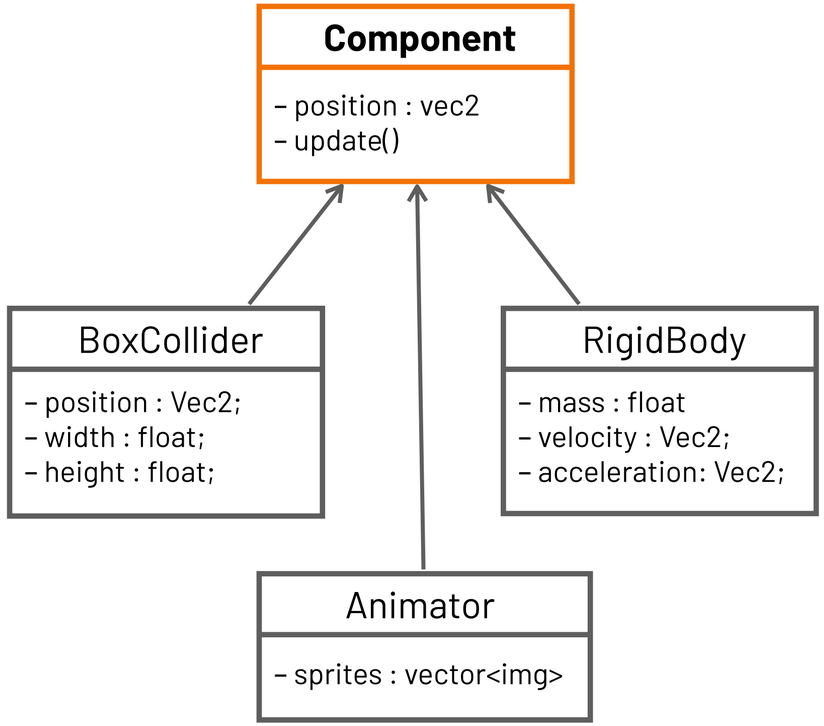
Em geral, objetos em jogos possuem muitas características independentes, o que gera uma explosão combinatória de classes com uma hierarquia profunda

- **Hierarquia de classes profundas**
  Alterações em classes base podem quebrar inesperadamente as classes derivadas.
- **Reutilização limitada**
  A herança só permite reutilizar código de superclasses, não de classes “irmãs".
- **Dificuldade em componentes dinâmicos**
  Objetos precisam mudar comportamentos durante a execução, o que é complicado com herança pura.



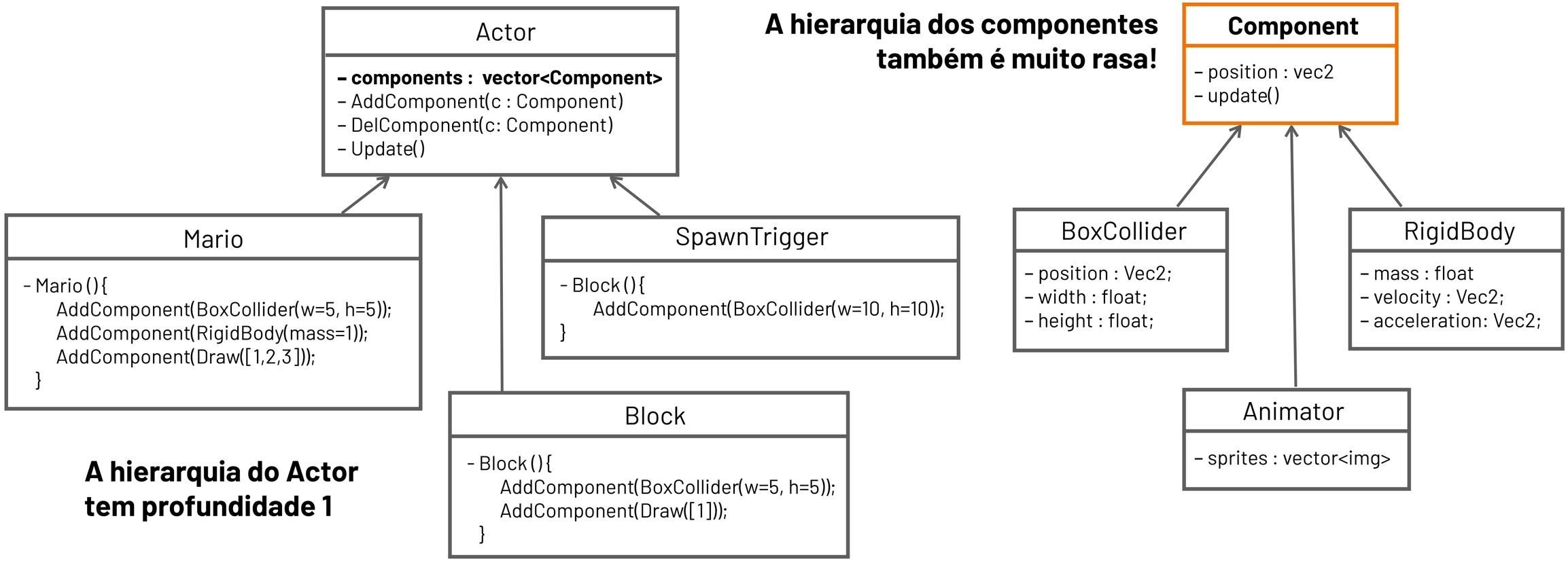
Modelo de Componentes

No **modelo de componentes**, cada objeto do jogo tem uma lista de componentes que, quando combinados, definem a sua funcionalidade (Ex., Unity).



Modelo de Componentes

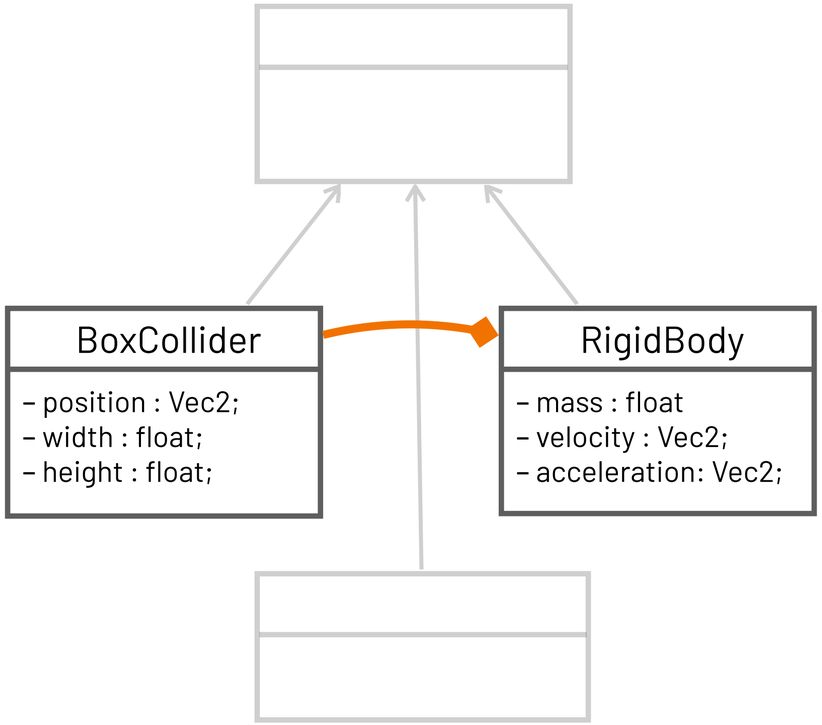
No **modelo de componentes**, cada objeto do jogo tem uma lista de componentes que, quando combinados, definem a sua funcionalidade (Ex., Unity).



Problemas do Modelo de Componentes

Em geral, diferentes componentes precisam comunicar entre si, fazendo com que hajam várias buscas por componentes, prejudicando performance

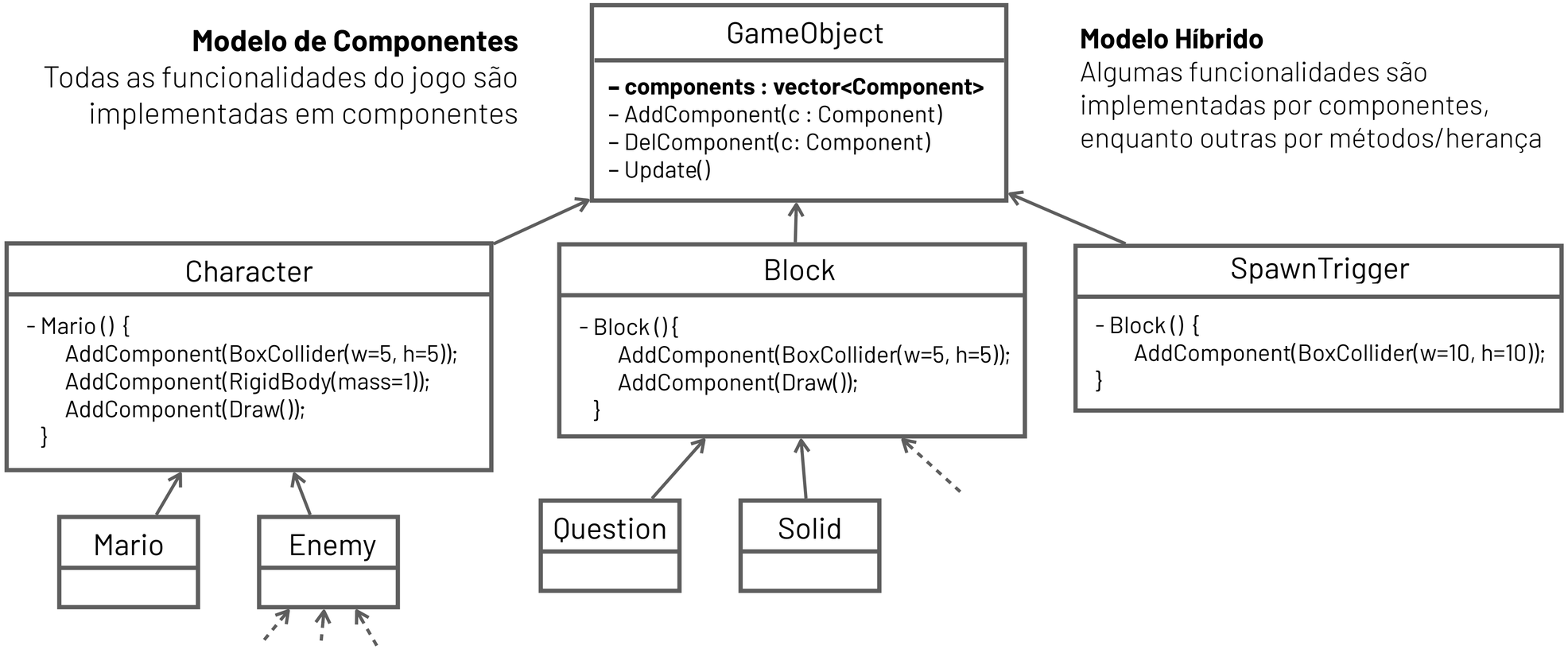
```
BoxCollider::CheckHorizontalCollision() {}
BoxCollider::CheckVerticalCollision() {}

BoxCollider::Update(float deltaTime) {
   // Get object velocity from RigidBody
   RigidBody *body = owner->GetComponent<RigidBody>();

   if(Math::Abs(body->GetVelocity().x) > 0)
      CheckHorizontalCollision();
   }

   if(Math::Abs(body->GetVelocity().y) > 0)
      CheckVerticalCollision();
   }
}
```



Modelo de Objetos Híbrido

No **modelo híbribo**, combinados uma hierarquia de classes com componentes, ou seja, algumas propriedades/funções são passadas por herança enquanto outras por componentes (Ex. Unreal)

Modelo de Componentes
Todas as funcionalidades do jogo são implementadas em componentes



Modelo Híbrido
Algumas funcionalidades são implementadas por componentes, enquanto outras por métodos/herança

Próxima aula

A7: Física I - Objetos Rígidos

- Geometrias de colisão
  - Falsos Positivos
- Detecção de colisão
  - Circunferência vs. Circunferência
  - AABB vs. AABB
- Resolução de Colisão
  - AABB vs. AABB