# Detecção de Colisão

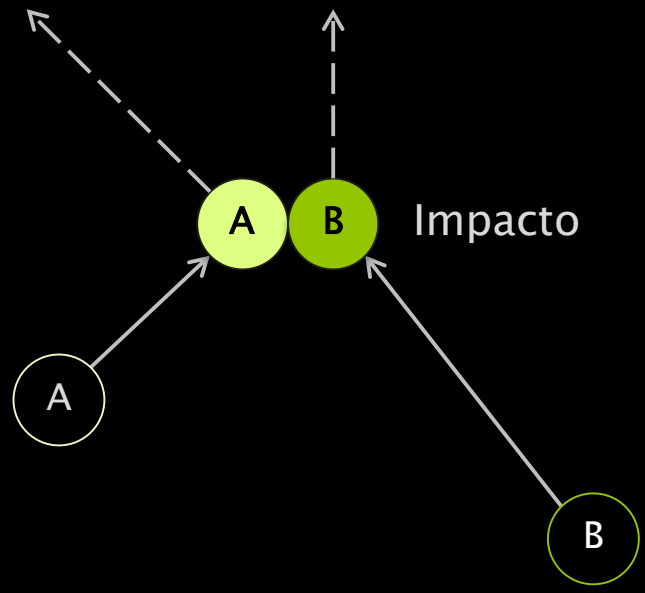
## Programação de Jogos

**Judson Santos Santiago**

# Introdução

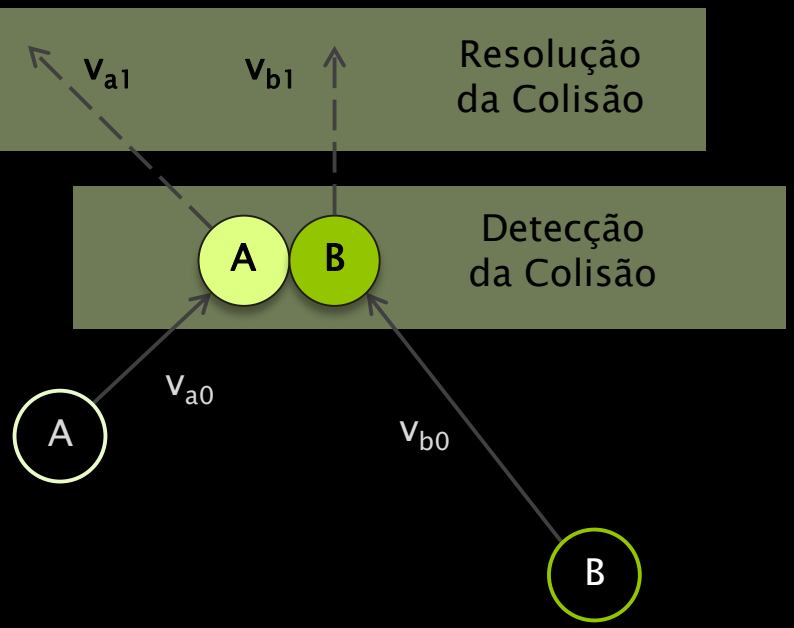
- No mundo real, os objetos estão condicionados às:
  - Leis da física
    - Inércia
    - Atrito
    - Gravidade
    - Etc.
  - Propriedades da matéria:

    "*Dois corpos não ocupam o mesmo lugar no espaço no mesmo instante de tempo*"



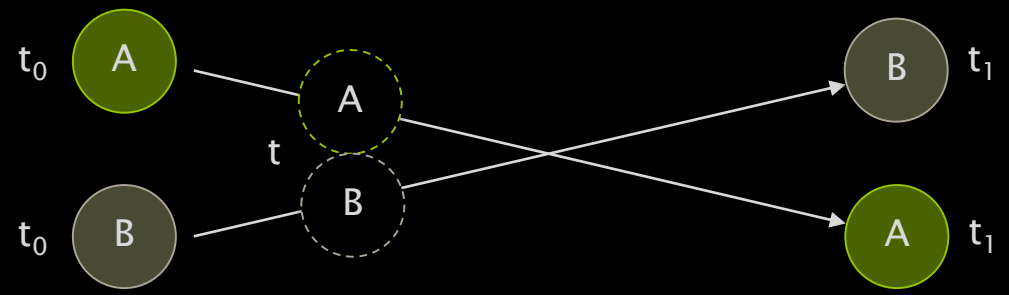
# Introdução

- No mundo virtual, nós definimos as regras
  - A detecção de colisão vai determinar se e quando dois objetos colidem
  - A resolução de colisão vai determinar o que acontece com os objetos após a colisão



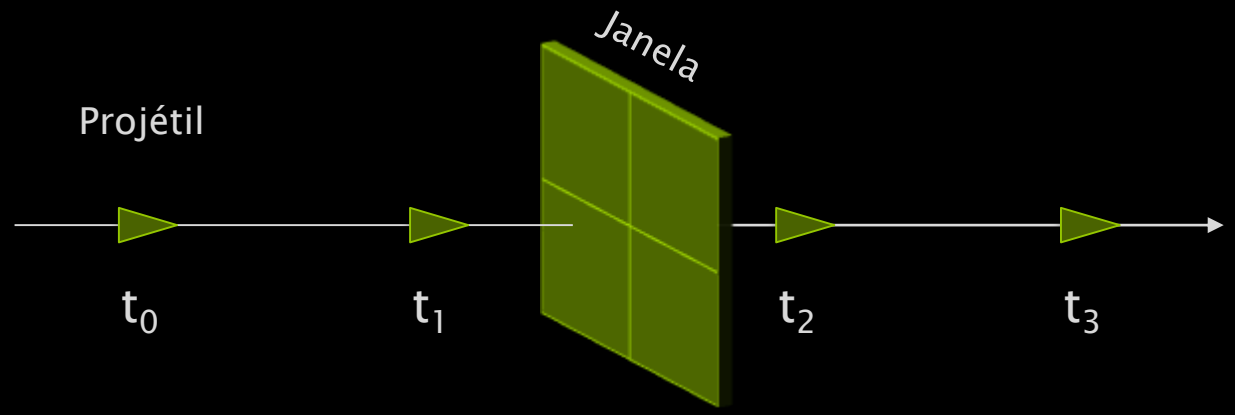
# Detecção de Colisão

- Para determinar a colisão existem duas técnicas:
  - Teste de sobreposição: detecta se uma colisão ocorreu
  - Teste de interseção: prediz, através de uma fórmula matemática, se uma colisão acontecerá no futuro

# Detecção de Colisão

- Determinar colisão não é uma tarefa fácil:
  - Movimentação muito rápida
    Ex.: balas em um jogo de tiro



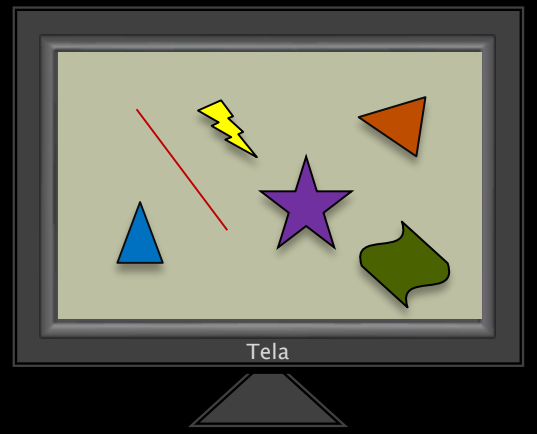

O teste se sobreposição requer um controle da velocidade dos objetos e da taxa de atualização do jogo.

# Detecção de Colisão

- Determinar colisão não é uma tarefa fácil:
  - O teste de colisão entre geometrias complexas tem alto custo computacional
    Ex.: personagens segurando armas
  - Custo elevado: cada objeto deve ser testado contra todos os demais objetos da cena, ou seja, é um procedimento com custo $O(n^2)$

Formas Complexas

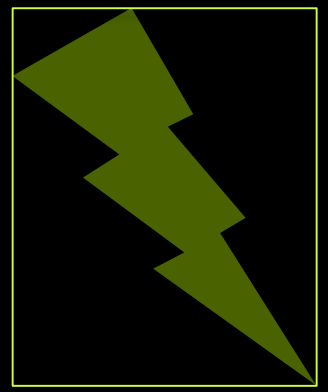
Custo Elevado



# Geometria Simplificada

- Uma solução para lidar com geometrias complexas é simplificar a geometria usando uma bounding box

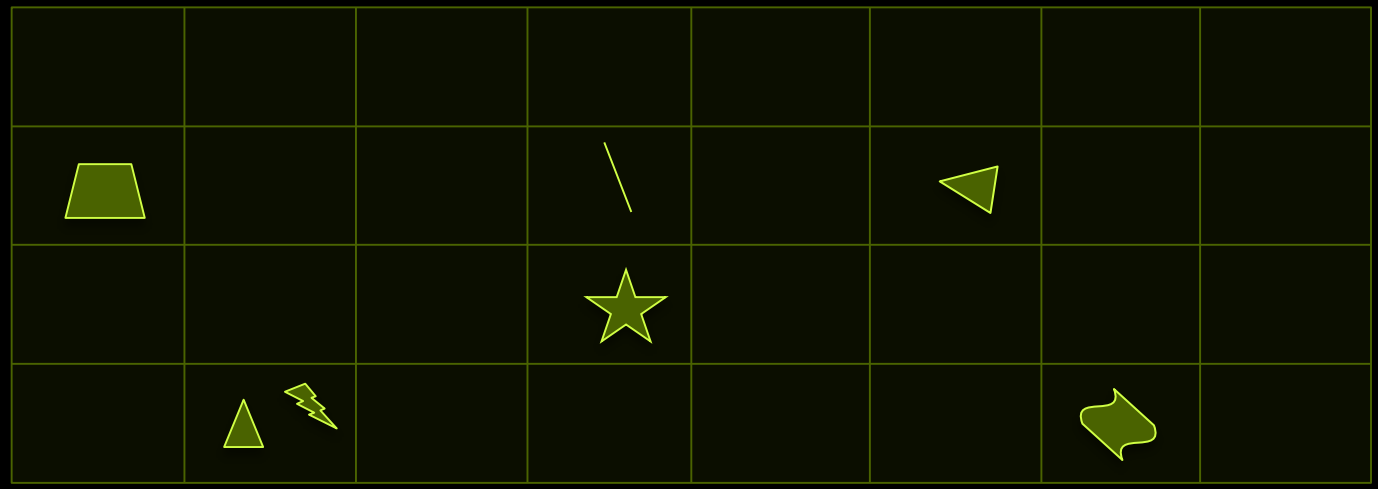
Bounding Box
Circular



Bounding Box
Retângular

# Particionamento do Espaço

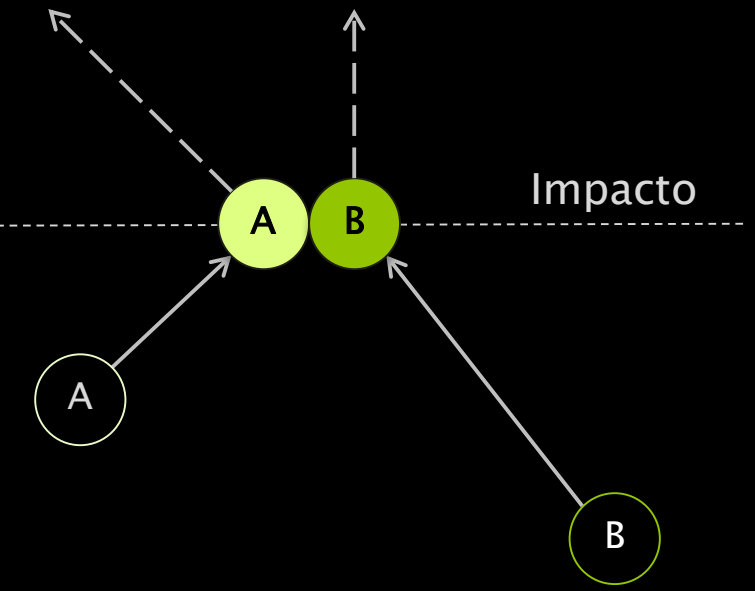
- Uma solução para a complexidade $O(n^2)$ é particionar o mundo do jogo em regiões menores
  - Objetos só são testados contra outros na mesma partição
  - Na média esse método obtém complexidade linear



Mundo do Jogo

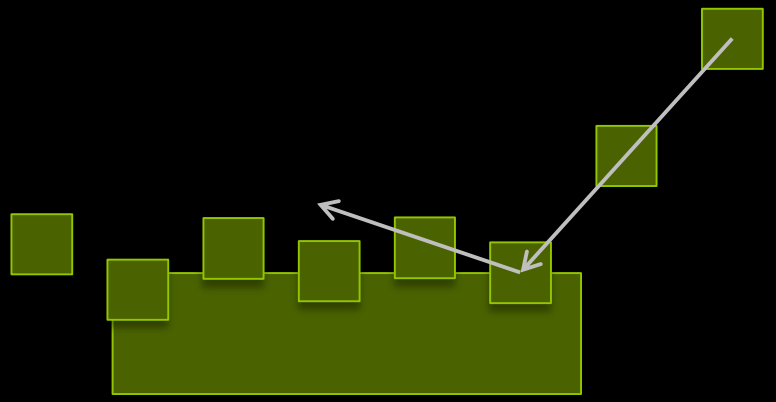
# Resolução da Colisão

- Uma vez que a colisão foi detectada, alguma ação deve ser tomada para resolver a colisão
  - Posicionar os objetos na posição de contato
  - Calcular novas velocidades
  - Gerar som do impacto
  - Iniciar animação
  - Destruir objetos

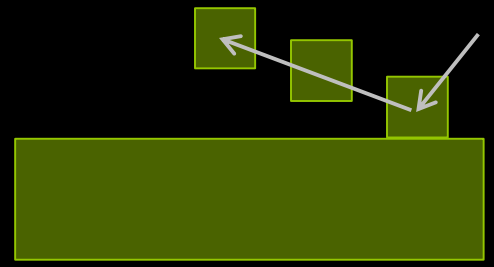


# Resolução da Colisão

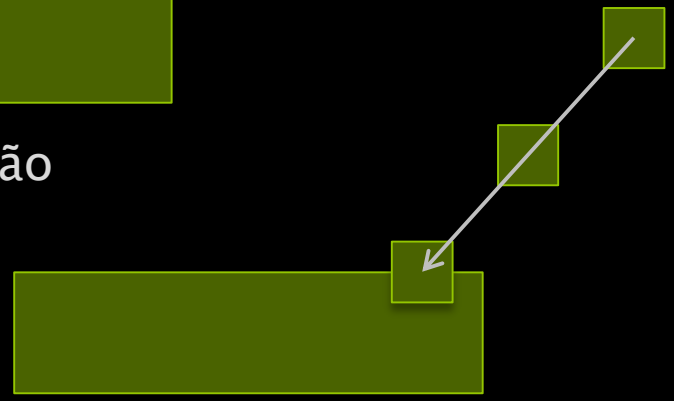
- Se o método de sobreposição for utilizado é necessário reposicionar os objetos
  - Os objetos podem entrar em um loop de colisão, dependendo:
    - Da frequência de atualização
    - Velocidade dos objetos



Loop de colisão



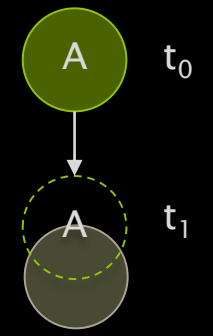
Resolução

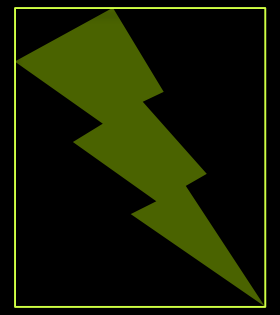


Detecção

# Sistema de Colisão

- Na disciplina utilizaremos um sistema de colisão:
  - Baseado no teste de sobreposição
  - Usando uma bounding box para cada objeto
  - Particionando o espaço em dois grupos:
    - Objetos em movimento: podem colidir com objetos estáticos ou outros objetos em movimento
    - Objetos estáticos: objetos que não colidem entre si, podendo colidir apenas com objetos em movimento
  - Complexidade $O(m^2)$



Bounding Box

# Sistema de Colisão

- O teste de sobreposição é o método de colisão mais fácil de implementar e o mais utilizado em jogos 2D
  - Requer a implementação de testes de sobreposição entre as geometrias suportadas:
    - Ponto
    - Retângulo
    - Círculo
    - Mista

Retângulo

Mista

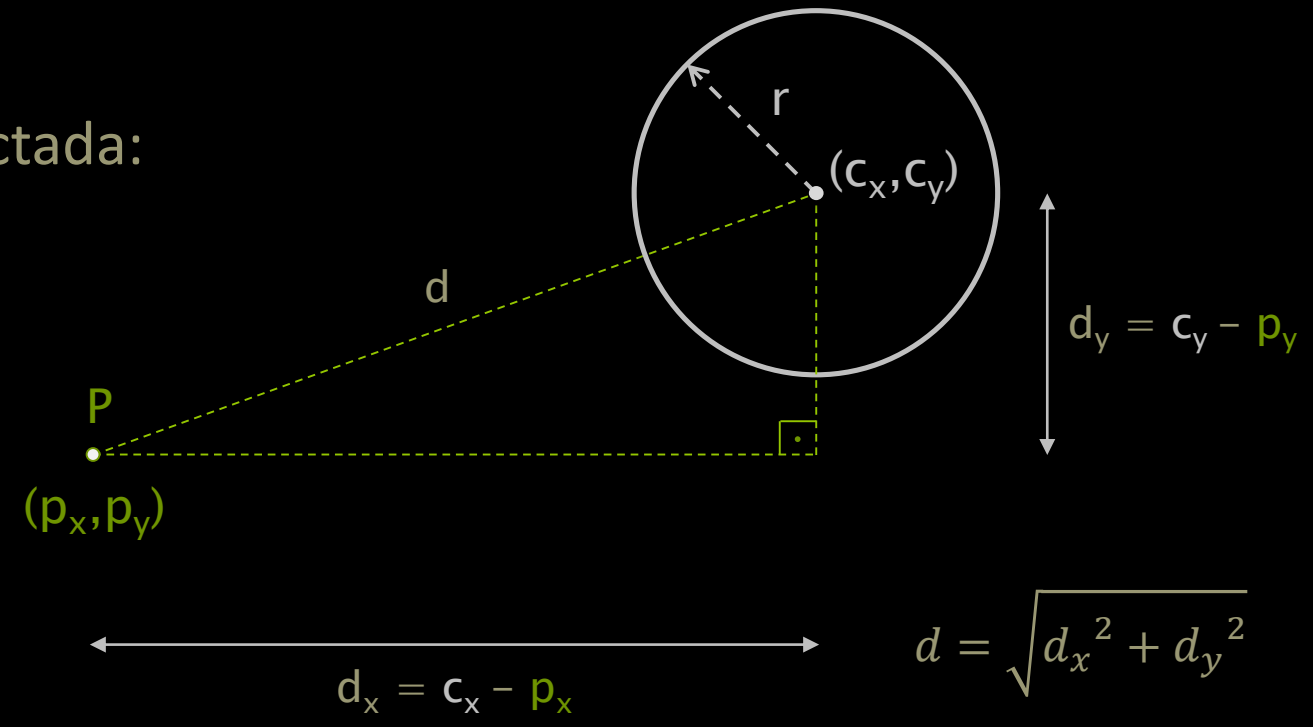
Círculo

Ponto

# Colisão Ponto-Círculo

- Teste de colisão entre um ponto e um círculo pode ser feito calculando a distância entre o ponto e o centro do círculo

Colisão Detectada:
$d \leq r$

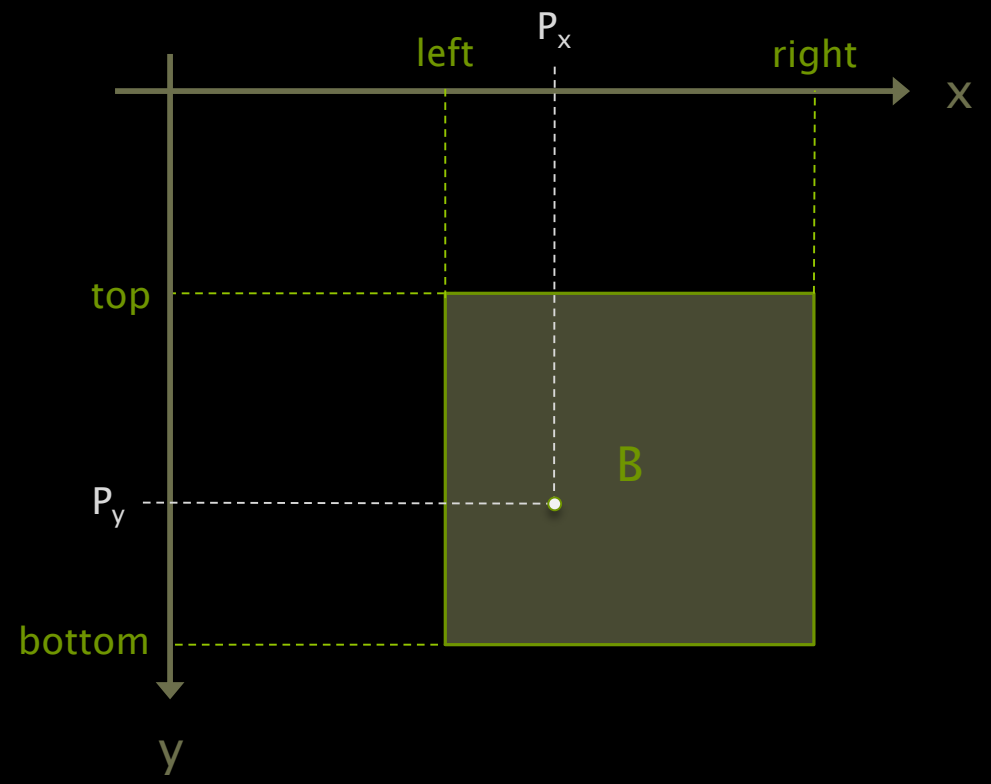


# Colisão Ponto-Retângulo

- Teste de colisão entre um ponto e um retângulo consiste em verificar se o ponto está dentro do retângulo

Colisão Detectada:

(left ≤ $P_x$ ≤ right) &&
(top ≤ $P_y$ ≤ bottom)

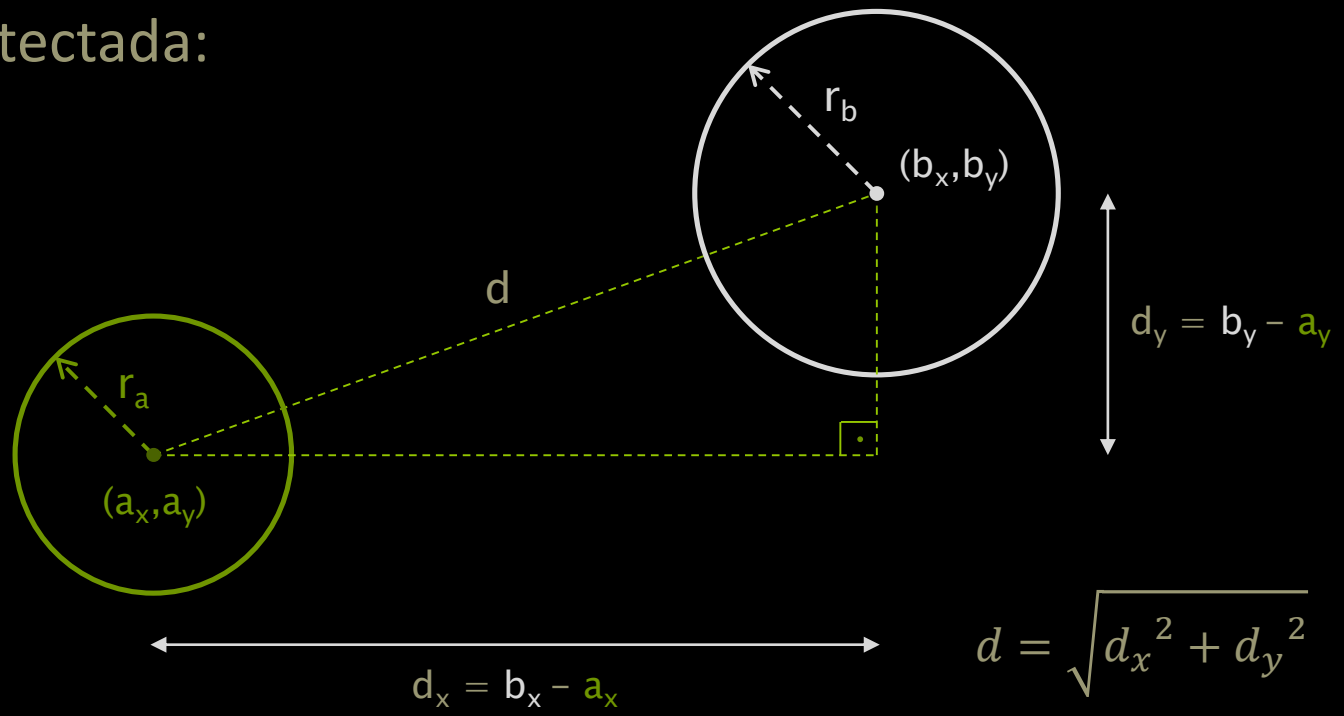


# Colisão Círculo-Círculo

- Teste de colisão entre dois círculos pode ser feito calculando a distância entre os centros dos círculos

Colisão Detectada:

$d \leq r_a + r_b$



# Colisão Retângulo-Retângulo

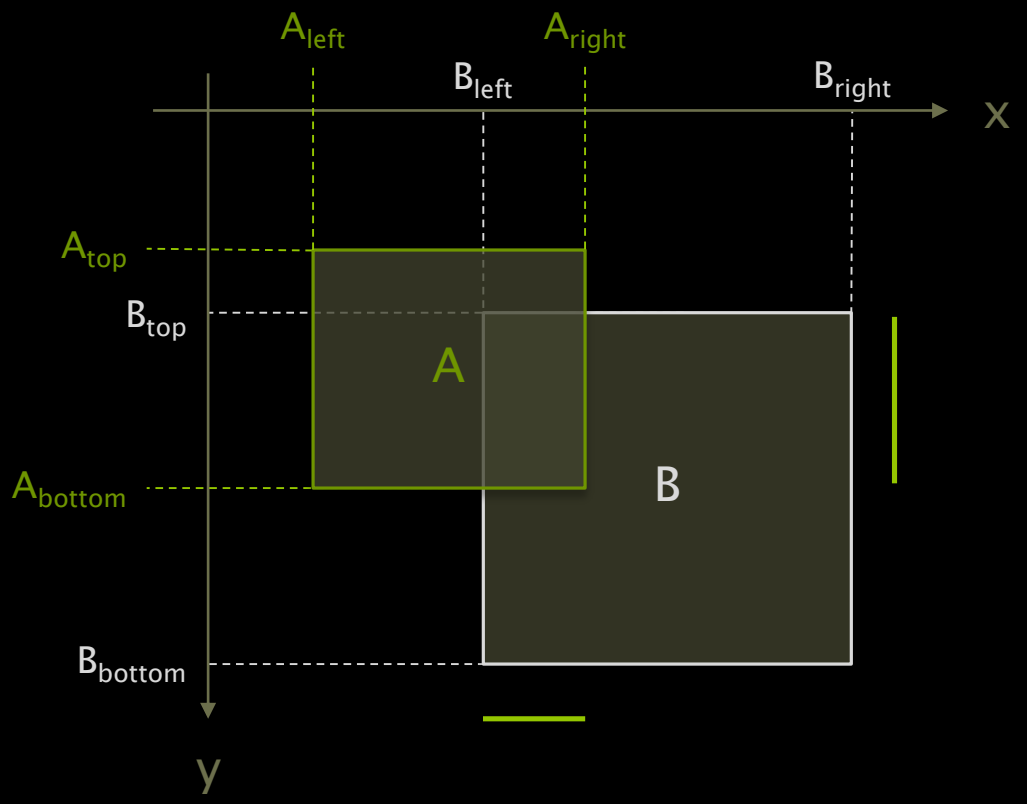
- Teste de colisão entre retângulos consiste em verificar se existe sobreposição dos retângulos em ambos os eixos

Colisão Detectada:

$$Ox = (B_{left} \leq A_{right}) \;\&\&\; (A_{left} \leq B_{right})$$

$$Oy = (B_{top} \leq A_{bottom}) \;\&\&\; (A_{top} \leq B_{bottom})$$

$$Ox \;\&\&\; Oy$$



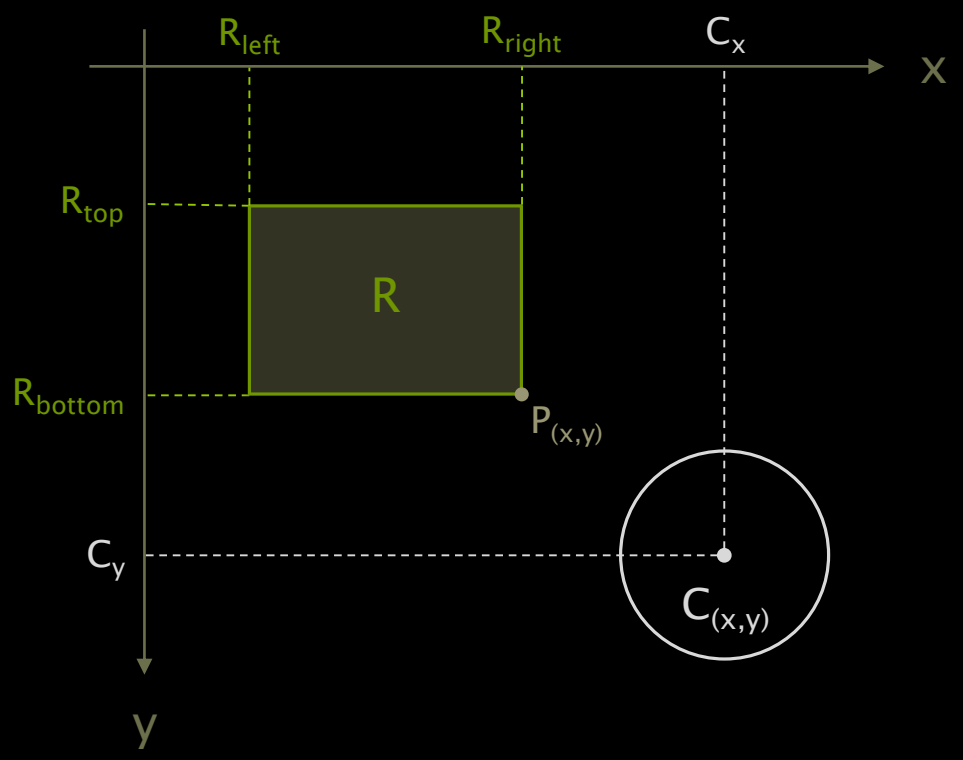
# Colisão Círculo-Retângulo

- Teste de colisão entre círculo e retângulo consiste em verificar se o ponto mais próximo colide com o círculo

Colisão Detectada:

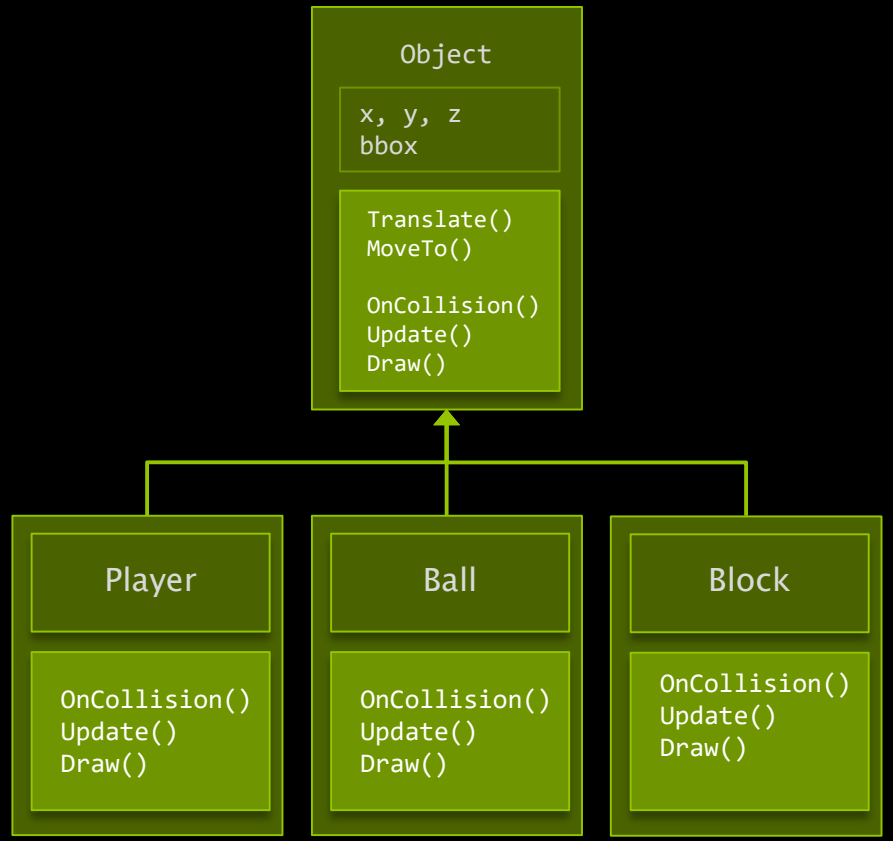
```
if (Cx < Rleft)          Px = Rleft
else if (Cx > Rright)    Px = Rright
else                     Px = Cx

if (Cy < Rtop)           Py = Rtop
else if (Cy > Rbottom)   Py = Rbottom
else                     Py = Cy
...
Collision(P, C)
```



# Implementação

- Todo objeto do jogo precisa:
  - De uma bounding box
    - É um ponteiro para uma geometria
      - Ponto
      - Círculo
      - Retângulo
      - Mista
  - De um método OnCollision
    - Código a ser executado quando for detectada a colisão do objeto com outro



# Implementação

- A classe Scene vai lidar com as colisões
  - Gerando uma lista de colisões
  - Chamando OnCollision dos objetos

Lista de Colisões

<A,B>

<A,C>

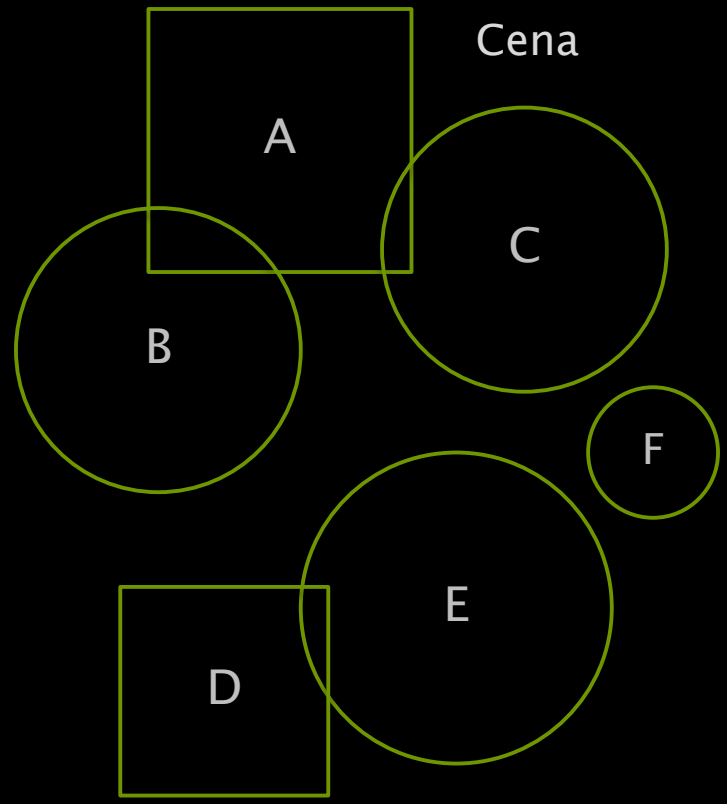
<D,E>

Uma colisão é um par de objetos

<A,B>

As colisões são tratadas pelos métodos OnCollision dos objetos:

```
A->OnCollision(B);
B->OnCollision(A);
```



# Resumo

- A detecção de colisão fornece o conceito de massa aos objetos de um jogo
  - Sem ela os objetos atravessariam uns aos outros
- O tratamento de colisão consiste em:
  - Detecção da colisão
  - Resolução da colisão
- É um procedimento de custo elevado
  - É preciso simplificar geometrias
  - Utilizar particionamento do espaço