DCC192

2025/2

UFMG

Desenvolvimento de Jogos Digitais

A8: Física II - Detecção de Colisão

Prof. Lucas N. Ferreira

Logística

Avisos

- As notas do **TP0: Configuração** foram publicadas no SIGA!
  - Se você tem qualquer questionamento sobre notas, favor entrar em contato comigo, com copiroa para o monitor

Última Aula

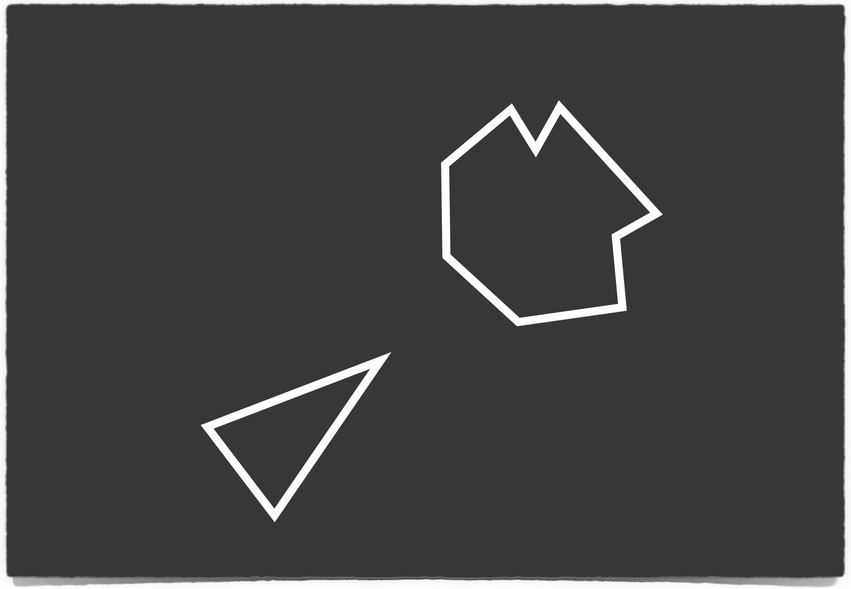
- Método de Euler Semi-implícito para movimentação de objetos rígidos
- Calcular a velocidade $\vec{v}$ e posição $\vec{s}$ a partir da aceleração $\vec{a}$
  - A aceleração é a soma das forças atuantes divido pela massa: $\vec{a} = \frac{\sum \vec{f}_i}{m}$

Plano de Aula

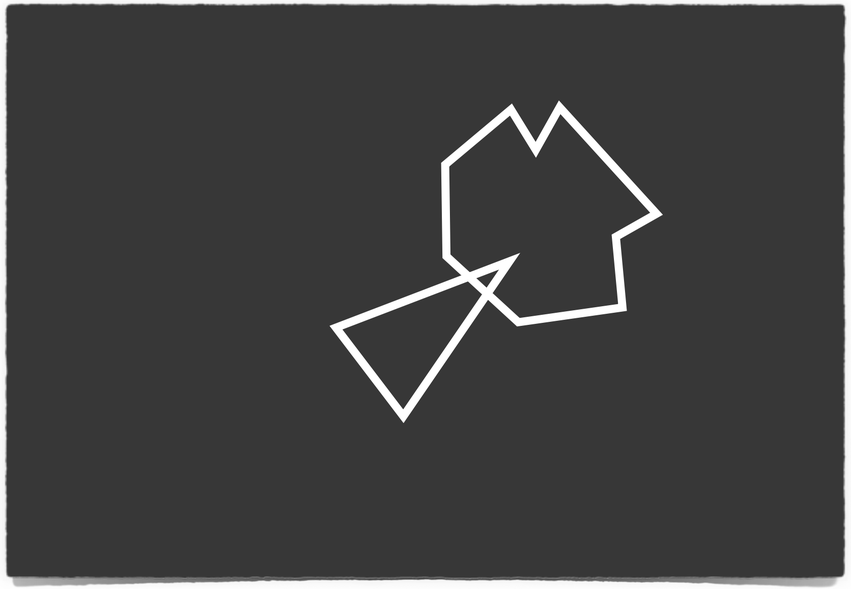
- Geometrias de colisão
  - Falsos Positivos
- Detecção de colisão
  - Circunferência vs. Circunferência
  - AABB vs. AABB
- Resolução de Colisão
  - AABB vs. AABB

Detecção de Colisão (entre dois objetos)

O problema de detecção de colisão envolve verificar de dois objetos do jogo estão colidindo entre si em um dado quadro do jogo:



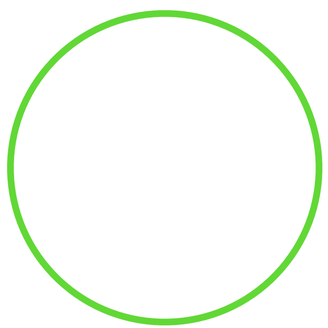
Quadro t ✗



Frame $t + 1$ ✓

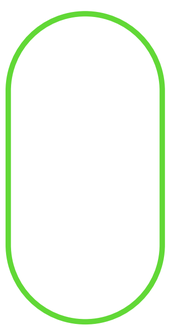
Geometrias de Colisão

A primeira etapa de um algoritmo de detecção de colisão é definir uma **geometria de colisão** para representar os corpos dos objeto do jogo. Algumas das geometrias mais usadas são:

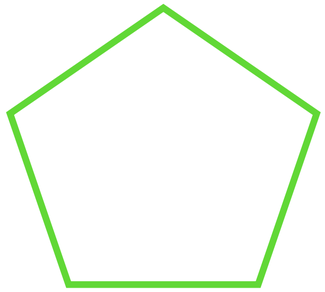


Circunferência

Caixa



Cápsula

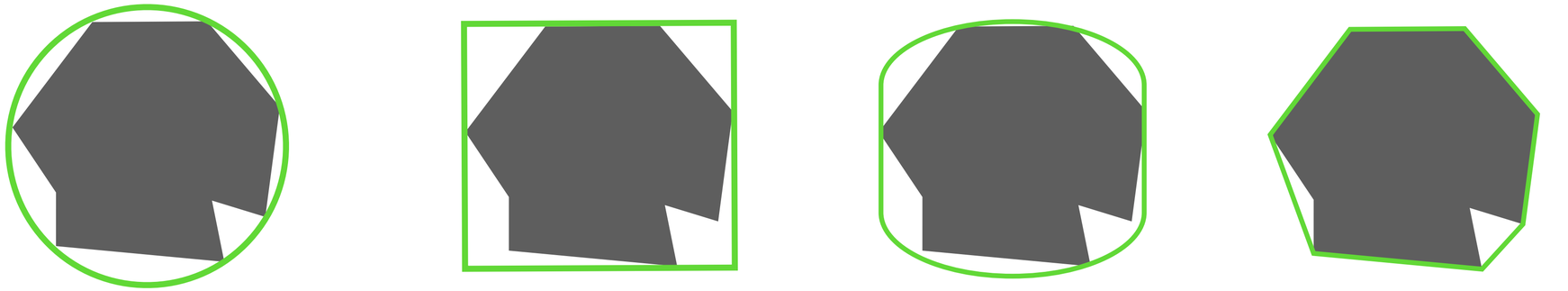


Polígonos Convexos

Em ordem de complexidade, as geometrias mais comuns para detecção de colisão são: circunferência, caixa, cápsula e polígonos convexos.

Geometrias de Colisão

Normalmente, escolhemos a geometria mais simples que melhor aproxima a representação visual do objeto.

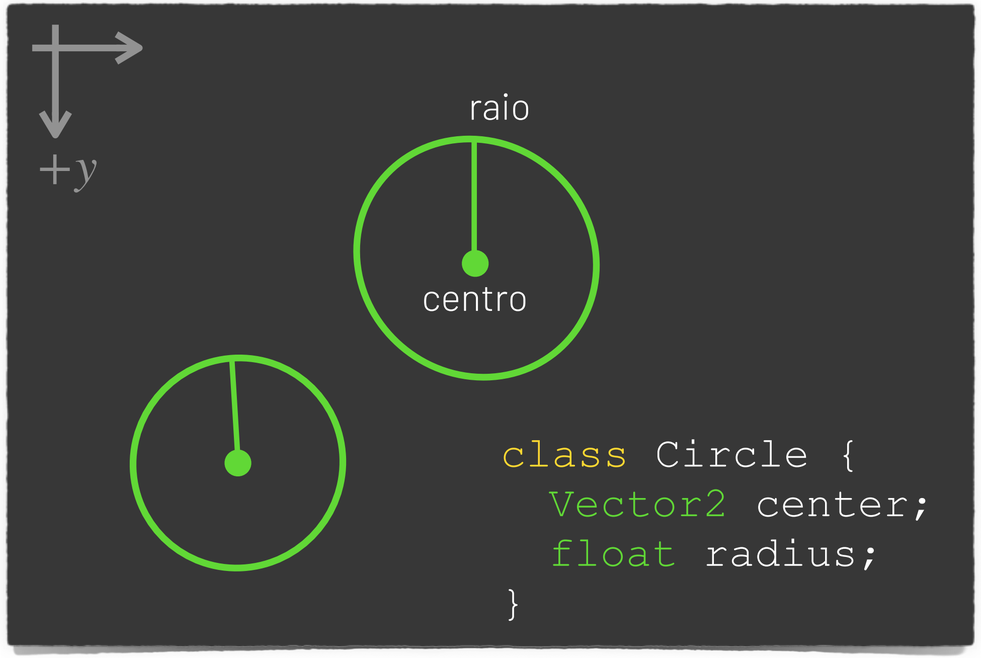


Circunferência | Caixa | Cápsula | Polígonos Convexos

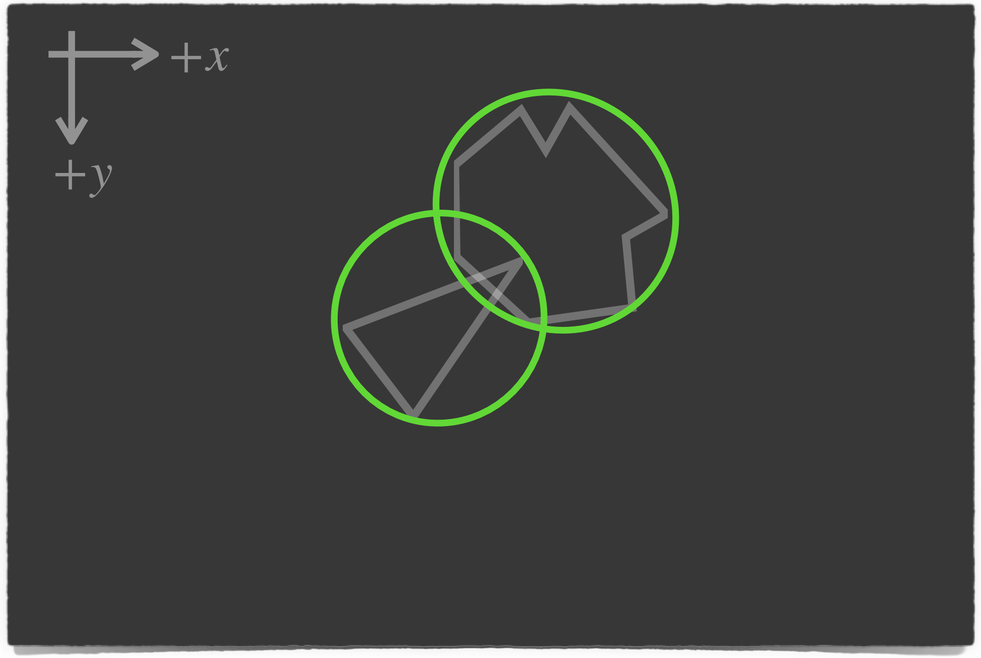
Da esquerda para a direita, exemplos de circunferência, caixa, cápsula e polígono convexo como geometria de colisão de um asteroide.

Circunferência

A **circunferência** é a geometria mais simples para detecção de colisão: ela é definida por um centro (ponto) e um raio



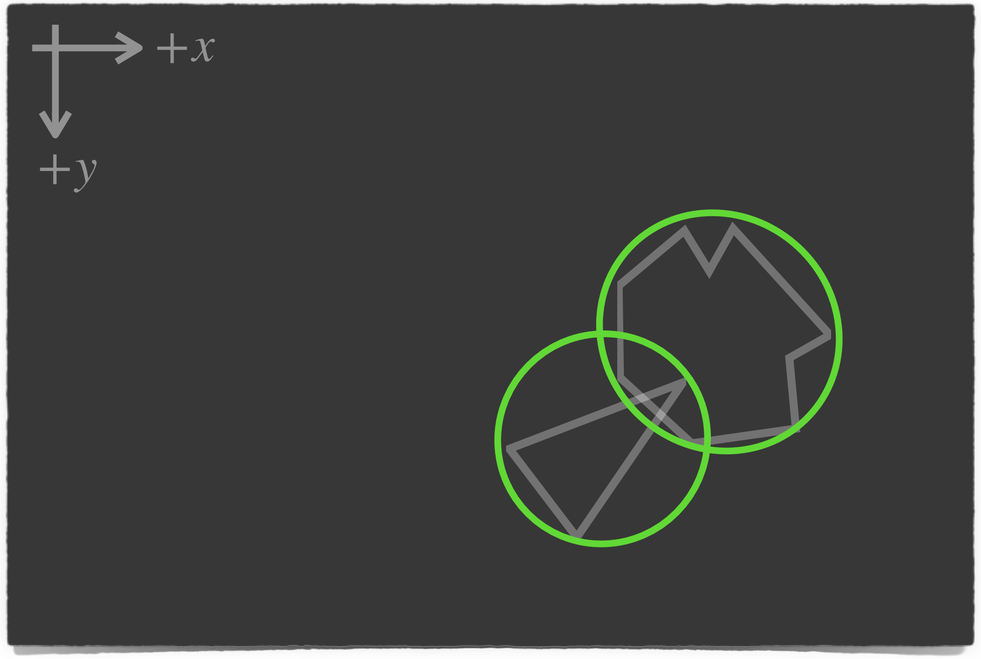
A circunferência é definida por um centro (ponto) e um raio.



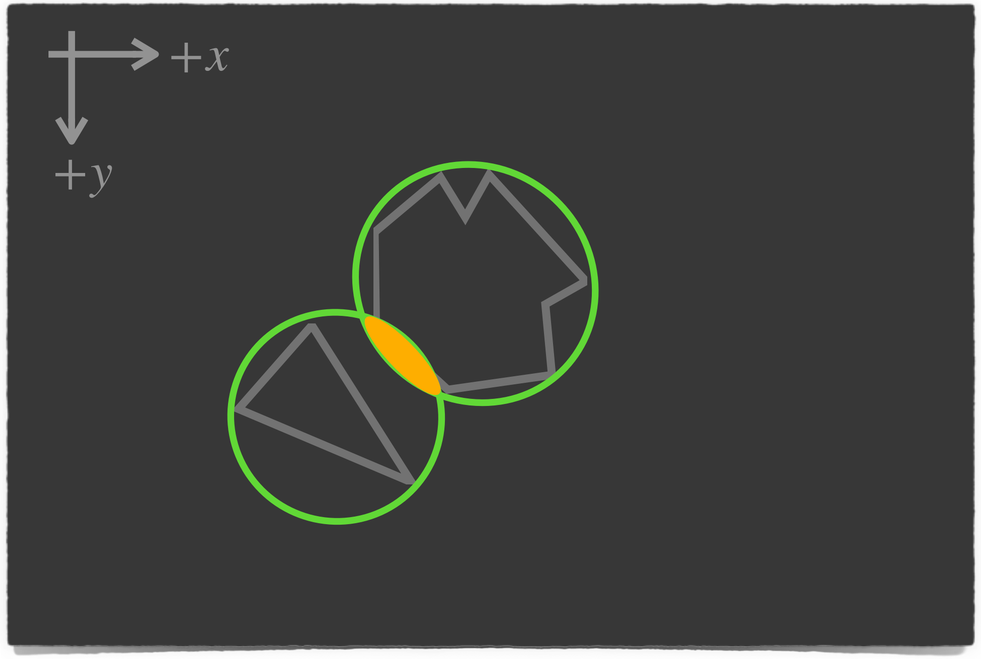
A circunferência aproxima bem o asteroide, mas **não** a nave

Falsos Positivos

Escolher geometrias de colisão inapropriadas pode gerar **falsos positivos.** Ou seja, uma colisão pode ser detectada quando os objetos não estão colidindo visualmente.



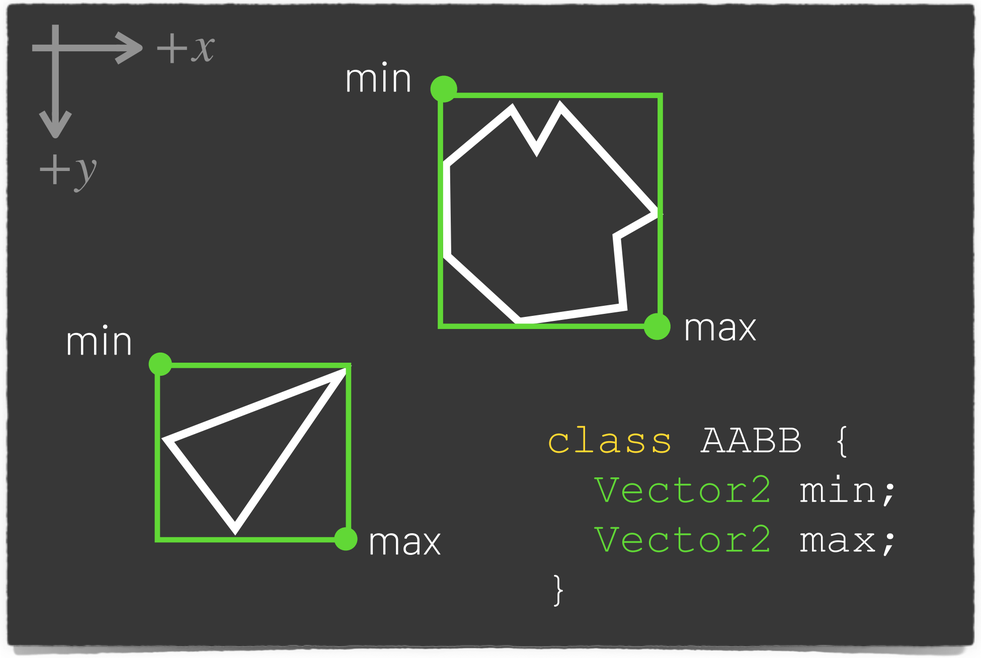
Verdadeiro Positivo



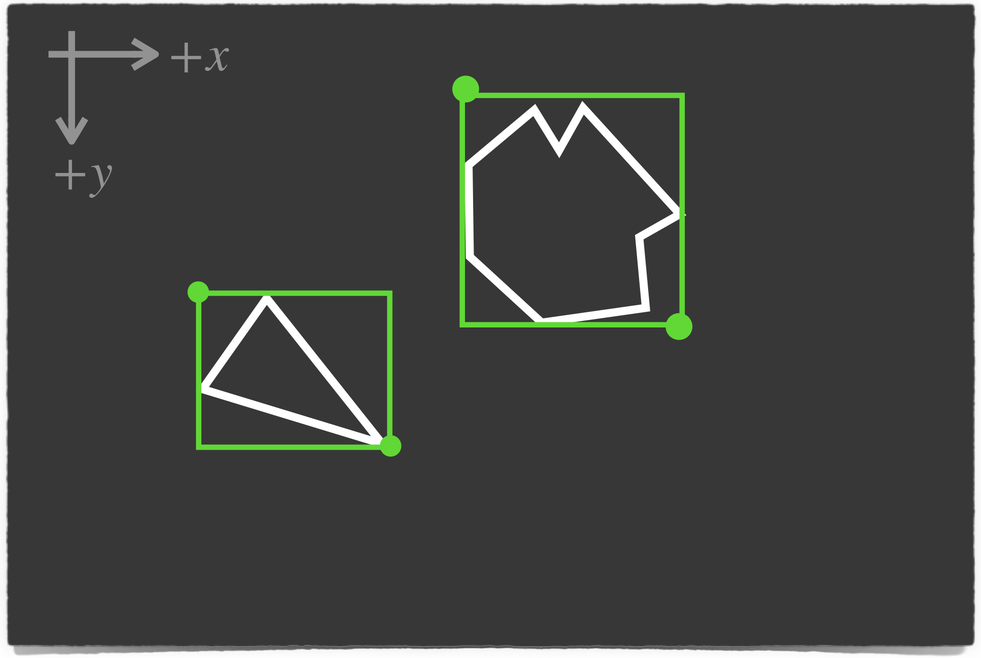
Falso Positivo

Caixa Delimitadora Alinhada com os Eixos

Uma **caixa delimitadora alinhada com os eixos, do inglês, *axis-aligned bounding box (AABB)*** é um retângulo com arestas paralelas aos eixos x e y.



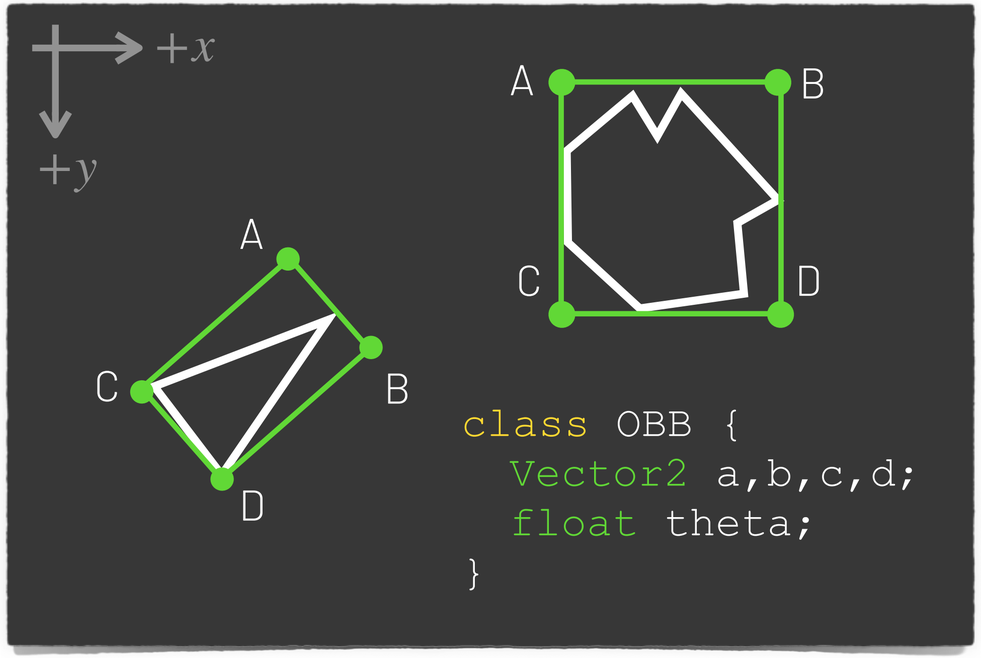
AABBs podem ser representadas por dois vértices: mínimo e máximo.



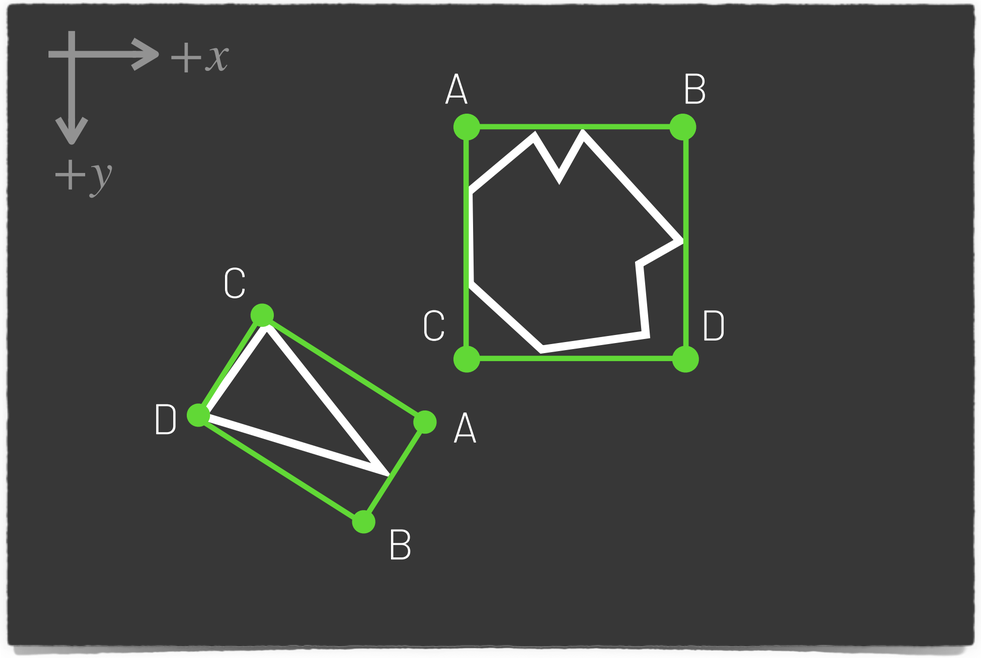
Quando o objeto é rotacionado, a AABB se mantém alinhada com os eixos

Caixa Delimitadora Orientada

Uma **caixa delimitadora orientada, do inglês, *oriented bounding box (OBB)*** é um retângulo sem a restrição de alinhamento com os eixos, ou seja, que pode rotacionar.



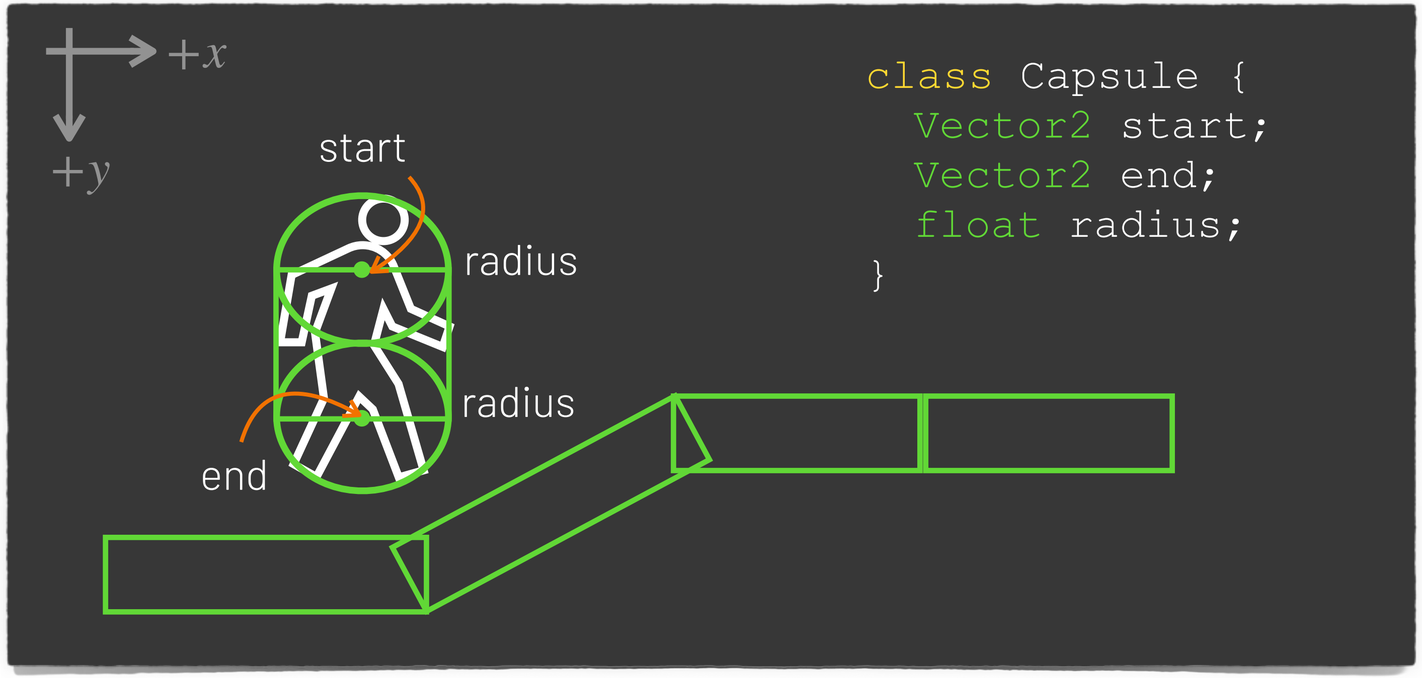
OBBs podem ser representadas por quatro vértices e um ângulo.



Quando o objeto é rotacionado, a OBB também é rotacionada.

Cápsulas

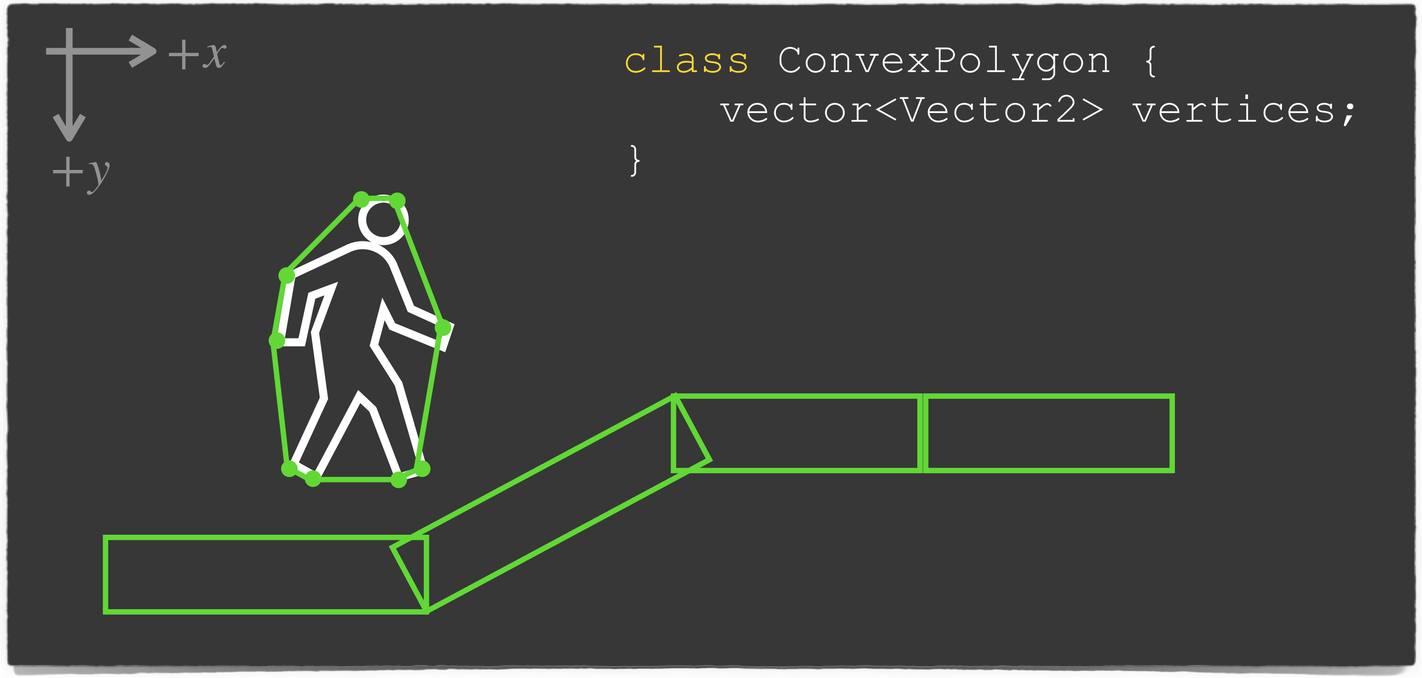
Cápsulas são muito utilizadas como geometrias de personagens humanoides, pois representam melhor o corpo humano e facilitam a detecção de colisão com rampas e escadas.



Cápsulas podem ser representadas por dois pontos e um raio.

Polígonos Convexos

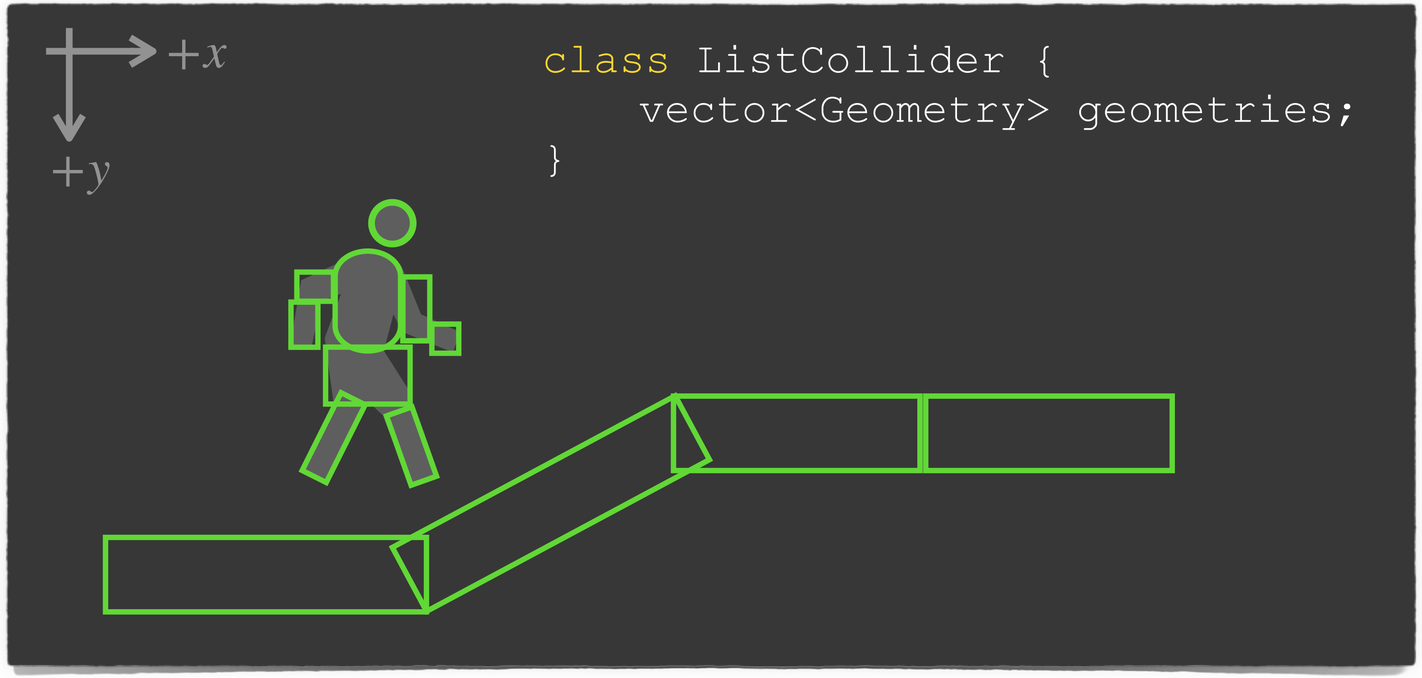
Polígonos convexos são geometrias mais flexíveis, possibilitando um melhor ajuste ao corpo real do objeto, porém a detecção de colisão com eles é mais complexa.



Polígonos convexos podem ser representados por um arranjo unidimensional de n vértices.

Lista de Geometrias

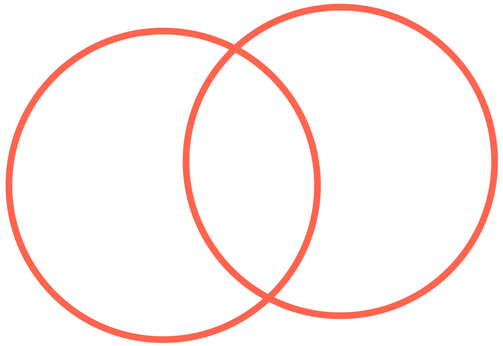
A representação do corpo de um objeto não precisa se limitar a uma única geometria. Podemos utilizar uma **lista de geometrias** para uma melhor aproximação.



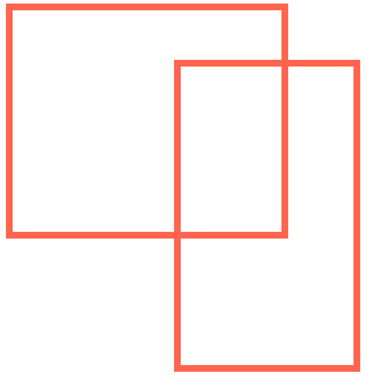
Precisamos definir uma classe base Geometry para agrupar todas as geometrias em uma lista.

Detecção de Colisão

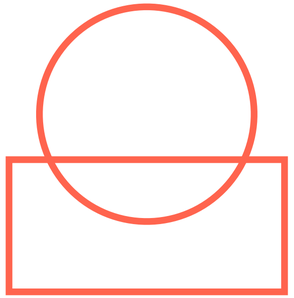
A escolha de geometria para representação dos corpos dos objetos do jogo define os algorimos de detecção de colisão que serão utilizados. Um algoritmo diferente é definido para cada par de geometrias, por exemplo:



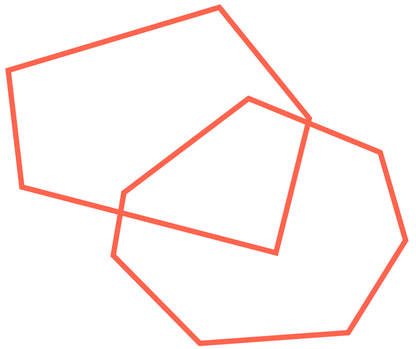
Circunferência vs. Circunferência



AABB vs. AABB



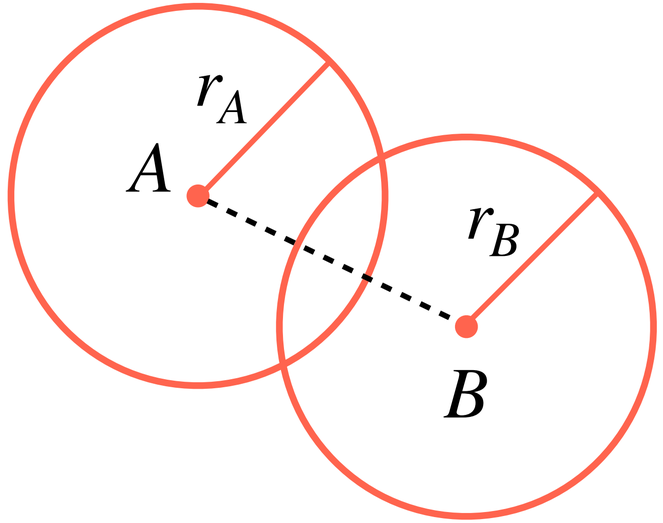
Circunferência vs. AABB



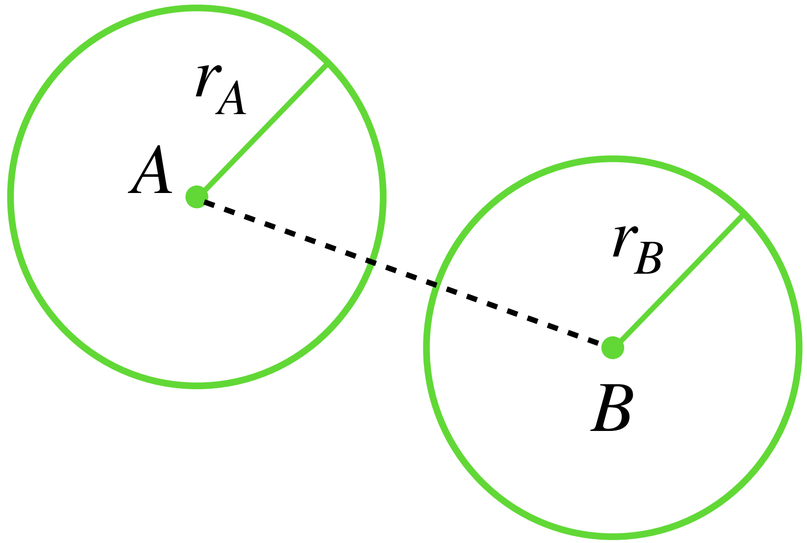
Polígono vs. Polígono (convexos)

Circunferência vs. Circunferência

Duas circunferências estão colidindo quando a distância entre seus centros for menor do que soma dos seus raios.



$$||A - B|| < (r_a + r_b)$$



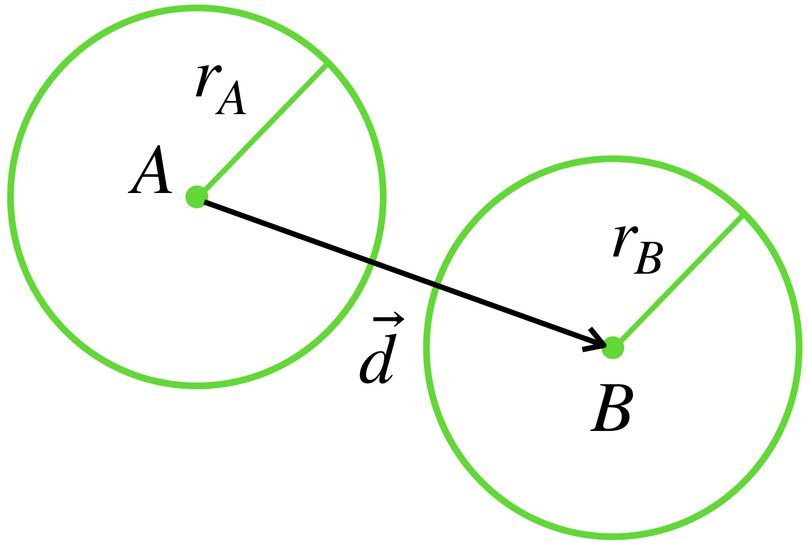
$$||A - B|| > (r_a + r_b)$$

Circunferência vs. Circunferência

Duas circunferências estão colidindo quando a distância entre seus centros for menor do que soma dos seus raios.

- Na prática, para evitar o cálculo de raizes quadradas, comparamos o quadrado da distância entre os centos com o quadrado da soma dos raios
- O quadrado da distância entre os centros pode ser calculado pelo produto escalar do vetor $\vec{d} = B - A$ com ele mesmo:

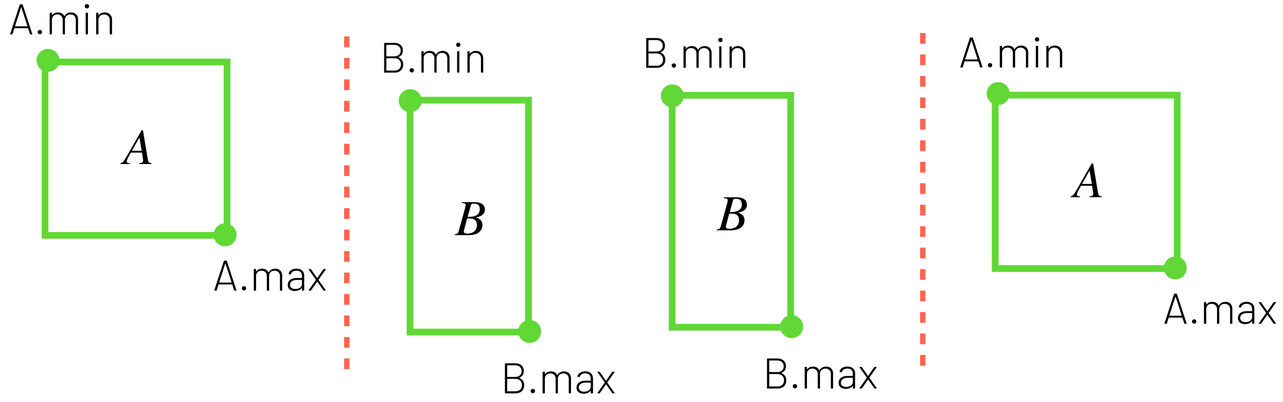
$$\vec{d} \cdot \vec{d} = d_x \cdot d_x + d_y \cdot d_y + d_z \cdot d_z$$



$$\vec{d} \cdot \vec{d} > (r_a + r_b)(r_a + r_b)$$

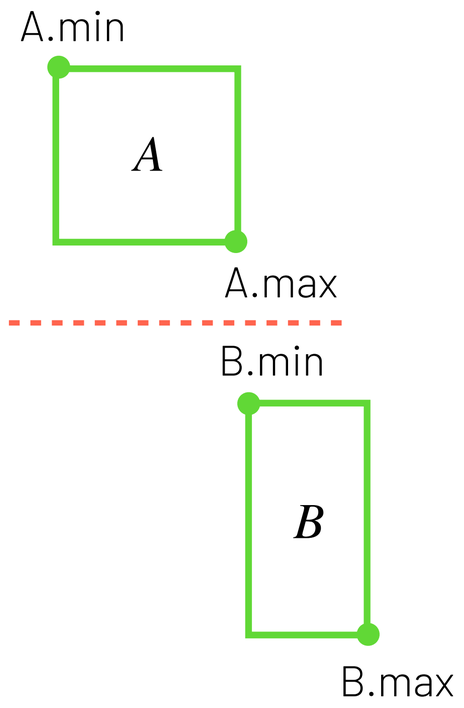
AABB vs. AABB

Para detectar a colisão entre AABBs, é mais fácil verificar os casos em que elas **não** estão colidindo. Se todos esses testes derem falso, elas estão colidindo.

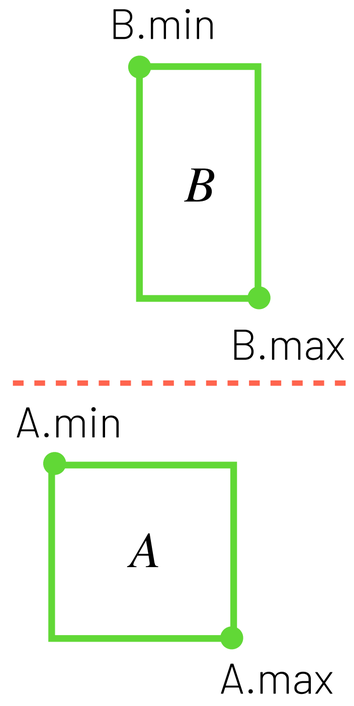


1. A à esquerda de B
A.max.x < B.min.x

2. B à esquerda de A
B.max.x < A.min.x



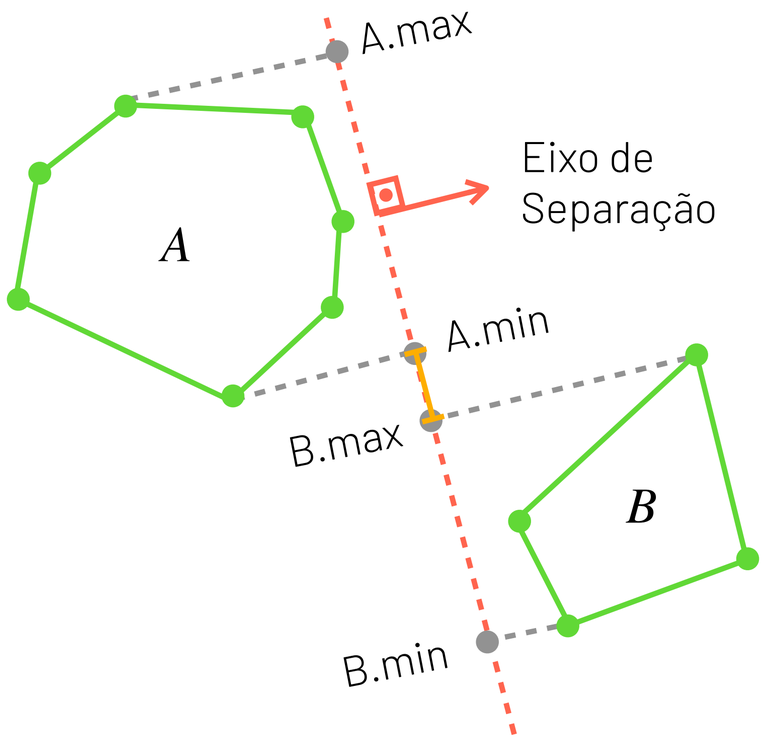
2. A em cima de B
A.max.y < B.min.y



4. B em cima de A
B.max.y < A.min.y

Polígonos Convexos vs. Polígonos Convexos

O algoritmo de verificação de colisão entre polígonos convexos é uma generalização do algoritmo AABB vs. AABB, que chamamos de ***Separation Axis Theorem***

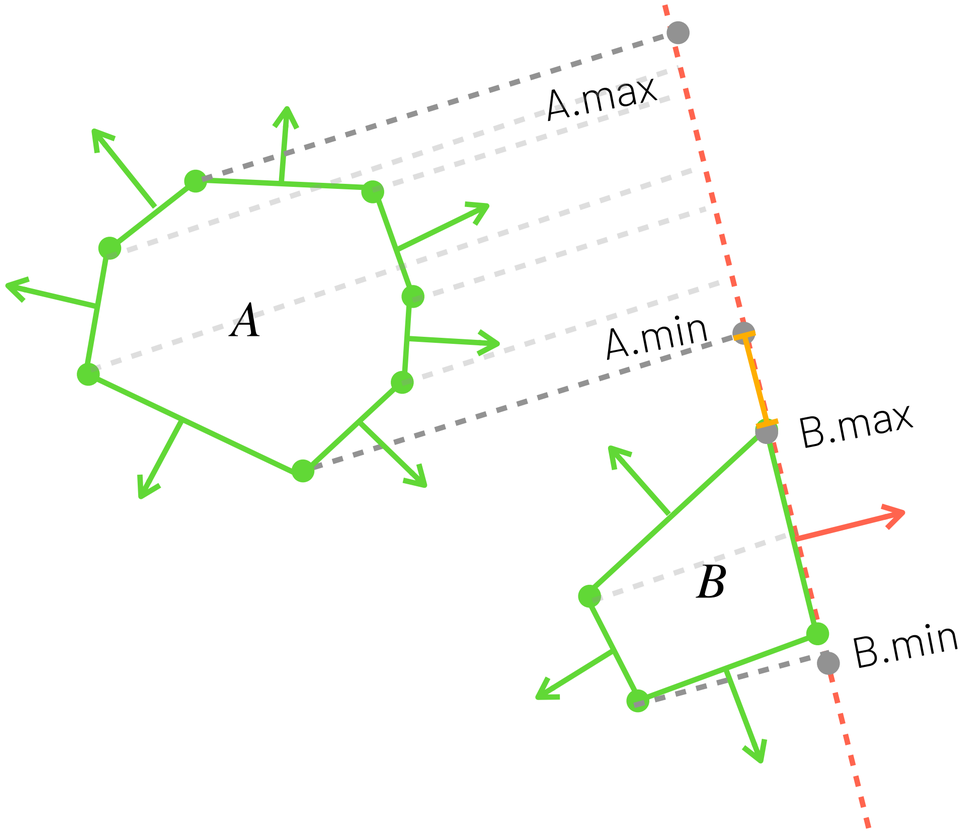


O ***Separation Axis Theorem*** afirma que dois polígonos convexos não colidem se, e somente se, existe pelo menos um eixo onde as projeções ortogonais dos polígonos nesse eixo não se cruzam.

- Em outras palavras, se existe uma reta que separa os dois polígos, eles não colidem.
- Como encontrar essa reta? Ou melhor, quais são as potenciais retas que poderíamos testar?

Polígonos Convexos vs. Polígonos Convexos

O ***Separation Axis Theorem*** nos garante *que* é suficiente testar apenas os eixos definidos pelos vetores normais aos lados dos dois polígonos.

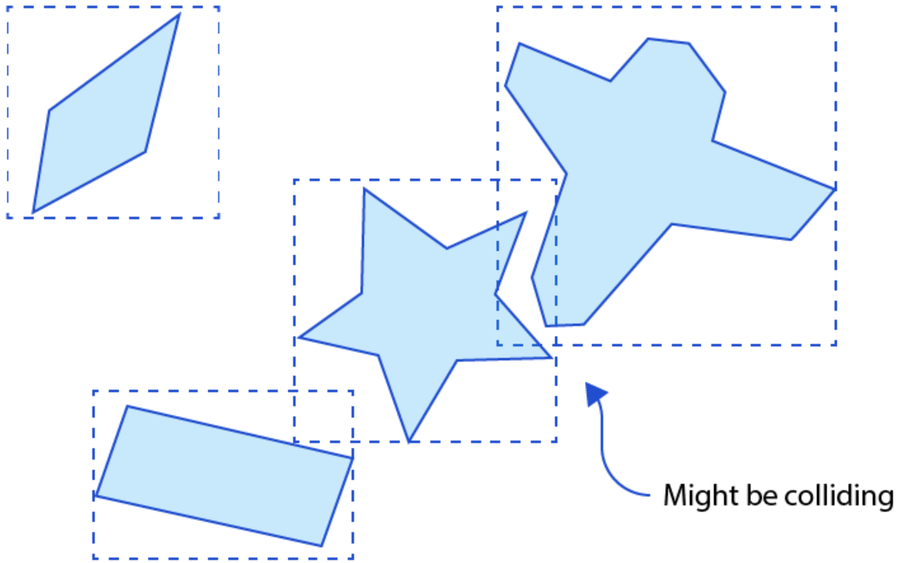


Sendo assim, podemos usar o seguinte algoritmo para verificar colisão entre dois polígonos convexos:

1. *Para cada polígono* $P \in \{A, B\}$
2. *Para cada aresta* $e_i \in P$*:*
3. *Calcule a normal* $\vec{n}$ *a essa aresta* e_i
4. *Projete todos os vértices* $a_j \in A$ *no eixo definido por* $\vec{n}$
5. *Encontre os vértices* $a_{min} \in A$ *e* $a_{max} \in A$
6. *Projete todos os vértices* $b_k \in B$ *no eixo definido por* $\vec{n}$
7. *Encontre os vértices* $b_{min} \in B$ *e* $b_{max} \in B$
8. *Se* $a_{max} < b_{min}$ *ou* $b_{max} < a_{min}$, ***não houve colisão***!
9. *Se nenhum eixo foi encontrado,* ***houve colisão***!

Broad Phase vs Narrow Phase

Em jogos 3D, ou jogos 2D onde as representações dos objetos precisam ser mais precisas, geralmente a verificação de colisão é feita em duas etapas:



1. **Broad Phase**

   Usar geometrias de colisões muito simples para detectar rapidamente objetos em potencial colisão;

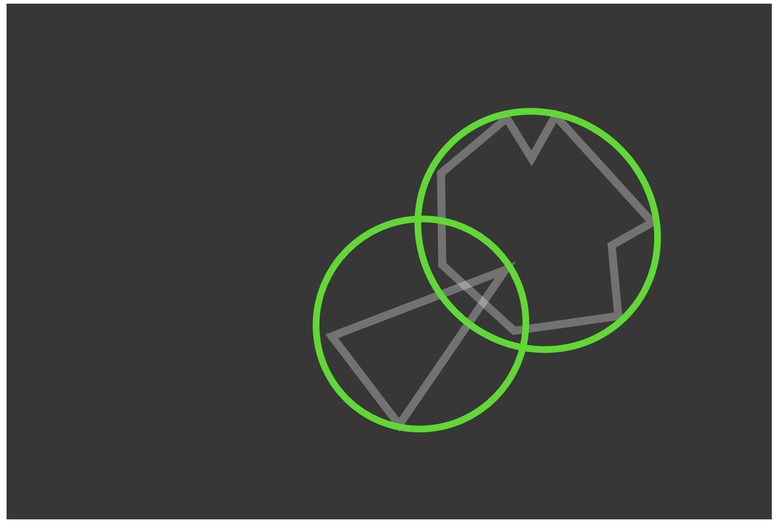
2. **Narrow Phase**

   Usar geometrias de colisões precisas, que aproximam melhor o corpo visual do objeto, visando detectar se colisões realmente ocorreram.

Resolução de Colisão

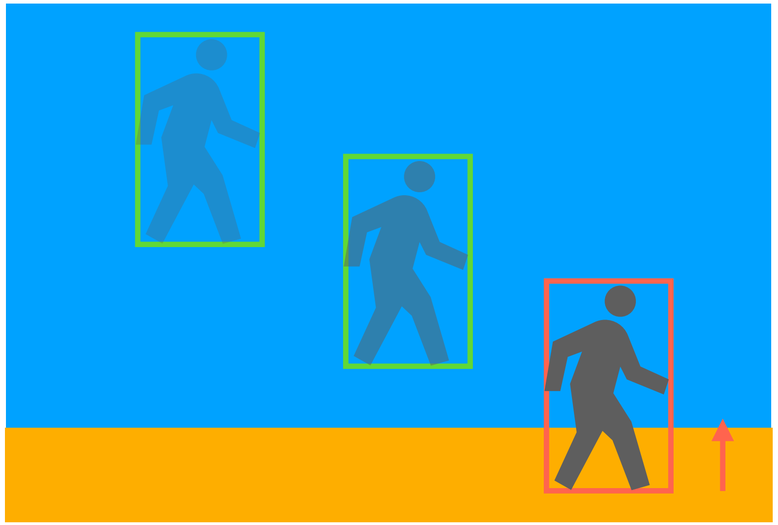
A resolução de uma colisão depende de decisões de design do jogo.

- O caso mais simples é quanto os dois objetos são destruídos após a colisão



Ex: Asteroids
Colisão entre a nave e os meteoros

- Quando os objetos não são destruídos, tipicamente é necessário separar as duas geometrias:



Ex: Super Mario Bros
Colisão entre o Mario e as plataformas

Resolução de Colisão: Separação de AABBs

A resolução de colisão entre AABBs é geralmente realizada separadamente por eixo (horizontal e vertical), pois, em um mesmo quadro, podem haver colisões nos dois eixos:

1. Descobrir qual lado de B colidiu com A

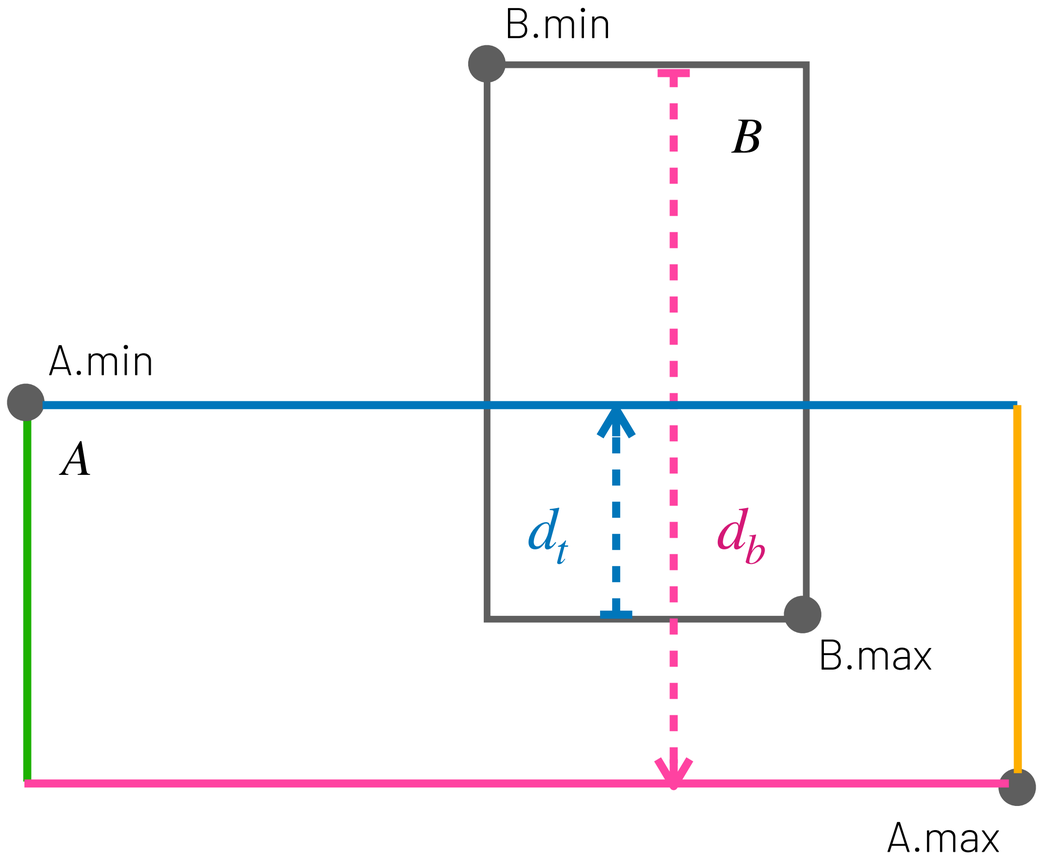
   Basta calcular os vetores entre os lados de A *e seus respectivos lados opostos em* B*:*

   d_t = (0, A.min.y - B.max.y) → cima

   d_b = (0, A.max.y - B.min.y) → baixo

2. Separar A de B após uma colisão:

   Basta somar à posição de B *o vetor de menor comprimento entre* d_t *e* d_b.



Resolução de Colisão: Separação de AABBs

A resolução de colisão entre AABBs é geralmente realizada separadamente por eixo (horizontal e vertical), pois, em um mesmo quadro, podem haver colisões nos dois eixos:

1. Descobrir qual lado de B colidiu com A

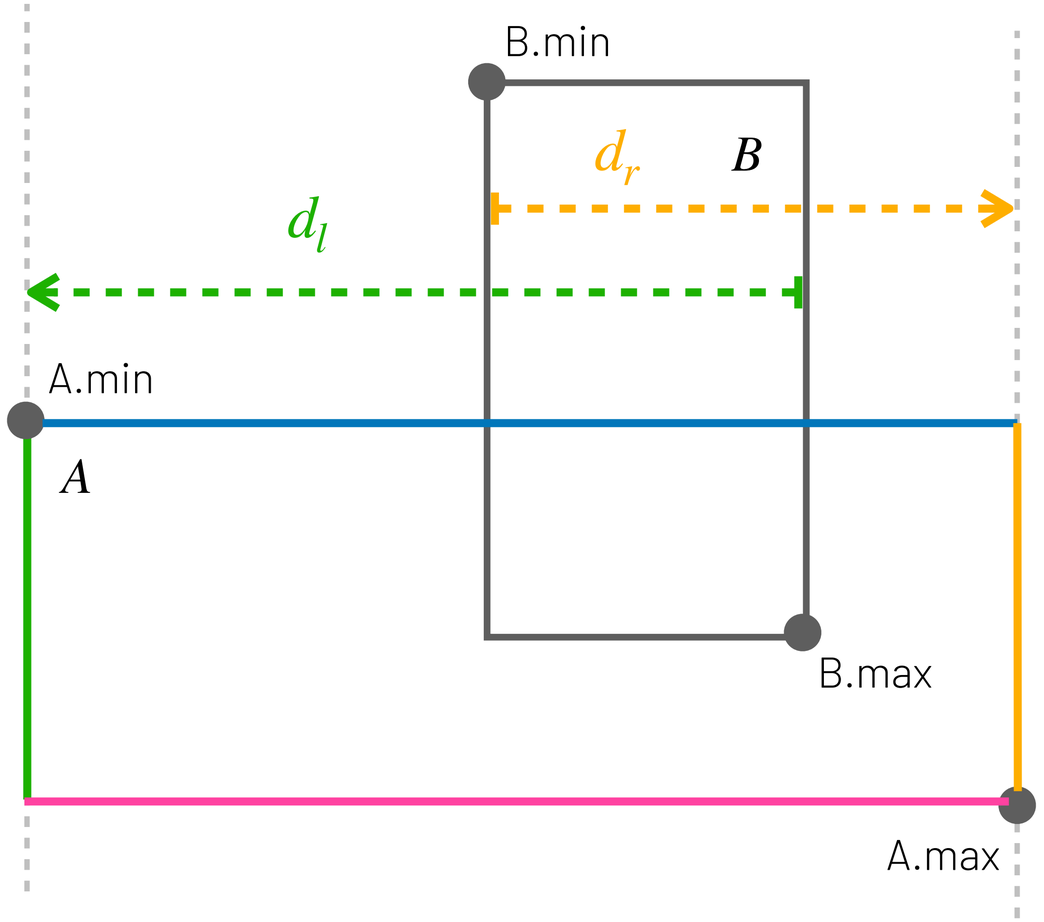
   Basta calcular os vetores entre os lados de A *e seus respectivos lados opostos em* B*:*

   d_l = (A.min.x - B.max.x, 0) → esquerda

   d_r = (A.max.x - B.min.x, 0) → direita

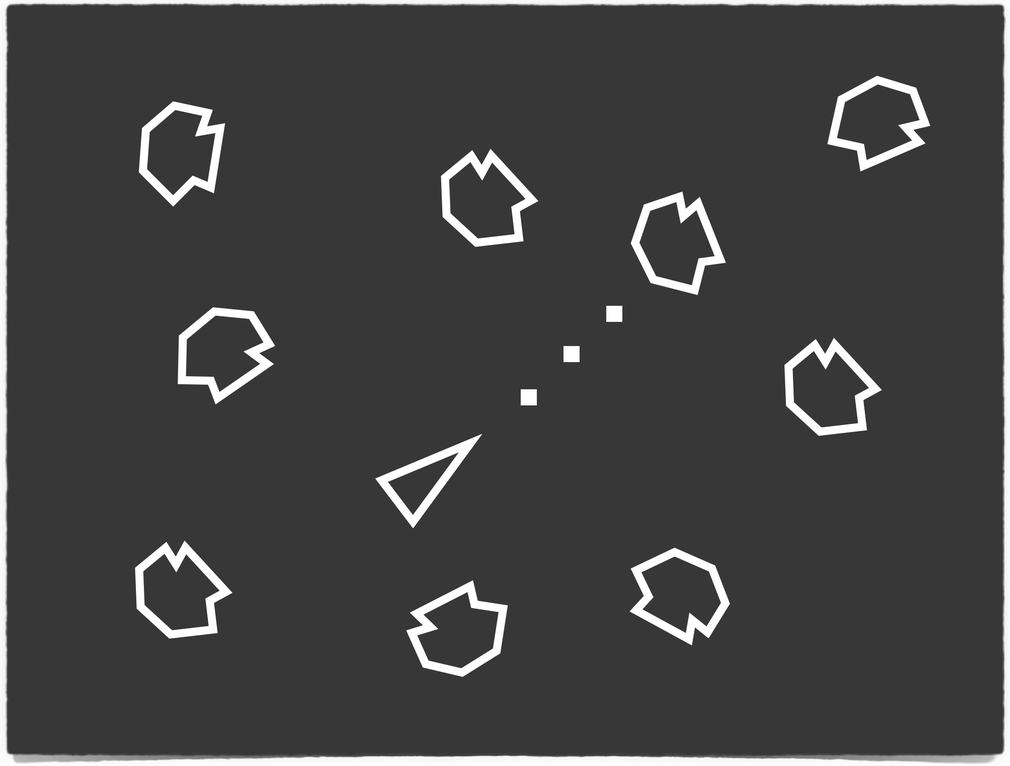
2. Separar A de B após uma colisão:

   Basta somar à posição de B *o vetor de menor comprimento entre* d_t *e* d_b.



Detecção de Colisão (entre múltiplos objetos)

Agora que sabemos algumas técnicas simples para detecção e resolução de colisão, vamos considerar o caso geral onde um objeto pode colidir contra vários objetos:



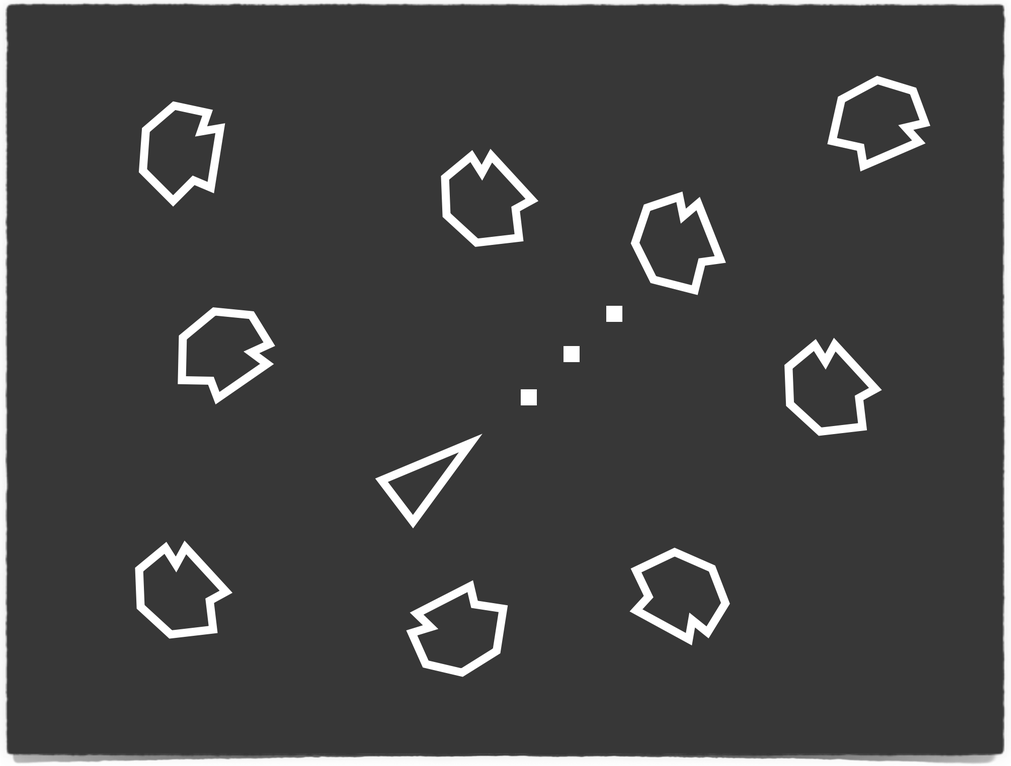
- O algoritmo mais simples consiste em verificar colisão de todos contras todos:

```
for (auto objA : mObjetos)
  for (auto objB : mObjetos)
      if objA != objB
        checkCollision(objA, objB)
```

Esse algoritmo é $O(n^2)$, onde n é o número de objetos em cena. Funciona bem se a cena não tiver muitos objetos, como no Asteroids!

Detecção de Colisão (entre múltiplos objetos)

A vantagem de se implementar uma engine é que podemos otimizar o jogo com base em suas regras. Por exemplo, no Asteroids, os meteoros não colidem entre si, portanto:



- O algoritmo se resume a verificar a colisão entre a nave vs. meteoros e tiros vs. meteoros:

```
for (auto meteor : mMeteors):
    checkCollision(ship, meteor)

for (auto bullet :mBullets)
  for (auto meteor : mMeteors)
    checkCollision(bullet, meteor)
```

Esse algoritmo é $O(nm)$, onde n é o número de meteoros e m o número de tiros.

Implementando Detecção de Colisão

Uma boa forma de implementar detecção de colisão é utilizando um componente que pode ser adicionado à lista de componentes dos objetos que terão colisão do jogo:

```
class BoxCollider : public Component
{
public:
  BoxCollider(Actor* owner, Rect rect);
  void DetectHCol();
  void DetectVCol();

  float GetMinVOverlap(BoxCollider* b);
  float GetMinHOverlap(BoxCollider* b);

  bool Intersect(const BoxCollider& b);

private:
  Vector2 mOffset;
  Vector2 mSize;
};
```

```
void DetectHCol() {
  // Get otherColliders from game
  for (auto b : otherColliders) {
    if(Intersect(b)) {
      float minHOverlap = GetMinHOverlap(b);
      ResolveHCol(minHOverlap);
      // Throw event to owner
      mOwner->OnHCollision(minHOverlap, b);
    }
  }
}
```

```
void DetectVCol(BoxCollider *body) {
  // Similar a função acima
}
```

Próxima aula

A9: Física III - Otimizações de Detecção de Colisão

- Sort and Sweep
- Estruturas de Dados Espaciais
  - Hashing Espacial
  - Quadtree/Octree
- Culling de Região Ativa