INF216

2023/2

UFV

Projeto e Implementação de Jogos Digitais

A5: Física - Detecção de Colisão

Logística

Avisos

- Teste T2 nessa sexta-feira!

Última aula

- Objetos Rígidos
- Física Newtoniana
- Método de Euler Explícito
- Método de Euler Semi-implícito

Plano de Aula

- Geometrias de colisão
- Detecção de colisão
  - Circunferência vs. Circunferência
  - AABB vs. AABB
- Resolução de colisão
- Otimização de colisão

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:

Figura 1: 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.

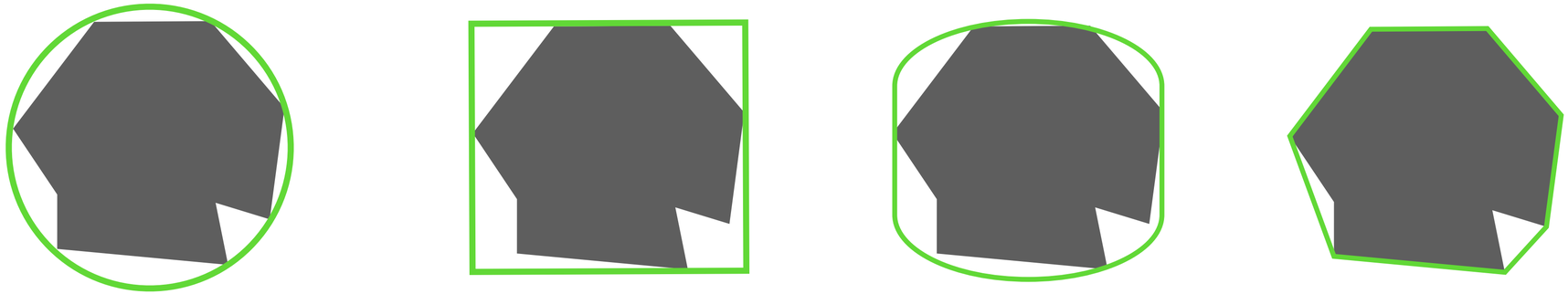


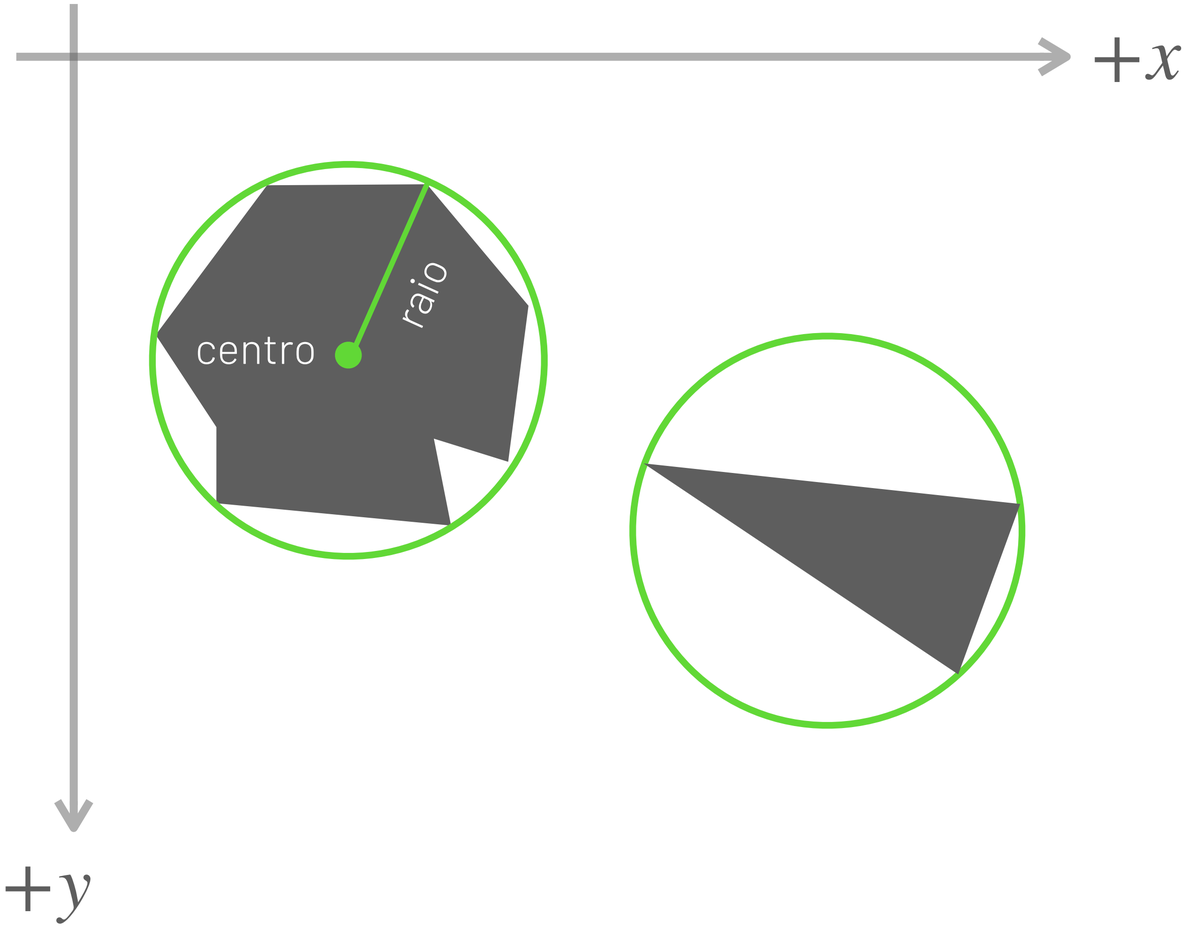
Figura 2: Da esquerda para a direita, exemplos de circunferência, caixa, cápsula e polígono convexo como geometria de colisão de um asteroide.

Circunferência

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

```
class Circle {
    Vector2 center;
    float radius;
}
```

Figura 3: uma circunferência aproxima bem um 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.

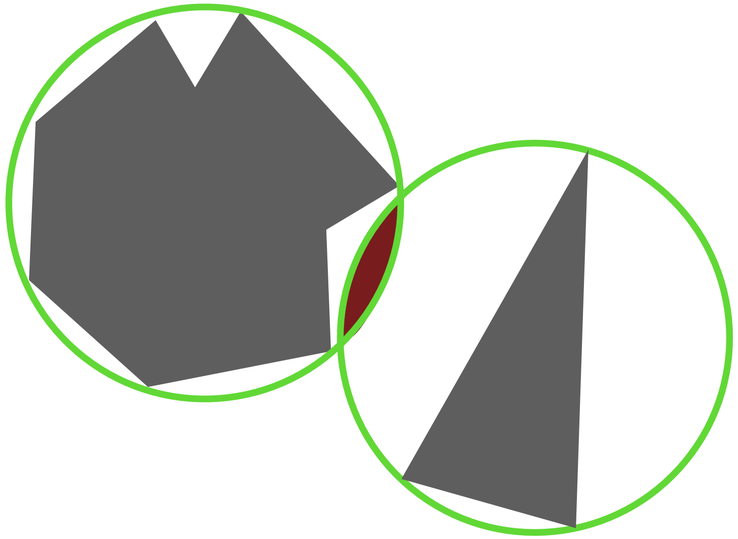


Figura 4: Exemplo de falso positivo em detecção de colisão.

Caixa Delimitadora

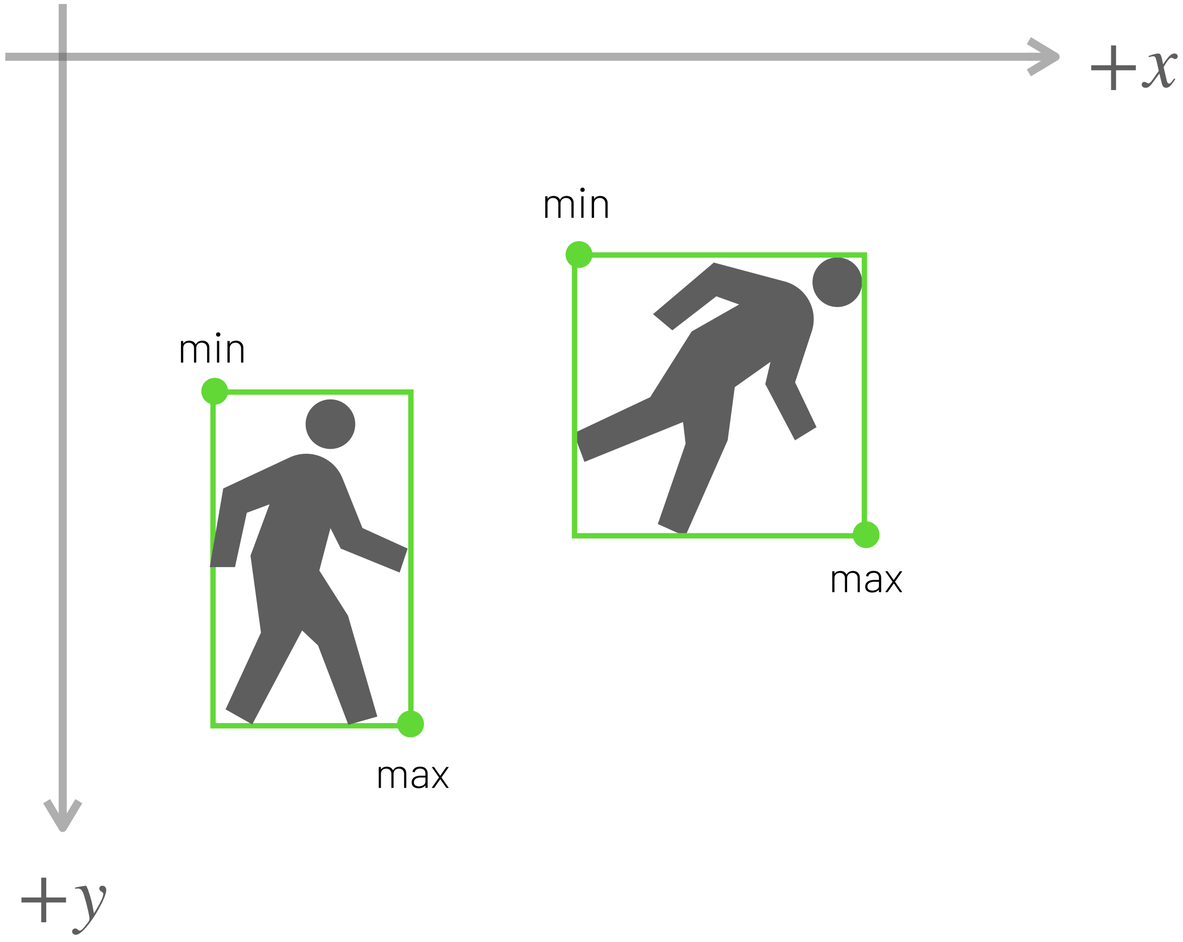
Alinhada com os eixos

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.

```
class AABB {
    Vector2 min;
    Vector2 max;
}
```

Figura 5: exemplos de AABBs como geometrias de colisão de personagens humanos. Note que quando o objeto é rotacionado, a AABB se mantém alinhada com os eixos



Caixa Delimitadora

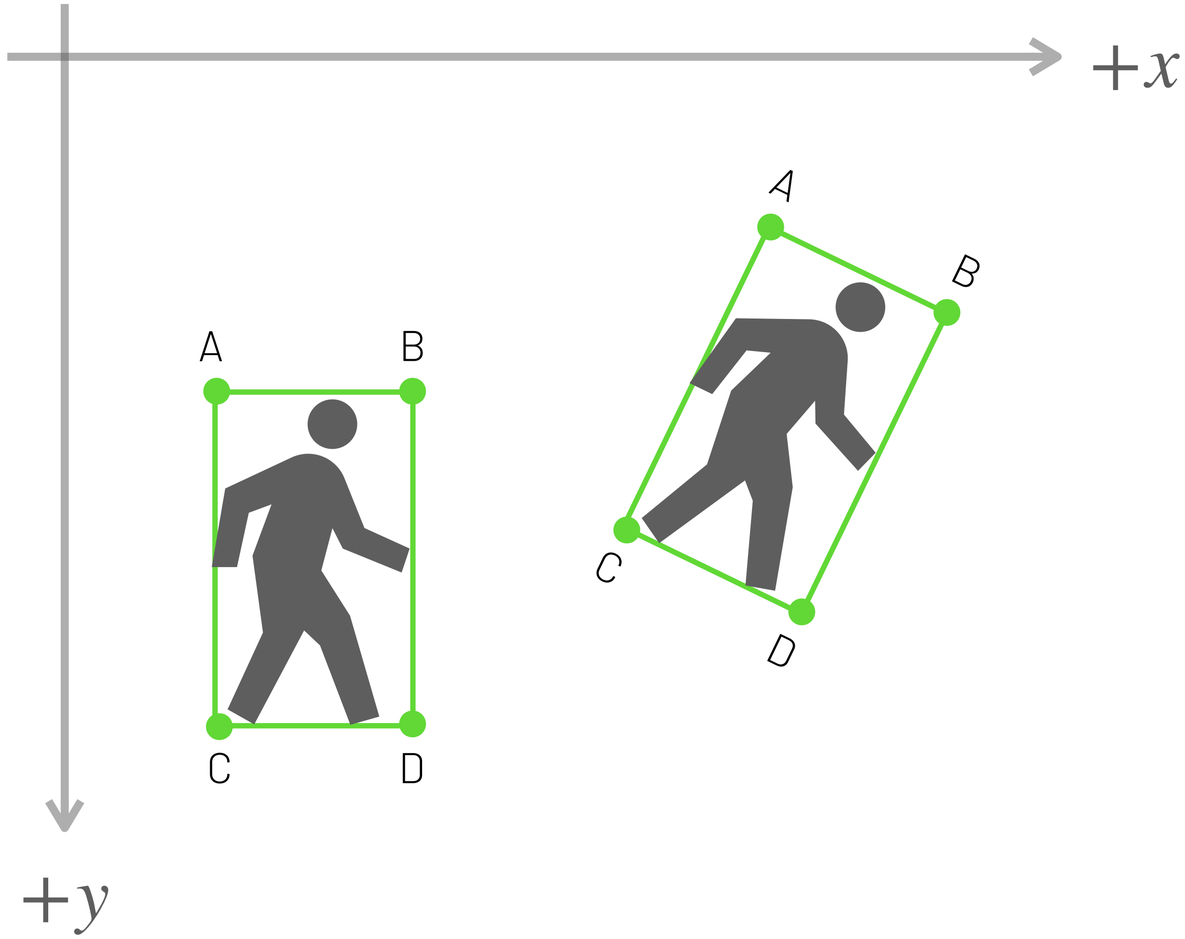
Orientada

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 (A, B, C, D) rotacionados de acordo com o ângulo do objeto.

```
class OBB {
    Vector2 a; Vector2 b;
    Vector2 c; Vector2 d;
}
```

Figura 6: exemplos de OBBs como geometrias de colisão de personagens humanos. Note que quando o objeto é rotacionado, a OBB também é rotacionada.



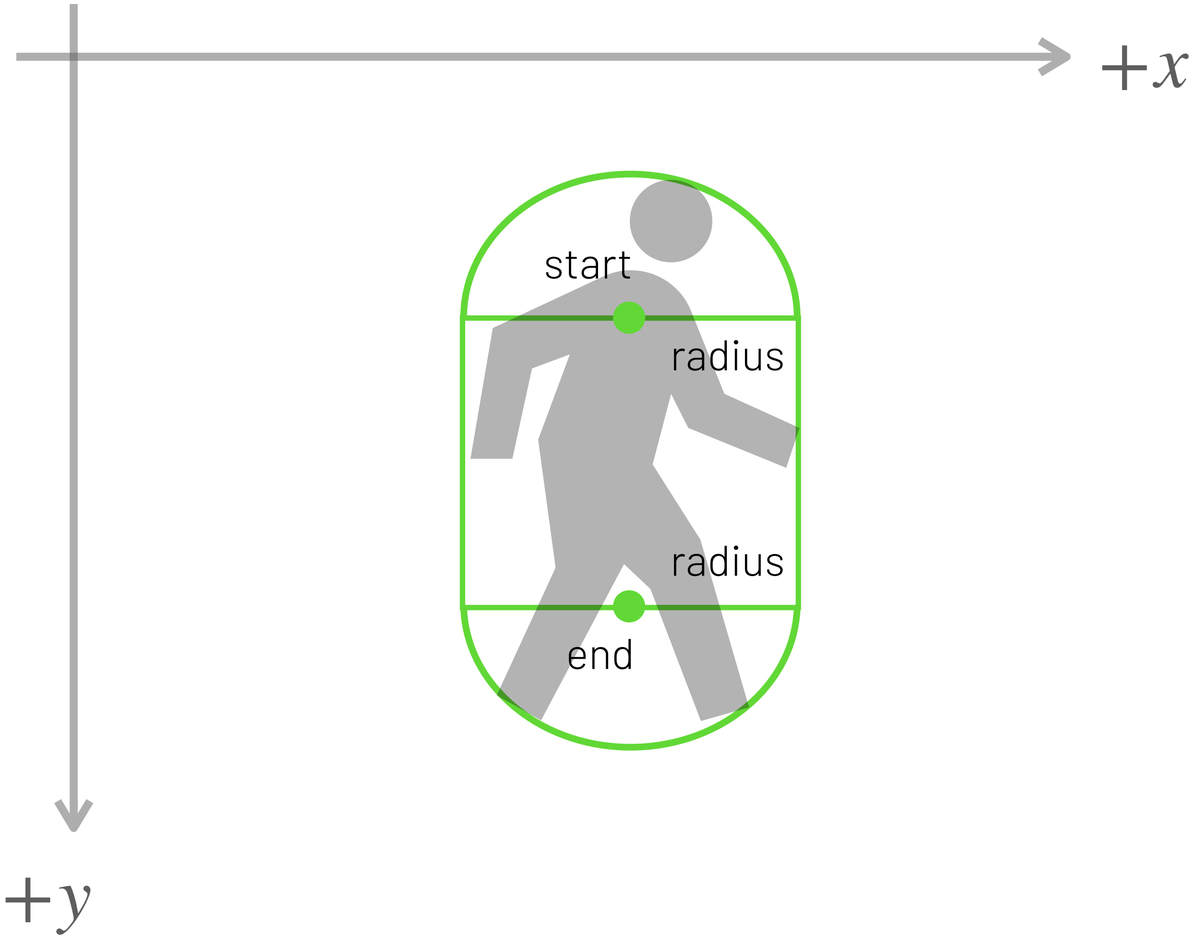
Cápsulas

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.

```
class AABB {
    Vector2 start;
    Vector2 end;
    float radius;
}
```

Figura 7: exemplo de cápsula como geometria de colisão para personagem humanoide.



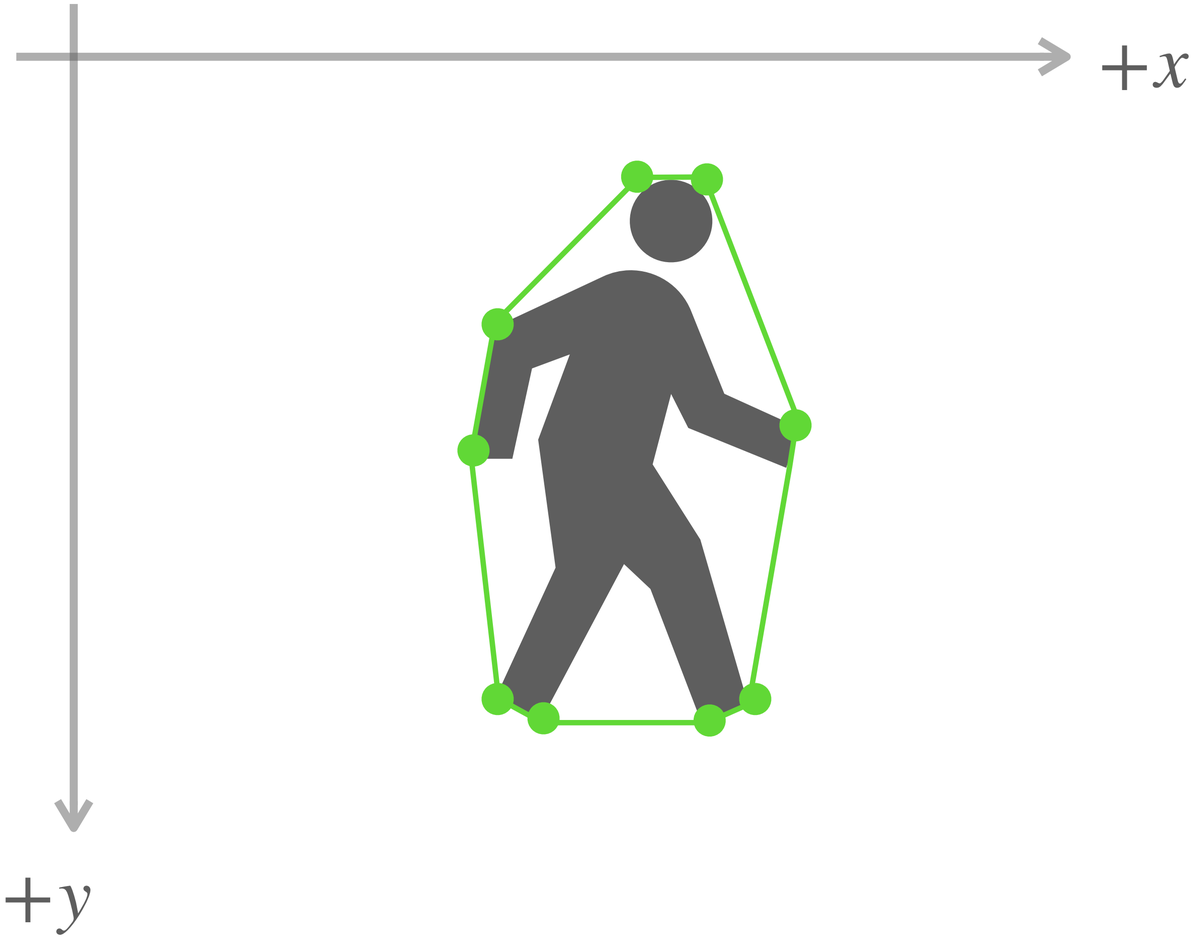
Polígonos Convexos

Polígonos convexos são geometrias mais flexíveis, porém a detecção de colisão com eles é mais complexa.

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

```
class ConvexPolygon {
    std::vector<Vector2> vertices;
}
```

Figura 8: exemplo de polígono convexo como geometria de colisão para personagem humanoide.



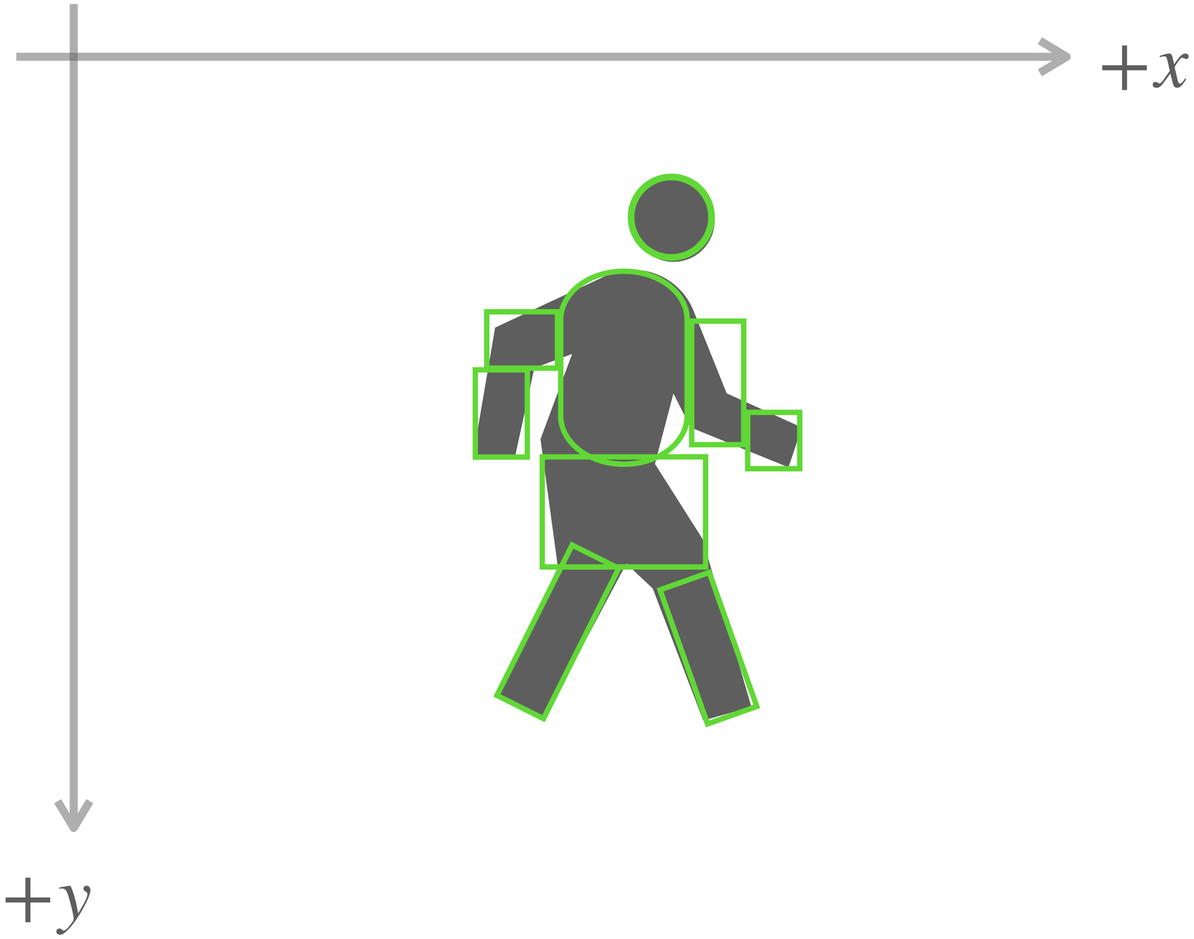
Lista de Geometrias

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 da representação visual do objeto.

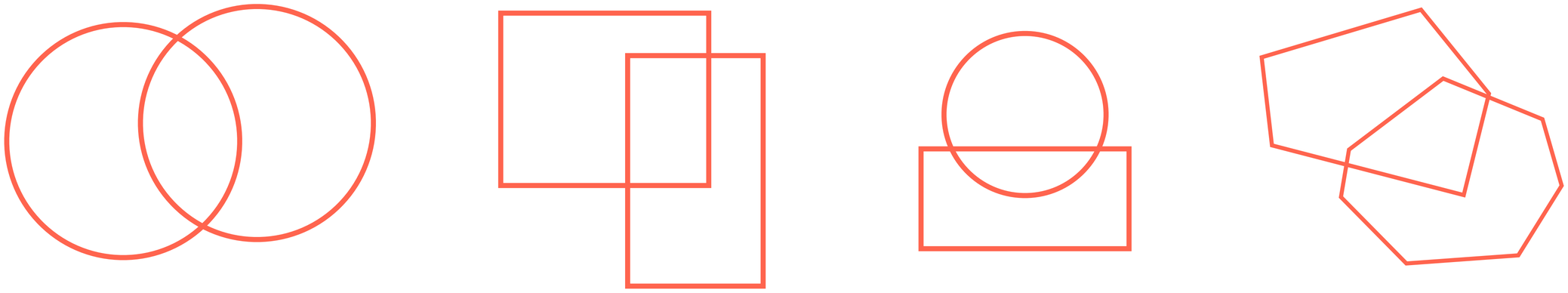
```
class ListCollider {
    std::vector<Geometry> geometries;
}
```

Figura 9: exemplo de lista de geometrias para representação de personagem humanoide.



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

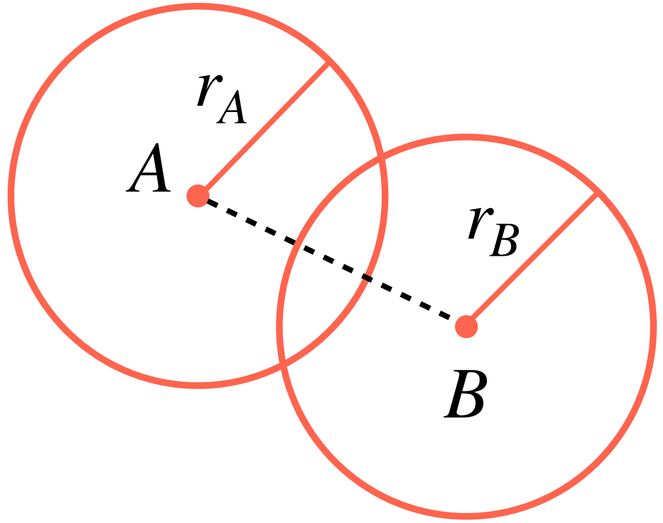
Circunferência vs. AABB

Polígono vs. Polígono (convexos)

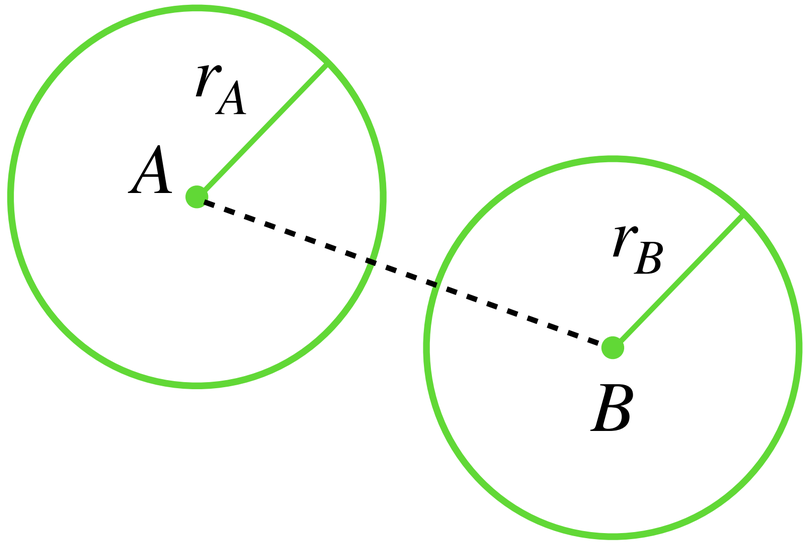
Figura 10: Exemplos de algorimos de detecção de colisão para diferentes pares de geometrias.

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

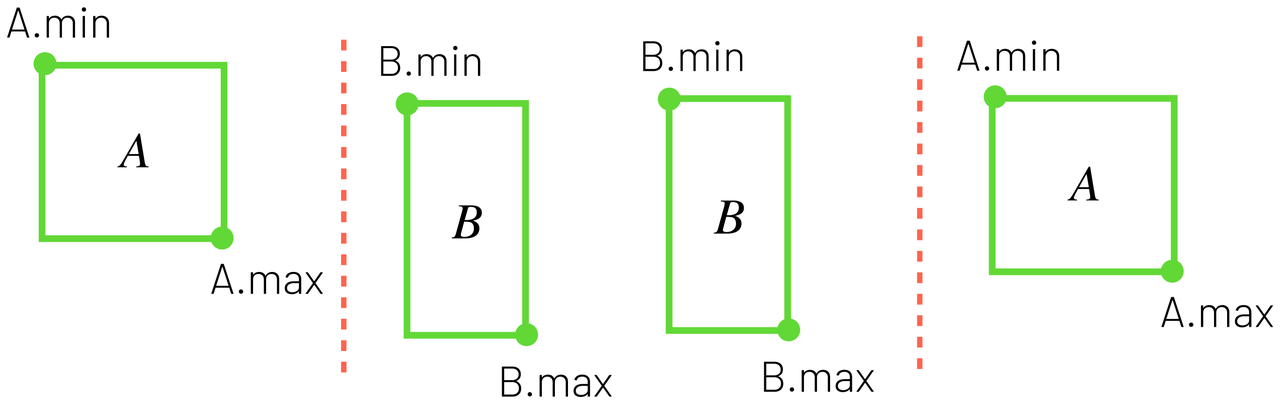
Na prática, para evitar o cálculo de raizes quadradas, comparamos o quadrado da distância entre os centos com o quadrado da soma dos raios.

O quadrado da distância entre os centros pode ser calculado pelo produto escalar do vetor $\vec{d} = B - A$ com ele mesmo.

```
bool CircleIntersection(Circle a, Circle b) {
    Vector2 d = b.center - a.center;

    float distSquared = Dot(d, d);
    float radiiSquared = (a.radius + b.radius) * (a.radius + b.radius);

    return (distSquared < radiiSquared);
}
```

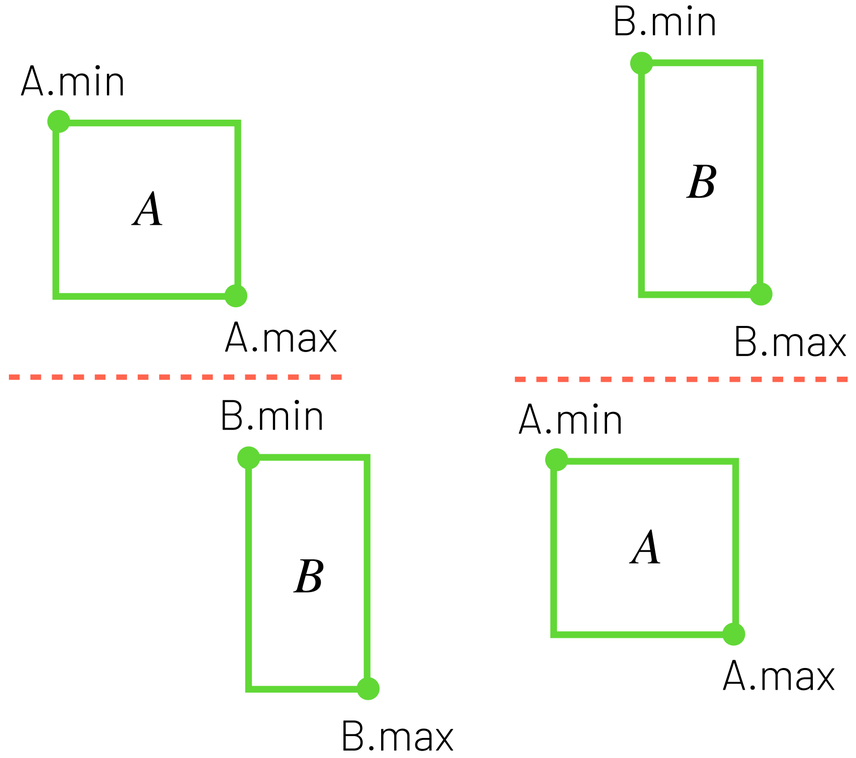
AABB vs. AABB

Para detectar a colisão entre AABBs, é mais fácil verificar os casos em que elas **não** estão colidindo. Se nenhum desses casos forem verdadeiros, 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



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

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

AABB vs. AABB

Para verificar se qualquer um dos quatro casos ocorreu, podemos utilizar uma expressão lógica com operadores OU entre os casos.

Se o resultado dessa expressão for verdadeiro, as AABBs **não** estão colidindo. Assim, a função de detecção deve retornar a negação dessa expressão.

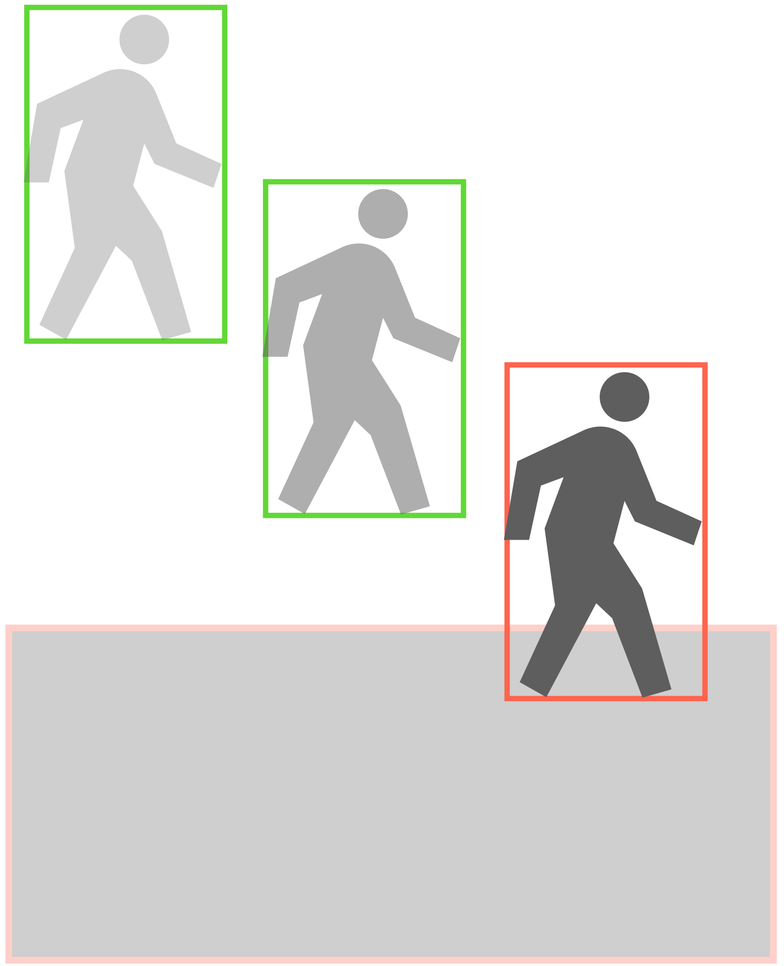
```
bool AABBIntersection(Circle a, Circle b) {
    bool notColliding = (A.max.x < B.min.x) || (B.max.x < A.min.x) ||
                        (A.max.y < B.min.y) || (B.max.y < A.min.x);

    return !notColliding;
}
```

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. (como projéteis em jogos de tiro).

Quando os objetos não são destruídos, tipicamente é necessário separar as duas geometrias (como em colisões de jogos de plataforma).

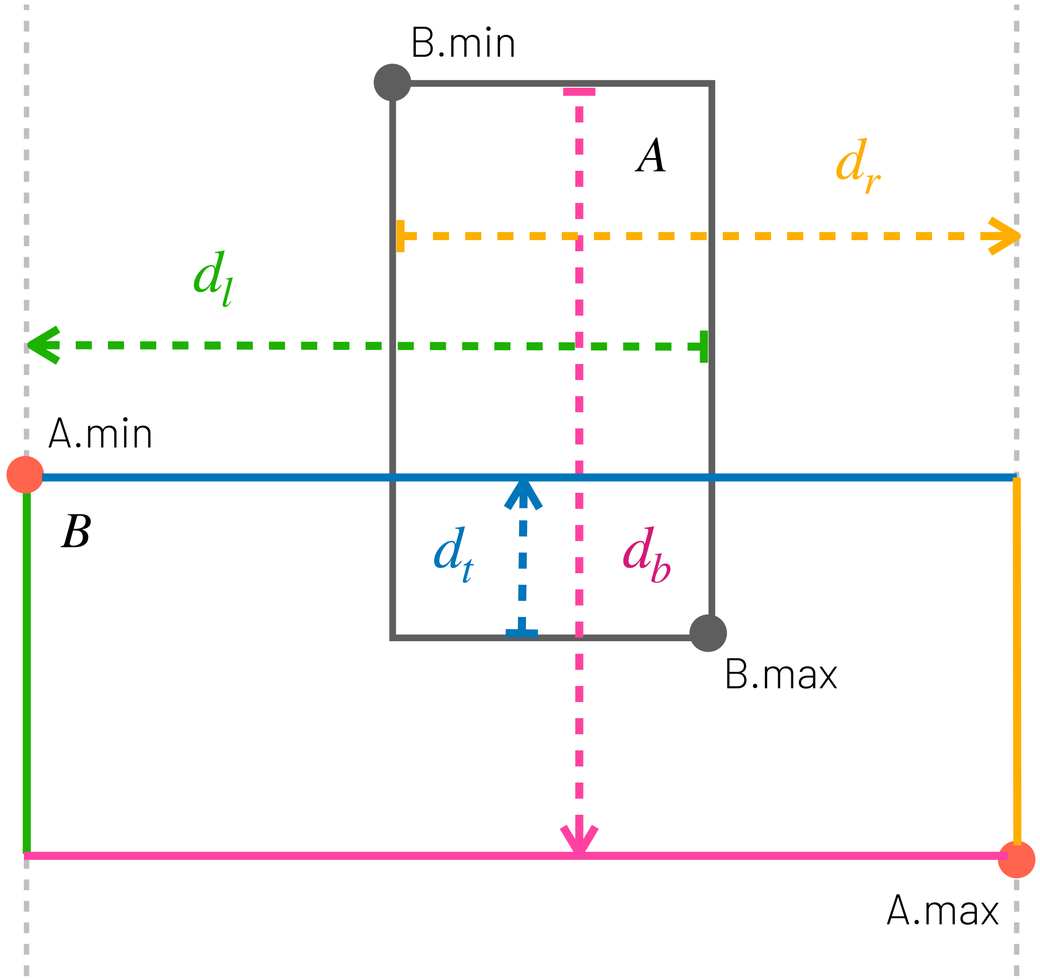


Resolução de Colisão de AABBs

Para descobrir qual lado de B que houve a colisão 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)
d_l = (A.min.x - B.max.x, 0) (esquerda)
d_r = (A.max.x - B.min.x, 0) (direita)

Para separar A de B após uma colisão, basta somar à posição de A o vetor de menor comprimento.

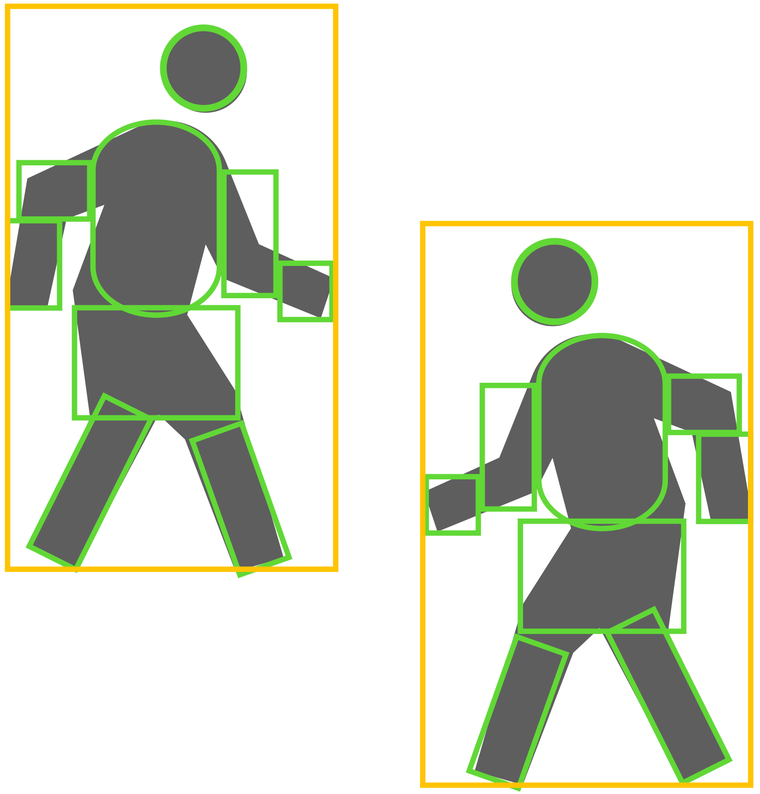


Otimização de Colisão

Quando os objetos do jogo possuem geometrias complexas (e.g., listas de geometrias), podemos otimizar a detecção de colisão com um algoritmo hierárquico:

1. Verificar a proximidade dos objetos com geometrias mais simples (e.g., **AABB**).

2. Caso haja colisão em 1., verificar as colisões das geometrias **mais complexas**.



Próximas aulas

A6: Gráficos 2D

Utilizando texturas para representar objetos 2D estáticos/animados e compor cenas com múltiplas camadas.

L6: Pacman - Parte 1

Implementar um componente RigidBody para a movimentação de objetos rígidos.