Biblioteca de Física Box2D

Márcio Sarroglia Pinho
Isabel Harb Manssour

Introdução

- Biblioteca 2D
- Objetos não deformáveis
- Módulos
  - Common
  - Collison
  - Dynamics
- Não faz *rendering*

Unidades

- Metros
- Kilogramas
- Segundos

Unidades

- Conversões entre Pixels e Metros
- A Box2D funciona bem em universos de 1 a 10 metros
- Pode ser necessário converter as coordenadas da tela para funcionar com a Box2D

Mundo

- Ambiente onde ocorre a simulação

```
b2World *world;
b2Vec2 gravity(0.0f, -10.0f);
bool doSleep = true;  //objetos dormir ?
world = new b2World(gravity, doSleep);
```


Mundo

- Formado por CORPOS
- São objetos que participam da simulação
- Componentes
  - Atributos
  - Fixtures

CORPOS

- Criação de Corpo

```
// Cria as definicoes do CORPO.
b2BodyDef bodyDef;
bodyDef.position.Set(posx, posy);
bodyDef.type = b2_dynamicBody ;

// Cria um novo corpo a partir da informações da bodyDef
// Nunca use NEW ou MALLOC para criar um corpo
b2Body* body = world->CreateBody(&bodyDef);
```


CORPOS
Atributos - type

- struct b2BodyDef::type
  - Define o tipo do corpo. É obrigatório.
    - *b2_staticBody*
    - *b2_kinematicBody*
    - *b2_dynamicBody*

CORPOS
Atributos - type

- *b2_staticBody*
  - Não se move durante a simulação
  - Tem massa infinita e velocidade zero
  - Pode ser movido manualmente pelo programa
  - Usado para criar obstáculos

CORPOS
Atributos - type

- *b2_kinematicBody*
  - Move-se de acordo com sua velocidade.
  - Não responde a forças.
  - Pode ser movido manualmente pelo programa, mas em geral deve ser movido por sua velocidade.

CORPOS
Atributos - type

- *b2_dynamicBody*
  - tem seu comportamento totalmente simulado
  - Pode ser movido manualmente pelo programa, mas em geral deve ser movido pelas forças a ele aplicadas.
  - Tem massa finita e diferente de zero.
  - Se for tentado colocar massa zero, ela é convertida para 1kg.

CORPOS
Atributos

- struct b2BodyDef::Position
  - Define a posição do corpo.
  - Deve ser usado antes de criar o objeto
- struct b2BodyDef::angle
  - Define a orientação do corpo.
  - Em radianos

CORPOS
Atributos

- struct b2BodyDef::linearDamping
  - Reduz a velocidade do objetos durante uma translação.
  - Não depende de fricção.
  - Usar entre 0 e 0.1.
- struct b2BodyDef::angularDamping
  - Reduz a velocidade do objetos durante uma rotação.
  - Não depende de fricção.
  - Usar entre 0 e 0.1.

CORPOS
Atributos

- struct b2BodyDef::awake
  - true/false
  - Define manualmente como o objeto está
  - Um objeto com awake == false não é simulado
  - Permite melhorar o desempenho da simulação
  - Se estiver dormindo e outro objeto colidir nele, ele acorda.

CORPOS
Atributos

- struct b2BodyDef::allowSleep
  - true/false
  - Permite que um objeto deixe de ser simulado.
  - A detecção é feita pela própria BOX
  - Melhora a performance

CORPOS
Atributos

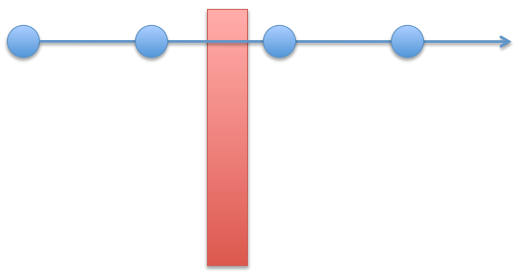
- struct b2BodyDef::fixedRotation
  - true/false
  - Evita que o corpo sofra rotações
  - Mesmo que alguma força seja aplicada sobre ele.

CORPOS
Atributos

- struct b2BodyDef::bullet
  - true/false
  - Afeta somente corpos dinâmicos.
  - Usado para forçar o uso de CCD(continuous collision detection)
  - Reduz o desempenho
  - Usa-se em objetos que se movem muito rápido no cenário

CORPOS
Atributos

- struct b2BodyDef::bullet
- Continuous Collision Detection



CORPOS
Atributos

- struct b2BodyDef::active
  - true/false
  - Define se um objeto participa ou não dos cálculos da física.

CORPOS
Atributos

- struct b2BodyDef::userData
  - ponteiro *void*
  - Pode ser usado para guardar dados da aplicação.
  - É usado em geral para guardar uma referência ao objeto que é simulado pelo corpo

CORPOS
Fixtures

- Definem as propriedades físicas do objeto
- A estrutura de dados para isto é a **b2FixtureDef**
- Atributos
  - Forma
  - Densidade
  - Fricção
  - Restituição

CORPOS
Fixtures

- Criação de uma Fixure

```
b2FixtureDef fixture;
// Cria uma nova Fixture a partir dos parâmetros
da física do objeto a partir dos dados da b2FixtureDef
// Nunca use malloc ou New
body->CreateFixture(&fixture);
.....
body->DestroyFixture(&fixture);
```

CORPOS
Fixtures

- struct b2FixtureDef::shape
  - Define a forma do objeto
  - Pode ser uma AABB ou uma OOBB. Isto deve ser definido com o método SetAsBox

CORPOS
Fixtures

- struct b2FixtureDef::shape
- AABB
  - void SetAsBox(float32 hx, float32 hy);
  - hx: metade da altura da AABB
  - hy: metade da largura da AABB

CORPOS
Fixtures

- struct b2FixtureDef::shape
- OOBB
  - void SetAsBox(float32 hx, float32 hy, const b2Vec2& center, float32 angle);
  - center : centro da box em coordenadas locais
  - angle: angulo de rotação da box

CORPOS
Fixtures

- struct b2FixtureDef::density
  - Usada para calcular a massa do objeto
  - Toma como base a área
  - Pode ser positiva ou nula.

CORPOS
Fixtures

- struct b2FixtureDef::friction
  - Define o coeficiente de atrito entre objetos
  - Usada para simular o deslocamento de um objeto sobre outro.
  - O atrito dinâmico e o estático são iguais
  - Usa-se entre 0 e 1, mas pode ser qualquer número não negativo.

CORPOS
Fixtures

- struct b2FixtureDef::friction
  - A fricção resultante entre dois objetos

$$FR = \sqrt{F1 * F2}$$

CORPOS
Fixtures

- struct b2FixtureDef::restitution
  - Usado para fazer objetos saltarem no momento de uma colisão.
  - Usa-se valores entre 0 e 1.
  - Um valor 0 define uma colisão inelástica
    - o objeto não salta
  - Um valor 1 define uma colisão perfeitamente elástica
    - A velocidade do objeto depois da colisão é a mesma de antes da colisão

CORPOS
Fixtures

- struct b2FixtureDef::restitution
  - A restituição resultante entre dois objetos é dada pela fórmula

$$RR = \max(R1, R2)$$

  - Para velocidades muito baixas
    - Usa colisões inelásticas
      - Evita jitter

Simulação

- Executada em duas fases
  - Velocity phase
    - Determina os impulsos produzidos pelos objetos uns nos outros
  - Position Phase
    - Ajusta a posição dos objetos apra reduzir as colisões
- Processos iterativos
    - Geram aproximações cada vez melhores
    - Muitas iterações
      - Simulação mais realista
      - Maior tempo de simulação

Simulação

- Executa com base em *Time Steps*
- Define-se o tempo em que se quer saber o resultado da simulação

```
timeStep = 1/60
world->Step(timeStep, velocityIterations,
positionIterations);
```

Simulação

- Gera posições e Orientações para cada corpo

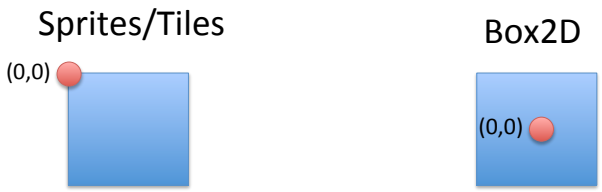
```
// obtém o valor atualizado do personagem
   b2Vec2 position = CorpoPersona->GetPosition();
   float32 angle = CorpoPersona->GetAngle();
// converte para a Pixels

// atualiza o personagem
```


Simulação

- Obter Informações de todos os objetos

```
void PrintBodies()
{
    b2Body *b;
    float ang;
    b2Vec2 pos;
    for(b = world->GetBodyList(); b; b=b->GetNext())
    {
      pos = b->GetPosition();
      ang = b->GetAngle();
      printf("%4.2f %4.2f %4.2f\n", pos.x, pos.y, ang);
    }
}
```

Detalhes de Posicionamento



Aplicação de Rotações

- Torque
  - Provoca um giro no objeto ao redor de seu centro de massa

```
Corpo->ApplyTorque (-1000);
```

Aplicação de Rotações

- Impulso Angular
  - Provoca um giro no objeto ao redor de seu centro de massa

```
Corpo->ApplyAngularImpulse(-1000);
```


Aplicação de Forças

- Usa vetores e posições *globais*

```
b2Vec2 Pos (0.0f, 0.0f);
b2Vec2 Forca (1000.0f, 0);
Pos = Corpo->GetWorldPoint(Pos);
Forca = Corpo->GetWorldVector(Forca);
```

- Se for feito fora do Centro de Massa, causa rotação do corpo

Aplicação de Forças

- Impulso Linear

```
b2Vec2 impulso(0,-1000);
b2Vec2 pos;
pos = Corpo->GetWorldCenter();
Corpo->ApplyLinearImpulse(impulso, pos);
```

- Se for feito fora do Centro de Massa, causa rotação do corpo

Aplicação de Forças

- Impulso Linear

```
b2Vec2 impulso(0,-1000);
b2Vec2 pos;
pos = Corpo->GetWorldCenter();
// altera o vetor de impulso
impulso = Corpo->GetWorldVector(impulso);
Corpo->ApplyLinearImpulse(impulso, pos);
```

- Se for feito fora do Centro de Massa, causa rotação do corpo