Box2d

What is Box2d:

A 2d physics engine designed for game development: It handles “all” calculations related to physics for you. It was developed by Erin Catto in the programming language c++. But it has since been ported into many other languages such as python.

Usage

Box2d handles/simulates “all” physics for you. You input your objects in the world. Set the parameters(ex gravity) define the amount of time to simulate and it calculates what will happen.

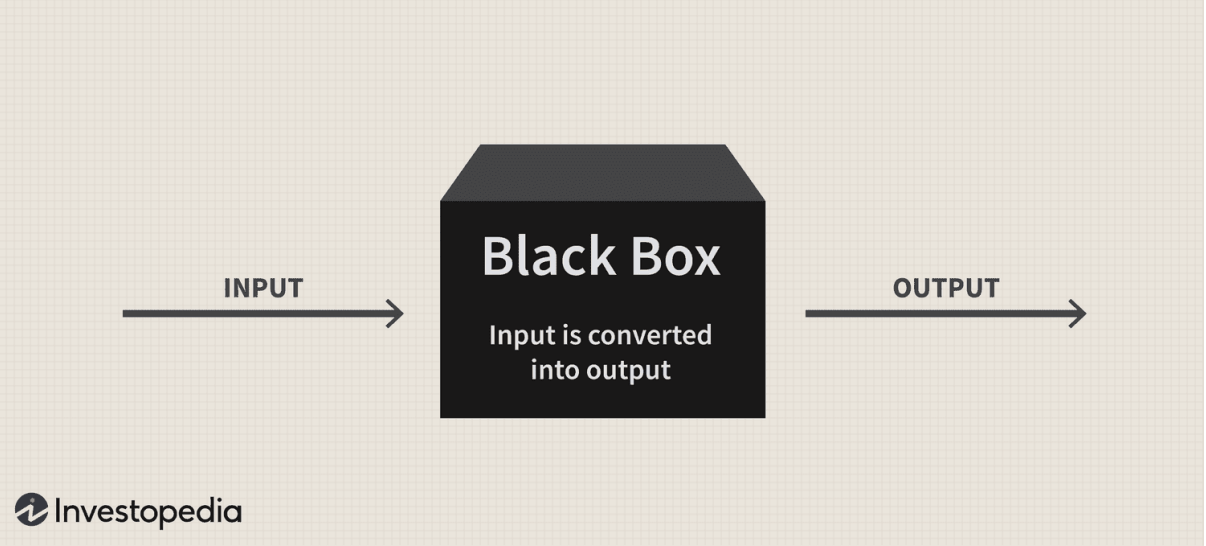


Image by Julie Bang © Investopedia 2019
More about black boxes: https://www.investopedia.com/terms/b/blackbox.asp

Unless you run into problems Box2D reliefs you from the need to understand the physics calculations or the exact implementation of these. You should and can use it as a black box into which you input your objects and it spits in what state those objects will be after the specified time step. This is a good example of the programming

> Abstraction can be defined as a process of handling complexity by hiding unnecessary information from the user. This is one

principle abstraction which helps make your code easier to understand and maintain.

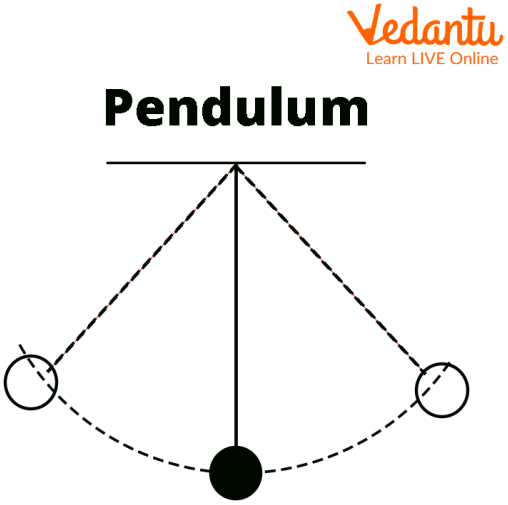
> of the core concepts of object-oriented programming (OOP) languages.

Overview

Core concepts:

- **Shape**
  - A shape is 2D geometrical object, such as a circle or polygon.
- **Fixture**
  - A fixture binds a shape to a body and adds material properties such as density, friction etc.
  - A fixture puts a shape into the collision system so that it can collide with other shapes.
- **Constraint**
  - A constraint is a physical connection that removes degrees of freedom from bodies.
  - A 2D body has 3 degrees of freedom (two translation coordinates and one rotation coordinate).

✨ Example:
If we take a body and pin it to the wall (like a pendulum) we have constrained the body to the wall. At this point the body can only rotate around the pin, so the constraint has removed 2 degrees of freedom.



source: https://www.vedantu.com/evs/uses-of-pendulum

- **World**
  - A physics world is a collection of bodies, fixtures, and constraints that interact together. Box2D supports the creation of multiple worlds, but this is usually not necessary or desirable.
- **Joint**
  - This is a constraint used to hold two or more bodies together.
  - Box2D supports several joint types: revolute, prismatic, distance, and more. Some joints may have limits and motors.
  - Can contain joint limits:
    - A joint limit restricts the range of motion of a joint. For example, the human elbow only allows a certain range of angles.

Bodies

You fill you Box2D world up with bodies of the 3 following types. The physics are simulated using these bodies. Box2D supplies you with functions to interact with these bodies. Such as applyForce, setLinearVelocity, applyLinearImpulse etc.x

- Static
  - A static body does not move under simulation and behaves as if it has infinite mass

    - Static bodies can be moved manually by the user.
    - A static body has zero velocity. Static bodies do not collide with other static or kinematic bodies.
- Kinematic
    - A kinematic body moves under simulation according to its velocity. Kinematic bodies do not respond to forces(ex gravity).
    - Kinematic bodies can be moved manually by the user, but normally a kinematic body is moved by setting its velocity.
    - A kinematic body behaves as if it has infinite mass
    - Kinematic bodies do not collide with other kinematic or static bodies.
- Dynamic
    - A dynamic body is fully simulated. They can be moved manually by the user, but normally they move according to forces.
    - A dynamic body can collide with all body types.

Box2D with Pygame

Pixels

Problem

Box2D uses internal settings which are made to work well with MKS: meters kilograms seconds. So when you input a polygon with width 1 and height 1 its an 1 by 1 meter box. So when you want a 300 by 300 pixels box on your pygame screen and literally input this into Box2D you would end up with a 300 by 300 meter box. This can lead to undesirable behaviour. Box2D is optimized to work best with objects between the sizes of 0.1 to 10 meters

Solution

Most conventionally you use a PPM a pixel per meter constant. Each time you want to use data from box2d for pygame you multiply by the PPM and when you want to input a pygame value into box2d you divide by the PPM.

Axis

Problem

In pygame the highest point on the window is y=0 in box2d y=0 is the bottom of the world. So the y axis is flipped.

Solution

when interacting between pygame and box2d the y axis needs to be flipped.

General Approach:

You write a function that converts from box2d to pygame and from pygame to box2d. Do not mix the usage of box2d and pygame units. Keep them clearly seperated to avoid confusion and keep your code clean.

Point of interaction:

some points where you will likely have to convert between the 2.

- User input
  - User arrow input might be translated into forces applied to box2d bodies
  - You might retrieve the mouse position from pygame and use this to tell box2d where to shoot a bullet
- Drawing
  - When drawing your world you might loop through all the bodies in the box2d world and then convert these into pygame units to blit shapes or sprites onto the screen
- When placing objects in your gameWindow
  - You might use the parameters Window Width and Window Height to know where your placing your box2d bodies.

Sources

Official Website with documentation: https://box2d.org/

https://github.com/pybox2d/pybox2d

https://github.com/erincatto/box2d