Box2D Library Integration (Intro to Box2D)

Before starting...

- ✗ Download and open the Box2D manual. You will use it during the class. (and until the delivery!)
- ✗ Have a look to the Box2D FAQ (Frequently Asked Questions)
- ✗ Most of the types of the library begin with the **b2** prefix

Terminology

Box2D works with rigidbodies in order to simulate movement and collision detection. Simply speaking, a **ridigbody** is a point in space. But each point needs a **shape** (square, rectangle, convex polygon, circle, or whatever). The element that binds rigidbodies and shapes is called a **fixture**, where we will specify mass, density, friction, ... A body can contain multiple fixtures.

The world

The world contains all the bodies, shapes, fixtures and constraints, so we need to **create** one:

```
b2Vec2 gravity(0.0f, -10.0f);
b2World world(gravity);
```

Box2D supports multiple world creation but it won't be necessary for our purposes.

Simulating the world

```
float32 timeStep = 1.0f / 60.0f;
int32 velocityIterations = 8;
int32 positionIterations = 3;
world->Step(timeStep, velocityIterations, positionIterations);
```

We need a **timestep**. Each timestep the physics world will update positions and rotations for all entities, solving the physics equations of its integrator. We need a constant timestep, for example, 60 times per second.

The number and velocity of iterations are needed for constraint solver.

Units

Box2D doesn't deal with pixels!

Internally, Box2D uses *meters*, *kilograms* and *seconds* for movement and *radians* for rotations. So, we have to #define a **macro** that converts pixels to meters and vice versa. So...

How many pixels represents a meter in the simulation?

Hint: Optimal performance values are 0.1 to 10 meters for dynamic bodies. For static bodies, we can reach 50 meters.

Creating bodies

```
b2BodyDef body_def;
body_def.type = b2_staticBody; // or b2_dynamicBody
body_def.position.Set(PIXEL_TO_METERS(x), PIXEL_TO_METERS(y));
b2Body* body= world->CreateBody(&body_def);
```

To create a **body**, we need a **Body Definition**. In that definition, we set up its type (*static, dynamic or kinematic*) and its position (remember to translate pixels to meters)

Creating shapes

```
b2CircleShape shape;
shape.m_radius = PIXEL_TO_METERS(radius);
```

Now, we create a **shape**. For instance, a circle.

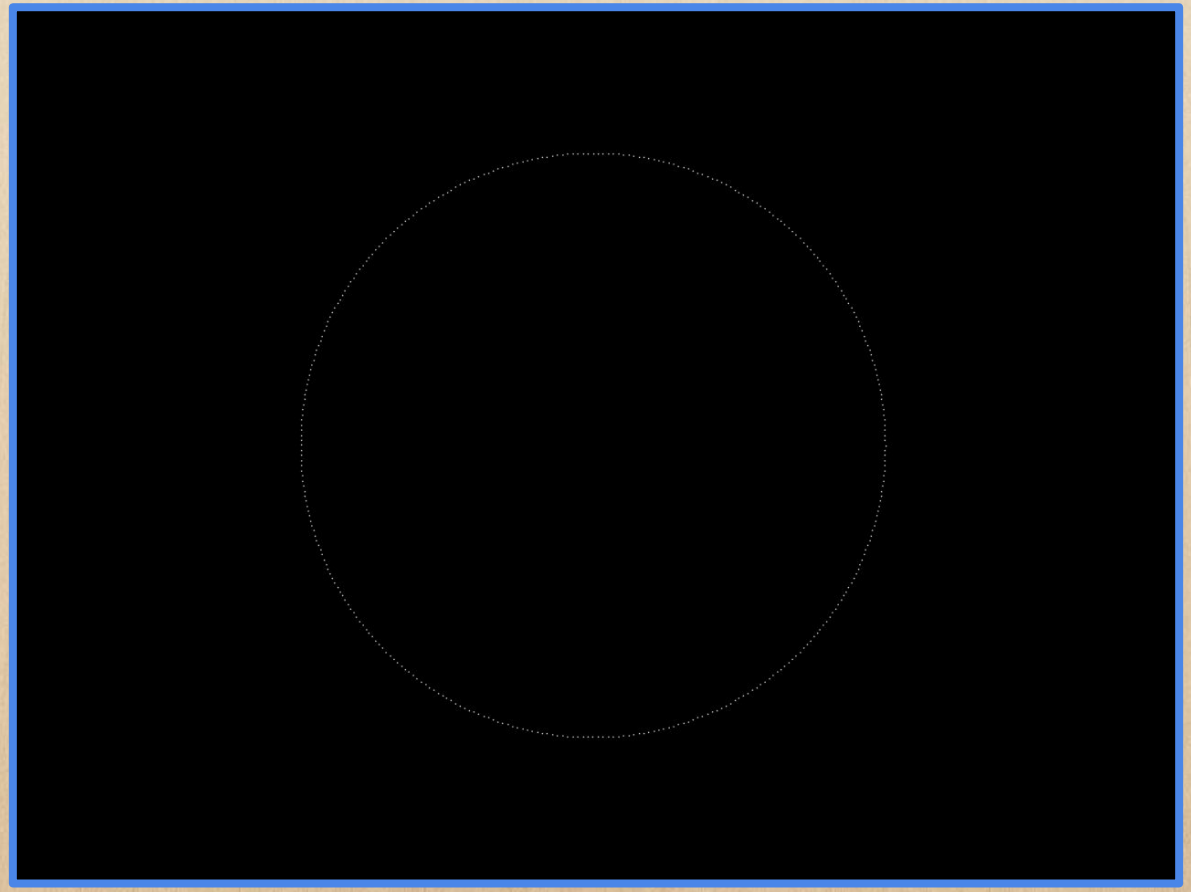
Box2D contains different classes for distinct shapes: b2ChainShape, b2CircleShape, b2EdgeShape, b2PolygonShape, ...

Creating fixtures

```
b2FixtureDef fixture;
fixture.shape = &shape;
body->CreateFixture(&fixture);
```

And finally, we bind **body** and **shape** with a **fixture**. The fixture accepts density and friction parameters.

The *shape* and *fixture* data, and the previous *BodyDef*, are copied by Box2D so they can be discarded.



Our goal

YOUR TURN !

TODO No 1

```
// TODO 1: Include Box 2 header and library
```

- ✗ All the required files are in the Box2D folder
- ✗ You should include the **debug** or the **release** version of the library
- ✗ In order to do that, you must check the _DEBUG macro and use #ifdef statement to include one or the other

TODO No 2

```
// TODO 2: Create a private variable for the world
    // - You need to send it a default gravity
    // - You need init the world in the constructor
    // - Remember to destroy the world after using it
```

- ✗ Check documentation to find it out (Section 2.1)
- ✗ Create the world at *Start* and delete it on *CleanUp*

TODO No 3

```
// TODO 3: Update the simulation ("step" the world)
```

- ✗ Check documentation to find it out (Section 2.4)
- ✗ We will use the values from the previous slide (or you can experiment a bit)

TODO No 4

```
// TODO 4: Create a big static circle as "ground" in the middle of the screen
// - Before, create macros to change from meters to pixels!
```

- ✗ Check documentation to find it out (Section 2.2)
- ✗ We want a BIG static circle located in the middle of the screen. But first, you need to define METER_TO_PIXEL and PIXEL_TO_METER macros.
- ✗ Uncomment the code in PostUpdate to draw all the shapes

TODO No 5

```
// TODO 5: On space bar press, create a circle on mouse position
// - You need to transform the position / radius
```

- ✗ Remember pixel ⇔ meter conversion!
- ✗ **ModuleInput** provides mouse position information

Homework

- ✗ Have you already created the repo?
- ✗ Do you have a partner?
- ✗ Have you chosen your unique pinball?

Next Week . . .

Shapes