

iPhone + Box2D

web: <http://blog.zincroe.com/2009/05/iphone-and-box2d>

twitter: [@lukelutman](#) / [#iphonebox2d](#)

email: luke@zincroe.com

Box2D is an **open source**
2D physics engine written in C++

Box2D is well-documented, has a helpful community and is being actively developed.

Box2D has more features
(like continuous collision detection)
than other 2D physics engines.

Box2D has been ported to many languages, like **Java** and **AS3**.

Box2D is used for the physics in **Rolando**.

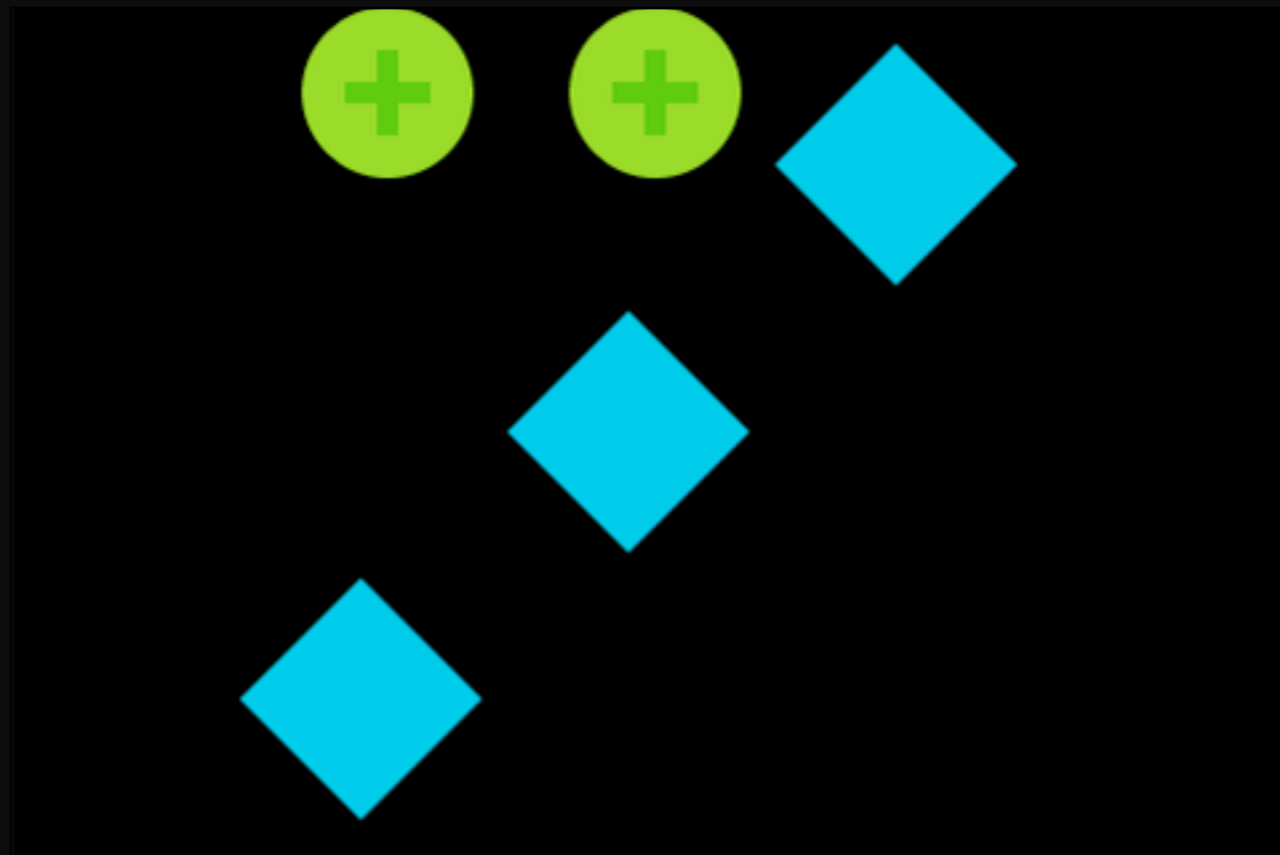
Some important **Box2D** vocabulary...

The **world** defines gravity, moves bodies
around and detects collisions.

Bodies contain shapes, have mass and velocity
and move around the world.

Shapes can be circles or polygons
and are used by the world to detect
collisions between bodies.

When the world **steps**, the bodies in the world are moved around and collisions are checked.



A simple Box2D iPhone game called **MyGame**.

MyGame uses the **View-Based Application** template.

The game begins when `MyGameViewController's viewWillAppear:` method is called.

```
// MyGameViewController.m
```

```
- (void)viewWillAppear:(BOOL)animated {
```

```
    [super viewWillAppear:animated];
```

```
    [self createWorld];
```

```
    [self startTimer];
```

```
    [[UIAccelerometer sharedAccelerometer] setDelegate:self];
```

```
    // other setup code ...
```

```
}
```

MyGameViewController creates the Box2D world.

```
// MyGameViewController.m
```

```
- (void)createWorld {
```

```
    b2AABB aabb;
```

```
    aabb.lowerBound.Set(-100.0f, -100.0f);
```

```
    aabb.upperBound.Set(100.0f, 100.0f);
```

```
    b2Vec2 gravity;
```

```
    gravity.Set(0.0f, 0.0f);
```

```
    [self setWorld:new b2World(aabb, gravity, false)];
```

```
    contactListener = new ContactListener;
```

```
    world->SetContactListener(contactListener);
```

```
}
```

And starts the timer..

```
// MyGameViewController.m
```

```
- (void)startTimer {  
    [self setTimer:  
        [NSTimer scheduledTimerWithTimeInterval:(1.0f/60.0f)  
        target:self  
        selector:@selector(onTimer)  
        userInfo:nil  
        repeats:YES]  
    ];  
}
```

When the timer fires,
MyGameViewController steps the world.

```
// MyGameViewController.m
```

```
- (void)onTimer {  
    [self world]->Step(1.0f/30.0f, 10, 10);  
    // other code ...  
}
```

So far, the game **creates the world**
and **steps the world on a timer.**

But... it's *empty*.

Actors

(aka Game Objects)

Balls, Obstacles and **Walls** are different types of actors.

```
@interface Ball : Actor
    // ball stuff
@end
```

```
@interface Obstacle : Actor
    // obstacle stuff
@end
```

```
@interface Wall : Actor
    // wall stuff
@end
```

Actors are game-specific classes that connect **bodies** and **shapes** in the physics engine to **views** on screen.

```
// Ball.m
```

```
- (id)init {  
    if(self = [super init]) {  
  
        [self setBodyDef:new b2BodyDef];  
        [self setShapeDef:new b2CircleDef];  
        [self shapeDef]->density = 2.0f;  
        [self shapeDef]->restitution = 0.75f;  
        [self setRadius:32.0f];  
  
        [self setBallView:  
            //code to create image view  
        ];  
  
    }  
    return self;  
}
```

The game keeps a **list of actors**.

```
// MyGameViewController.h
```

```
@property (nonatomic, retain) NSMutableSet *actorSet;
```

```
- (void)addActor:(Actor *)anActor;
```

```
- (void)removeActor:(Actor *)anActor;
```

```
- (void)removeAllActors;
```

When the game starts, it adds some actors...

```
// MyGameViewController.m
```

```
- (void)viewWillAppear:(BOOL)animated {
```

```
    // code to create world, start timer, etc.
```

```
    Ball *aBall = [[[Ball alloc] init] autorelease];
```

```
    [aBall setPosition:CGPointMake(160.0f, 64.0f)];
```

```
    [self addActor:aBall];
```

```
    // code to add more actors...
```

```
}
```

When an Actor is added to the game,
it's `actorDidAppear` method is called.

```
// Ball.m
```

```
- (void)actorDidAppear {
```

```
    [super actorDidAppear];
```

```
    [self setBody:[game world]->CreateBody([self bodyDef])];
```

```
    [self body]->CreateShape([self shapeDef]);
```

```
    [self body]->SetUserData((void *)self);
```

```
    [self body]->SetMassFromShapes();
```

```
    [[game view] addSubview:[self ballView]];
```

```
}
```

Every time the game steps the world, it **notifies the actors** so they can update their views.

```
// Ball.m

- (void)worldDidStep {

    [super worldDidStep];

    CGAffineTransform aTransform;

    // code to convert the position and rotation
    // of the ball's body to a CGAffineTransform...

    [[self ballView] setTransform:aTransform];

}
```

Now the world has things in it, and the things move.

But... the things don't know about each other.

Contacts

When we created the world, we set a **contact listener**.

```
// MyGameViewController.m
```

```
- (void)createWorld {
```

```
    b2AABB aabb;
```

```
    aabb.lowerBound.Set(-100.0f, -100.0f);
```

```
    aabb.upperBound.Set(100.0f, 100.0f);
```

```
    b2Vec2 gravity;
```

```
    gravity.Set(0.0f, 0.0f);
```

```
    [self setWorld:new b2World(aabb, gravity, false)];
```

```
    contactListener = new ContactListener;
```

```
    world->SetContactListener(contactListener);
```

```
}
```

When the world steps, the contact listener is notified about collisions between shapes.

The contact listener maps the shapes to actors and notifies the actors about the collision.

```
// ContactListener.m
```

```
void ContactListener::Add(const b2ContactPoint *point) {  
    id actor1 = (id)point->shape1->GetBody()->GetUserData();  
    id actor2 = (id)point->shape2->GetBody()->GetUserData();  
    if(actor1 && actor2) {  
        [actor1 addContact:actor2];  
        [actor2 addContact:actor1];  
    }  
}  
  
void ContactListener::Remove(const b2ContactPoint *point) {  
    id actor1 = (id)point->shape1->GetBody()->GetUserData();  
    id actor2 = (id)point->shape2->GetBody()->GetUserData();  
    if(actor1 && actor2) {  
        [actor1 removeContact:actor2];  
        [actor2 removeContact:actor1];  
    }  
}
```

Each actor keeps a **list of contacts**.

```
// Actor.h
```

```
@property (nonatomic, readonly) NSSet *contactSet;
```

```
- (void)addContact:(Actor *)aContact;
```

```
- (void)removeContact:(Actor *)aContact;
```

Actors use the list of contacts to **respond to collisions**.

```
// Obstacle.m
```

```
- (void)worldDidStep {
```

```
    // code to update the position and rotation of  
    // the normal and highlighted views...
```

```
    if([[self contactSet] count]) {  
        [[self normalView] removeFromSuperview];  
        [[[self game] view] addSubview:[self highlightedView]];  
    } else {  
        [[self highlightedView] removeFromSuperview];  
        [[[self game] view] addSubview:[self normalView]];  
    }  
}
```

Finally, the world has things, the things move
and the things know about each other.

Where to go from here...

MyGameViewController **doesn't know how any of the actors work**, just that they can be **added, notified and removed**.

Adding **new behavior** is as easy as implementing **new Actor subclasses**.

Actors don't have to have
one body and **one view**.

Actors can **add or remove other actors**.

The physics and game code are distinct from the display code, making it relatively easy to switch from UIKit to OpenGL.

iPhone + Box2D

web: <http://blog.zincroe.com/2009/05/iphone-and-box2d>

twitter: [@lukelutman](#) / [#iphonebox2d](#)

email: luke@zincroe.com