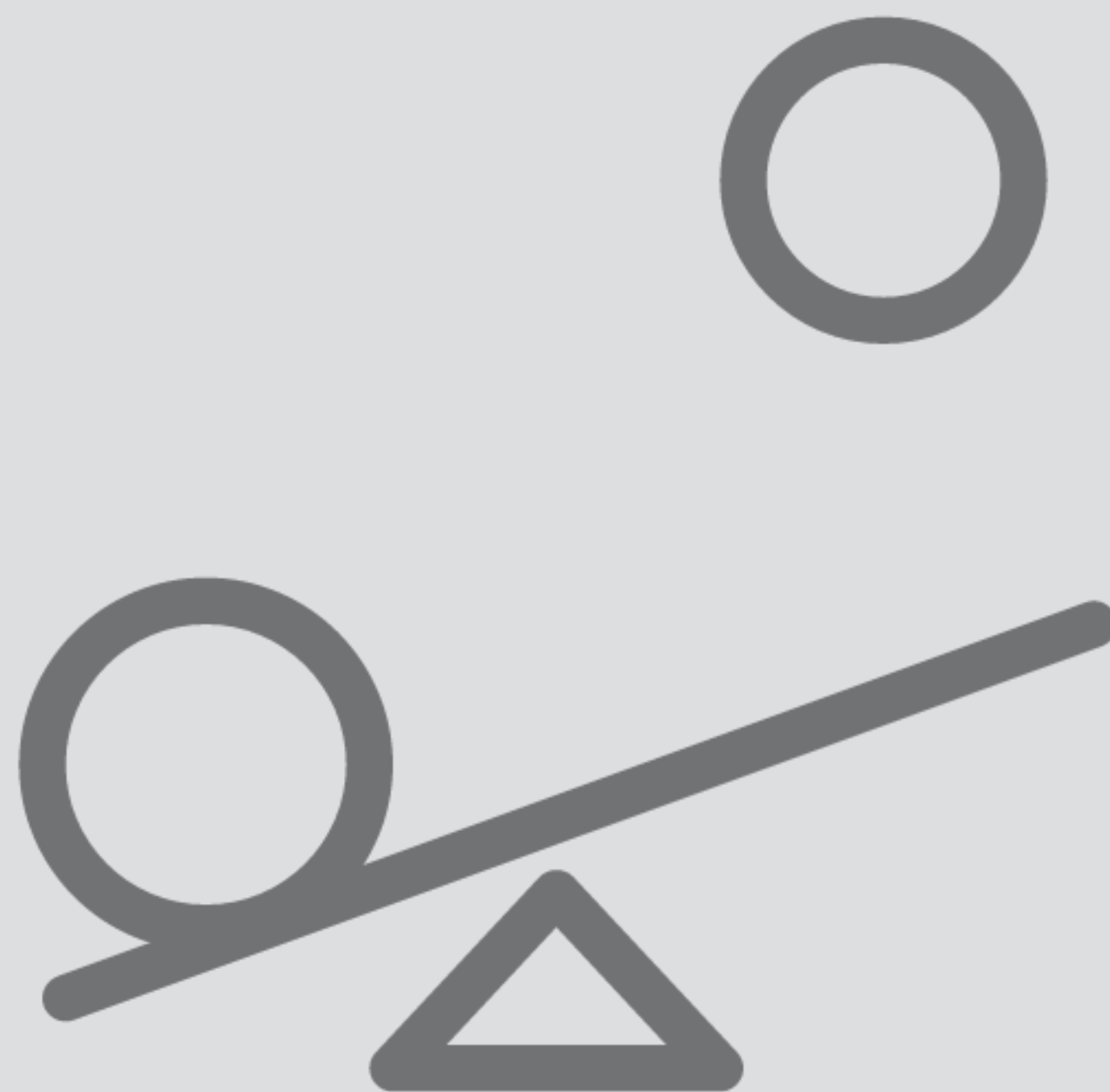
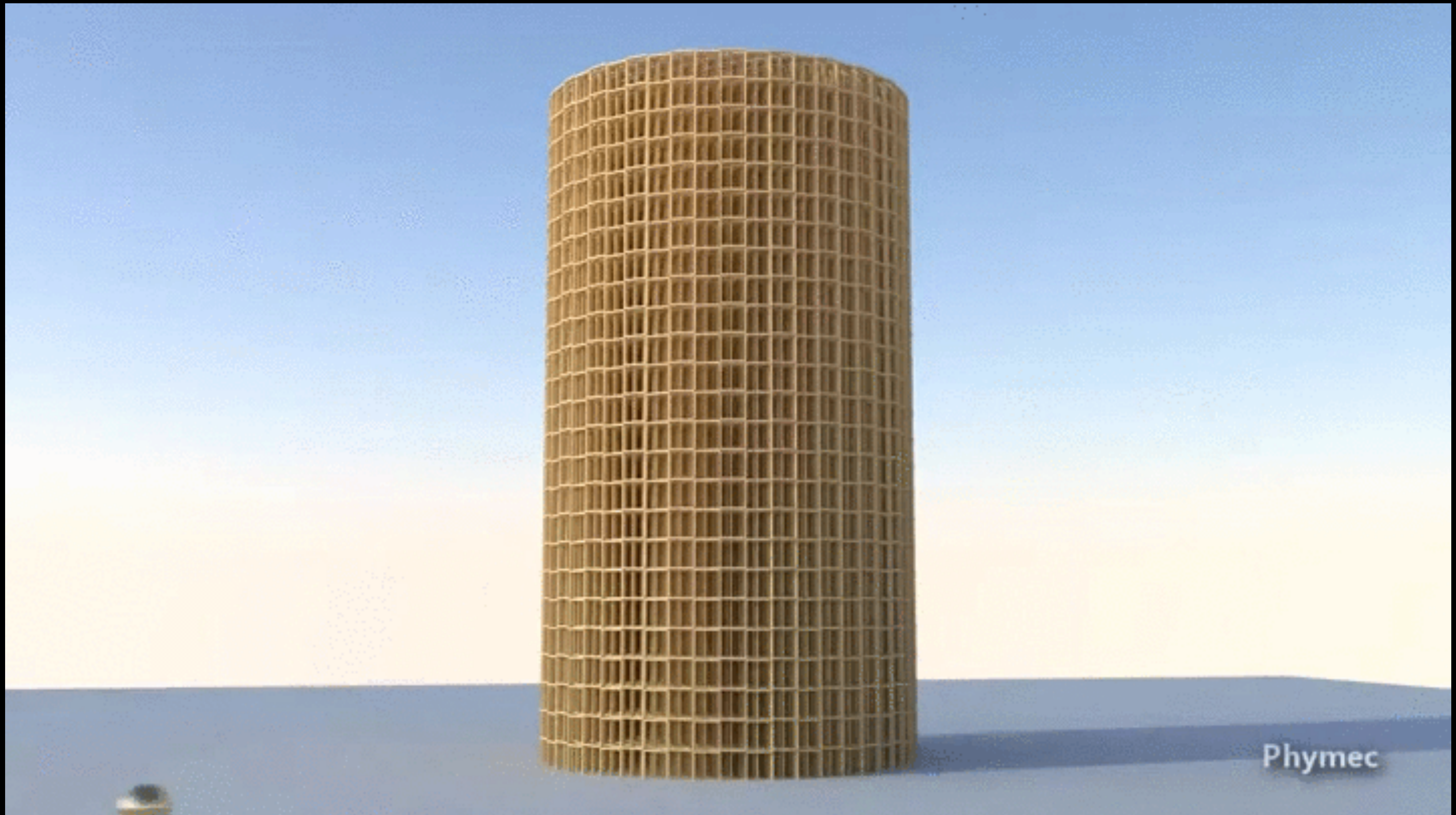


Physics engines.

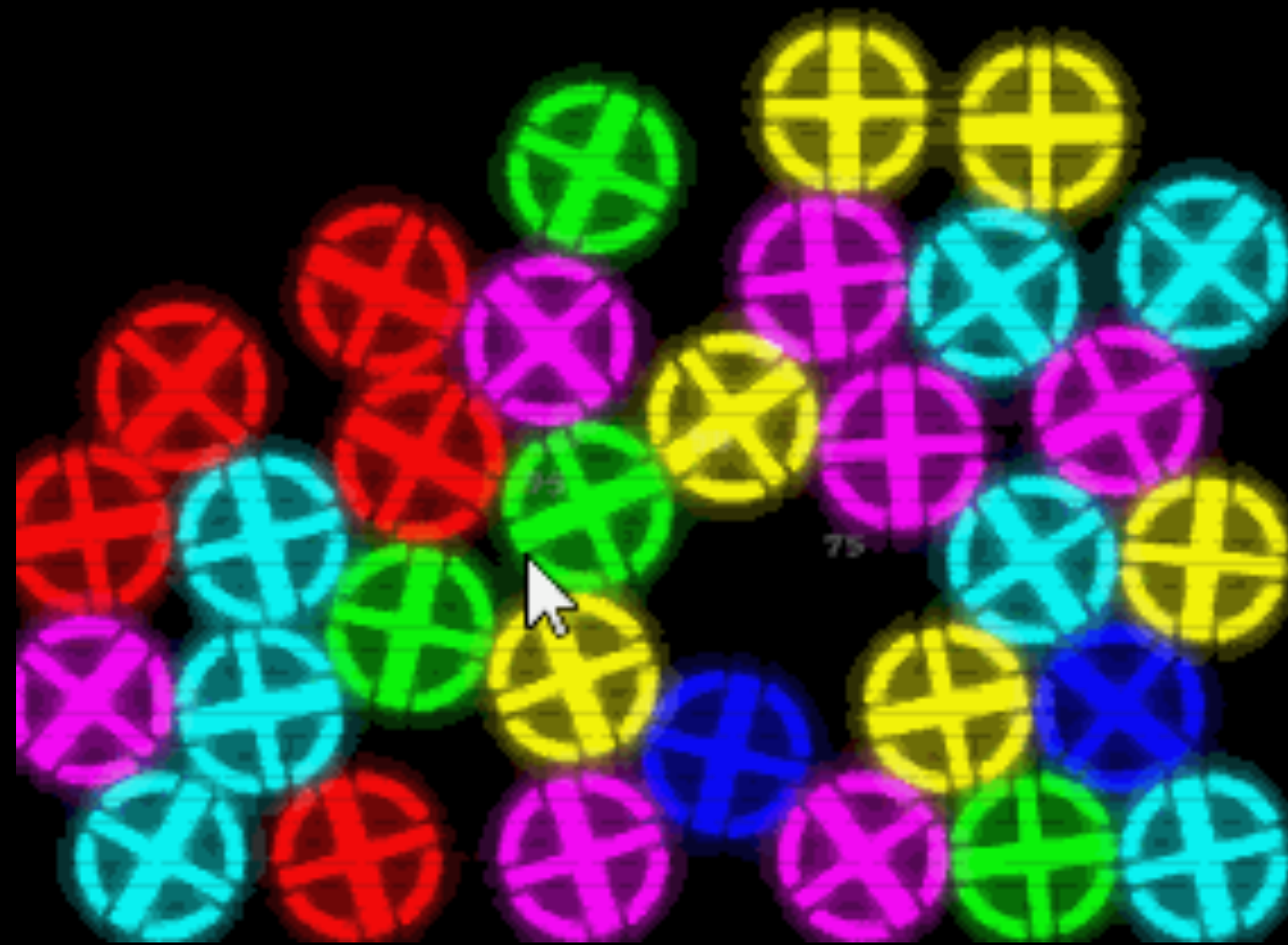


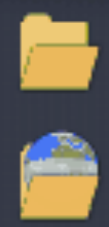






Score: 6400
Level: 5



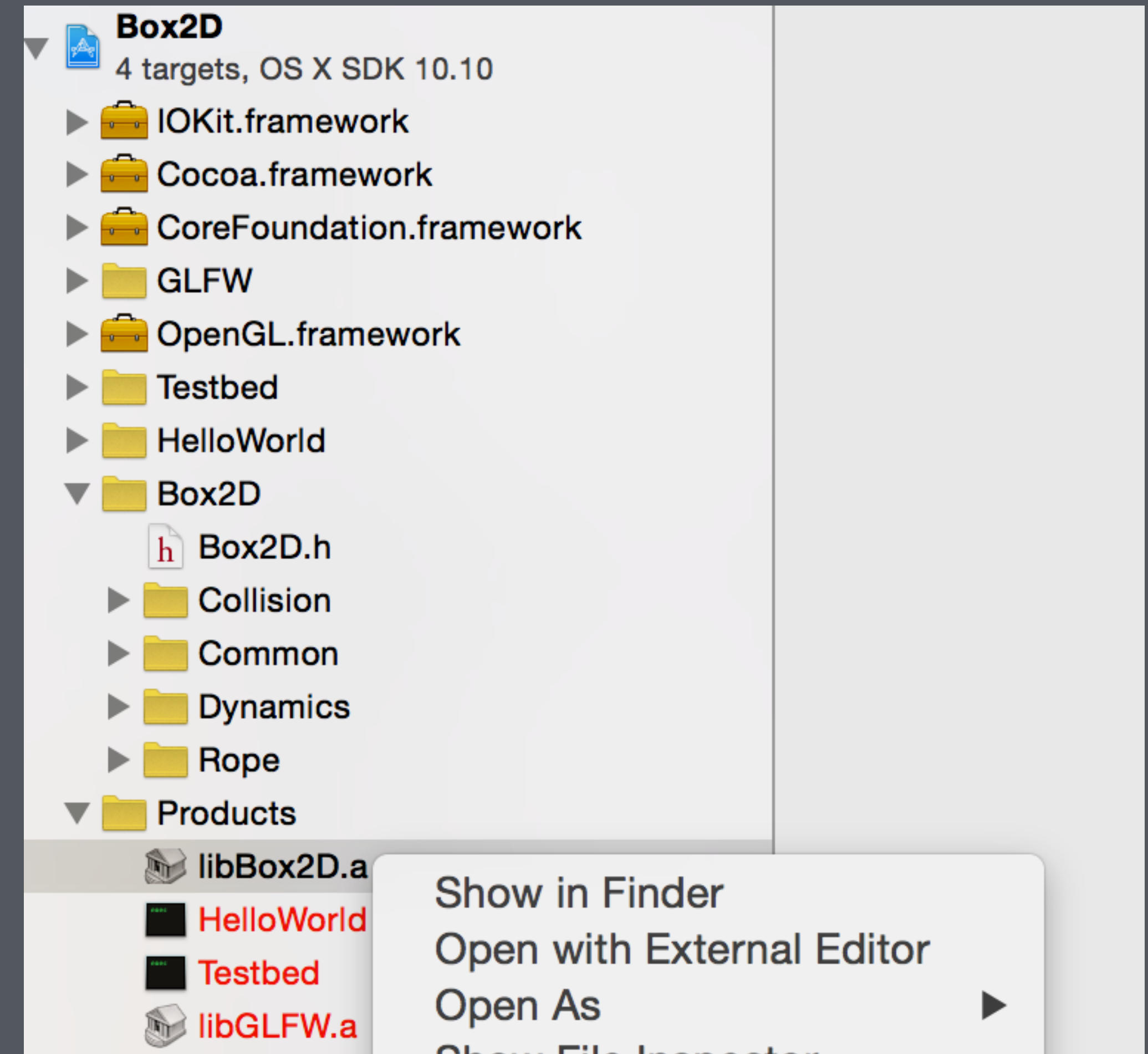
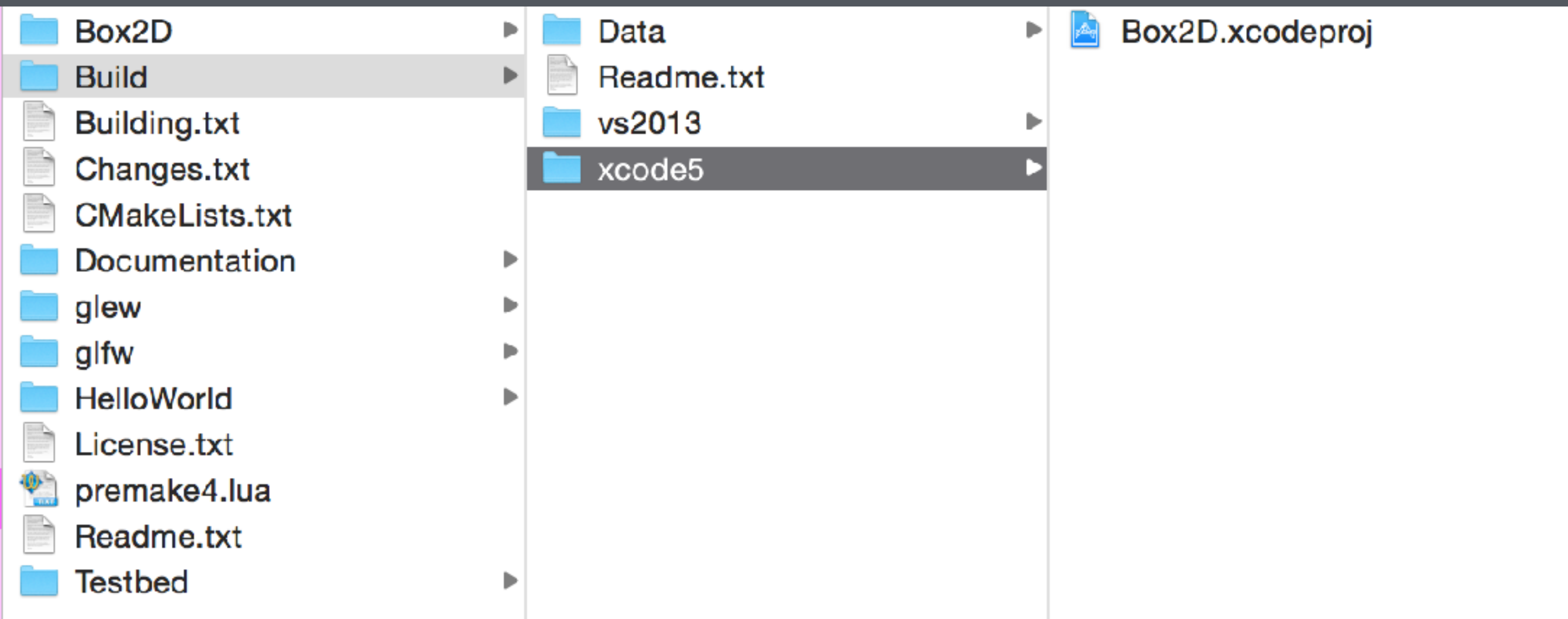


Box2D

<http://box2d.org/>

<https://github.com/erincatto/Box2D/releases>

Building Box2D



Using Box2D

**Set the Box2D folder as an
additional include path in your project**

**Set the path where you copied the
Box2D library as an additional library path in your project.**

Link libBox2D.a or libBox2D.lib (if on Windows)

Creating a Box2D world.

Include the header

```
#include "Box2D/Box2D.h"
```

Create the Box2D world.

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

```
b2World *physicsWorld = new b2World(gravity);
```

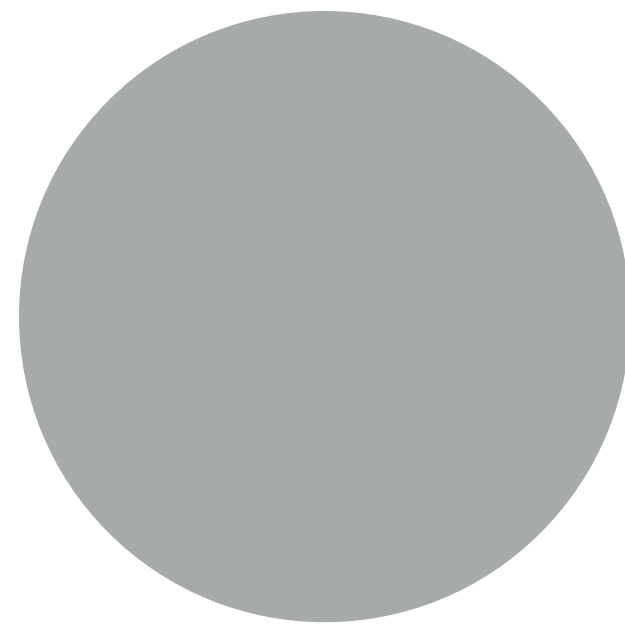

Step the world using elapsed time.

```
void Update(float elapsed) {  
    physicsWorld->Step(elapsed, 10, 10);  
}
```

On a fixed timestep!

Creating a **physics** object.

Step 1: Create a **shape**.



```
b2CircleShape shape;  
shape.m_radius = 0.3f;
```



```
b2PolygonShape shape;  
shape.SetAsBox(0.3, 0.3);
```

Step2: Create a fixture definition.

```
b2FixtureDef fDef;  
fDef.friction = 1.0f;  
fDef.restitution = 0.2f;  
fDef.density = 10.0f;  
fDef.filter.groupIndex = 0;
```

... and set our shape to it.

```
fDef.shape = &shape;
```


Step 3: Create a body definition and set its type and position.

```
b2BodyDef bodyDef;  
bodyDef.position.x = 0.2;  
bodyDef.position.y = -0.5;  
bodyDef.type = b2_dynamicBody;
```

..or

```
bodyDef.type = b2_staticBody;
```

for static entities.

Now create it!

```
b2Body *body = physicsWorld->CreateBody(&bodyDef);  
b2Fixture *fixture = body->CreateFixture(&fDef);
```


Putting it all together:

```
b2CircleShape shape;  
shape.m_radius = 0.3f;
```

```
b2FixtureDef fDef;  
fDef.friction = 1.0f;  
fDef.restitution = 0.2f;  
fDef.density = 10.0f;  
fDef.filter.groupIndex = 0;  
fDef.shape = &shape;
```

```
b2BodyDef bodyDef;  
bodyDef.type = b2_dynamicBody;
```

```
b2Body *body = physicsWorld->CreateBody(&bodyDef);  
b2Fixture *fixture = body->CreateFixture(&fDef);
```

Now what?

Connect the simulated
Box2D body with an entity.


```
class Entity {
public:

    Entity();

    void Update(float elapsed);
    void Render(ShaderProgram &program);

    SheetSprite sprite;

    float x;
    float y;
    float rotation;

    b2Body *body;
    b2Fixture *fixture;

};
```

```
void Entity::Update(float elapsed) {
    if(body) {
        b2Vec2 position = body->GetPosition();
        x = position.x;
        y = position.y;
        rotation = body->GetAngle();
    }
}
```



```
Entity platform;  
platform.sprite = SheetSprite(tilesTexture, 648.0/1024.0,  
0.0/1024.0, 70.0/1024.0, 70.0/1024.0, 0.8);
```

```
b2PolygonShape shape;  
shape.SetAsBox(0.4, 0.4);
```

```
b2FixtureDef fDef;
```

```
b2BodyDef bodyDef;  
bodyDef.position.y = -0.5;  
bodyDef.position.x = 0.0;  
bodyDef.type = b2_staticBody;
```

```
platform.body = physicsWorld->CreateBody(&bodyDef);  
platform.fixture = platform.body->CreateFixture(&fDef);
```

```
entities.push_back(platform);
```

**Make sure the size of your entity is the
same as your box2d shape!**

Character physics using Box2D.

Fixed rotation!

```
bodyDef.fixedRotation = true;
```

Setting velocity.

```
body->SetLinearVelocity(b2Vec2(3.0, 0.0));
```


Detecting collisions.

Need to make a
subclass of b2ContactListener and add a
BeginContact method to it:

```
class ClassDemoApp : public b2ContactListener {  
    public:  
  
        void BeginContact (b2Contact *contact);  
};
```

Now we can pass our class in as a
contact listener.

```
physicsWorld->SetContactListener(this);
```

Now our BeginContact method will be called every time there is a new contact!

```
void ClassDemoApp::BeginContact (b2Contact *contact) {  
    b2Fixture *fixtureA = contact->GetFixtureA();  
    b2Fixture *fixtureB = contact->GetFixtureB();  
}
```


We can look at the fixtures in the contact and compare to fixtures in our entities to check what is colliding.

```
void ClassDemoApp::BeginContact (b2Contact *contact) {  
  
    b2Fixture *fixtureA = contact->GetFixtureA();  
    b2Fixture *fixtureB = contact->GetFixtureB();  
  
    for(int i=0; i < entities.size(); i++) {  
        if(fixtureA == player->fixture && fixtureB == entities[i].fixture  
            && entities[i].type == TYPE_COIN) {  
  
            // player touching a coin  
  
        }  
    }  
}
```

We have to be careful because we cannot remove objects in the callback. If we want to remove, we must add them to a list and remove after we do our Step.

```
void ClassDemoApp::BeginContact (b2Contact *contact) {  
  
    b2Fixture *fixtureA = contact->GetFixtureA();  
    b2Fixture *fixtureB = contact->GetFixtureB();  
  
    for(int i=0; i < entities.size(); i++) {  
        if(fixtureA == entities[0].fixture && fixtureB == entities[i].fixture &&  
entities[i].type == TYPE_COIN) {  
            coinsToRemove.push_back(&entities[i]);  
        }  
    }  
}
```

We have to be careful because we cannot remove objects in the callback. If we want to remove, we must add them to a list and remove after we do our Step.

```
physicsWorld->Step(elapsed, 10,10);

for(int i=0; i < coinsToRemove.size(); i++) {

    physicsWorld->DestroyBody(coinsToRemove[i]->body);
    coinsToRemove[i]->body = NULL;
    coinsToRemove[i]->visible = false;

}
coinsToRemove.clear();
```

Bullets and sensors.

You can make an object into a sensor that tracks collisions but doesn't create responses.

```
b2FixtureDef fDef;  
fDef.isSensor = true;
```

```
b2BodyDef bodyDef;  
bodyDef.type = b2_dynamicBody;  
bodyDef.gravityScale = 0.0;
```

By setting its gravity scale to 0, you can make it unaffected by gravity and just set its velocity directly.

Thank you!