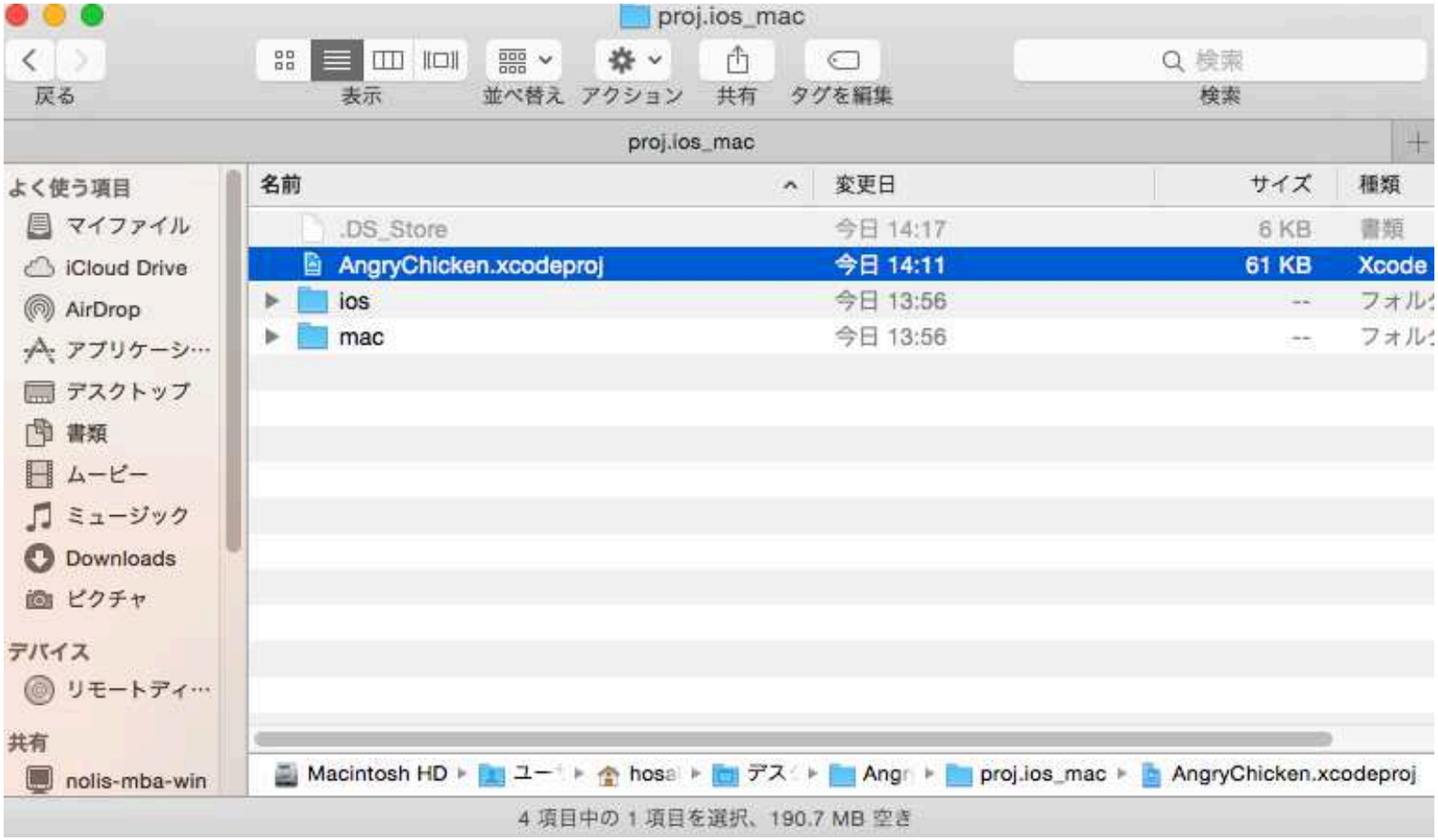
Cocos2d-xで作る物理演算ゲーム
～　chipmunk編　～

＝　2015年08月22日　＝

前回のあらすじ

cocos2d-xの開発環境を構築し、
初期プロジェクトの実行まで行いました。

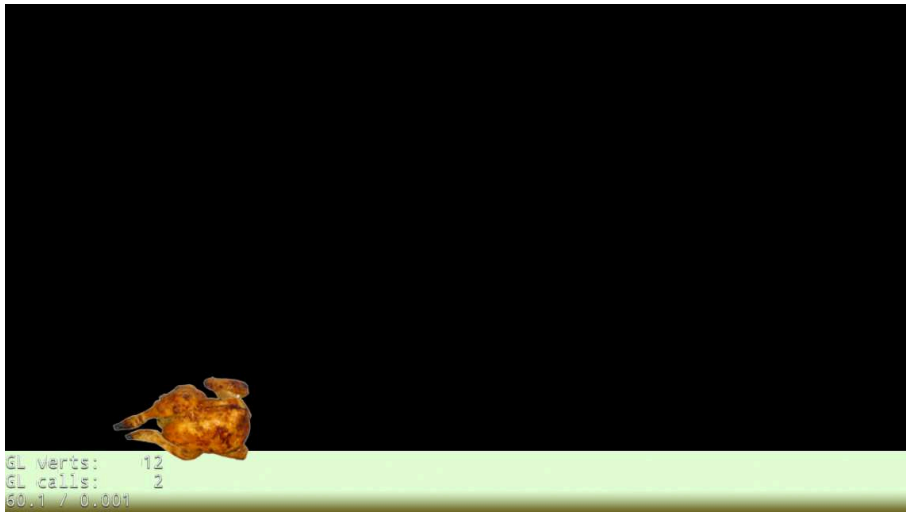
AngryChickenをxcodeで実行してみる。




Hello World
COCOS2D-X
GL verts: 78
GL calls: 3
59.3 / 0.002

今回は物理演算Chipmunkを
使って「モノが落下して、地面に
落ちる」までやる。

これ



画像の準備

チキン：１７２ｘ１７２
chicken.png

地面：1136 x 70
ground.png

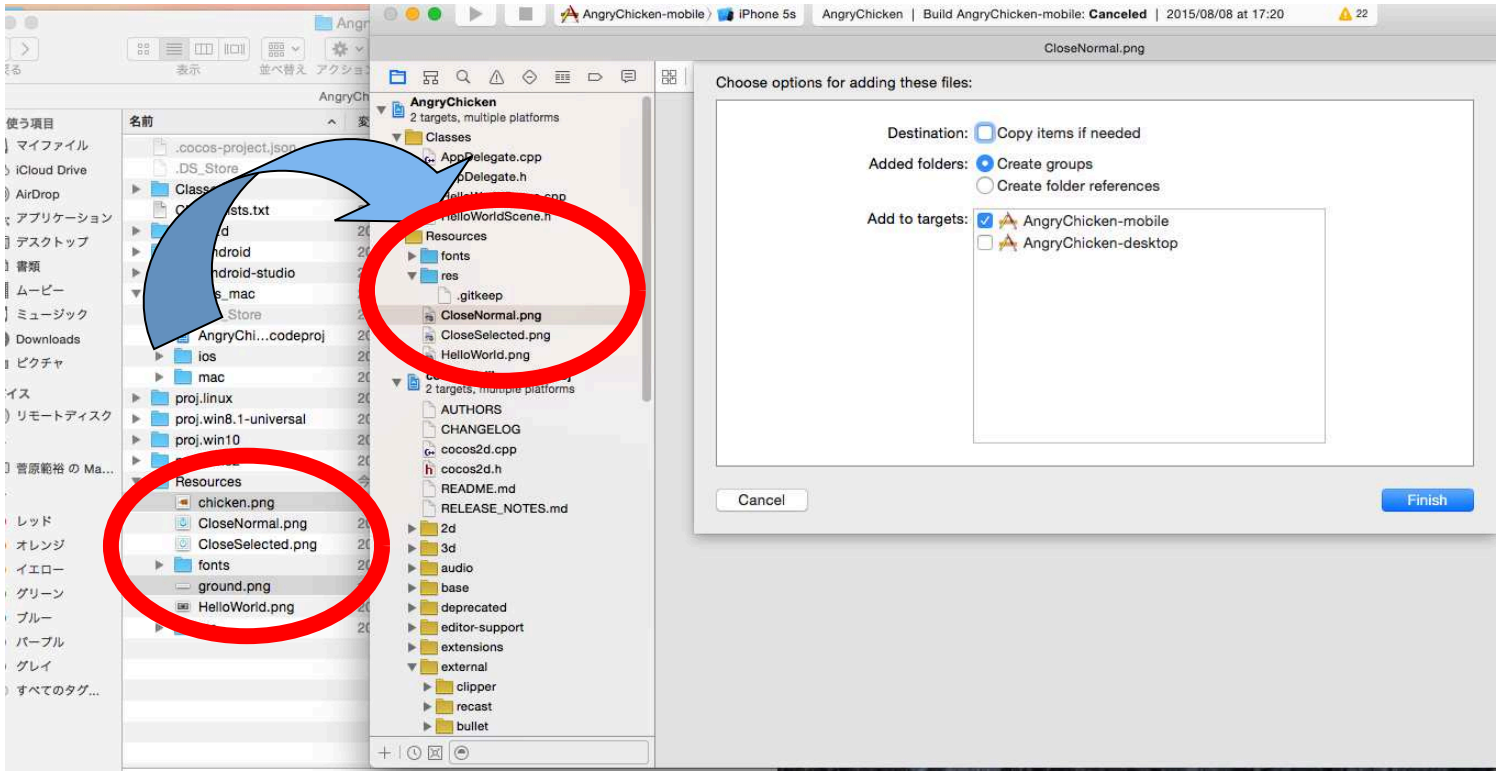
画像をプロジェクトに入れる

Resourceフォルダにぶち込もう

Resources	今日 19:00	--	フォルダ
chicken.png	昨日 18:18	32 KB	PNG イメージ
CloseNormal.png	2015年8月8日 13:08	4 KB	PNG イメージ
CloseSelected.png	2015年8月8日 13:08	3 KB	PNG イメージ
fonts	2015年8月8日 13:08	--	フォルダ
ground.png	今日 18:58	153 KB	PNG イメージ
HelloWorld.png	2015年8月8日 13:08	138 KB	PNG イメージ
res	2015年8月8日 13:08	--	フォルダ

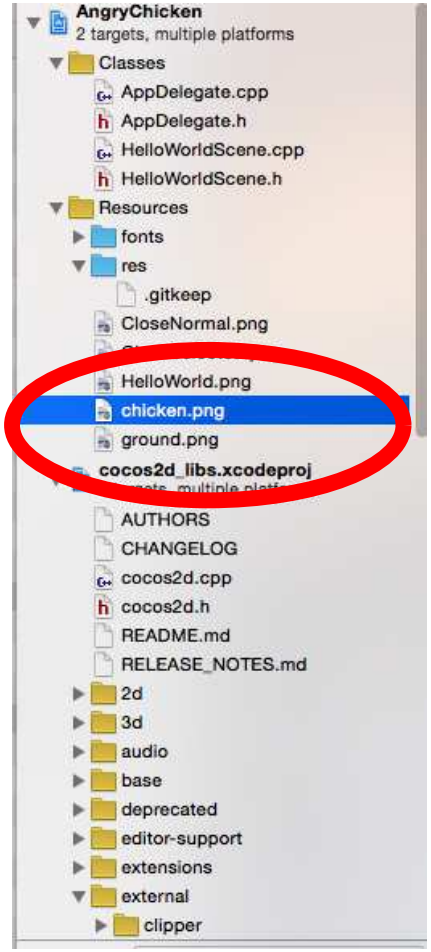
xcodeに画像を登録

フォルダからXcodeのプロジェクトナビゲーターのResource配下に
ドラッグアンドドロップ



管理方法を問われるので画像のような状態にしてＦＩＮＩＳＨ

xcodeに追加された状態



いよいよソースをイジイジ

まずは全体に影響ある部分から

AppDelegate.cppを修正

```
USING_NS_CC;

static cocos2d::Size designResolutionSize = cocos2d::Size(480, 320);
static cocos2d::Size smallResolutionSize = cocos2d::Size(480, 320);
static cocos2d::Size mediumResolutionSize = cocos2d::Size(1024, 768);
static cocos2d::Size largeResolutionSize = cocos2d::Size(2048, 1536);
```

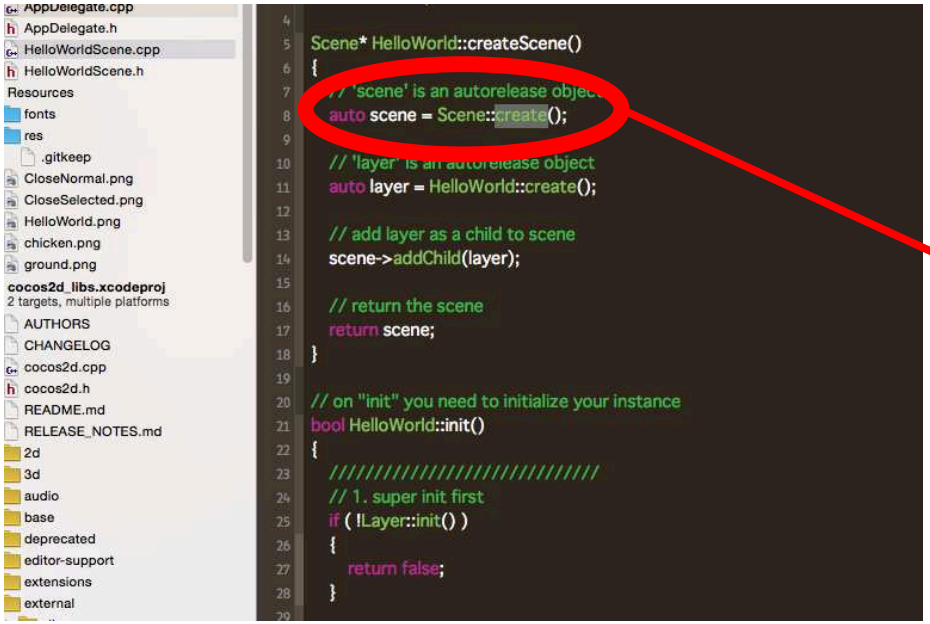
```
USING_NS_CC;

static cocos2d::Size designResolutionSize = cocos2d::Size(1136, 640);
static cocos2d::Size smallResolutionSize = cocos2d::Size(480, 320);
static cocos2d::Size mediumResolutionSize = cocos2d::Size(1024, 768);
static cocos2d::Size largeResolutionSize = cocos2d::Size(2048, 1536);

AppDelegate::AppDelegate() {
```

＊これは何をやってるの？
ゲームには当然画像が必要になる。そうなると画像を作成するにあたり、どの解像度を基準にして画像を作るのか？ということになる。その解像度だ。今回は私の端末iPhone5s(1136x640)で、画像を作るので、上記のように設定する。恐らく、cocos2d-xではこのパラメータと実際の解像度の比率を求めて画像をスケーリングしているのであろう。

次はシーン生成のクラスを変更



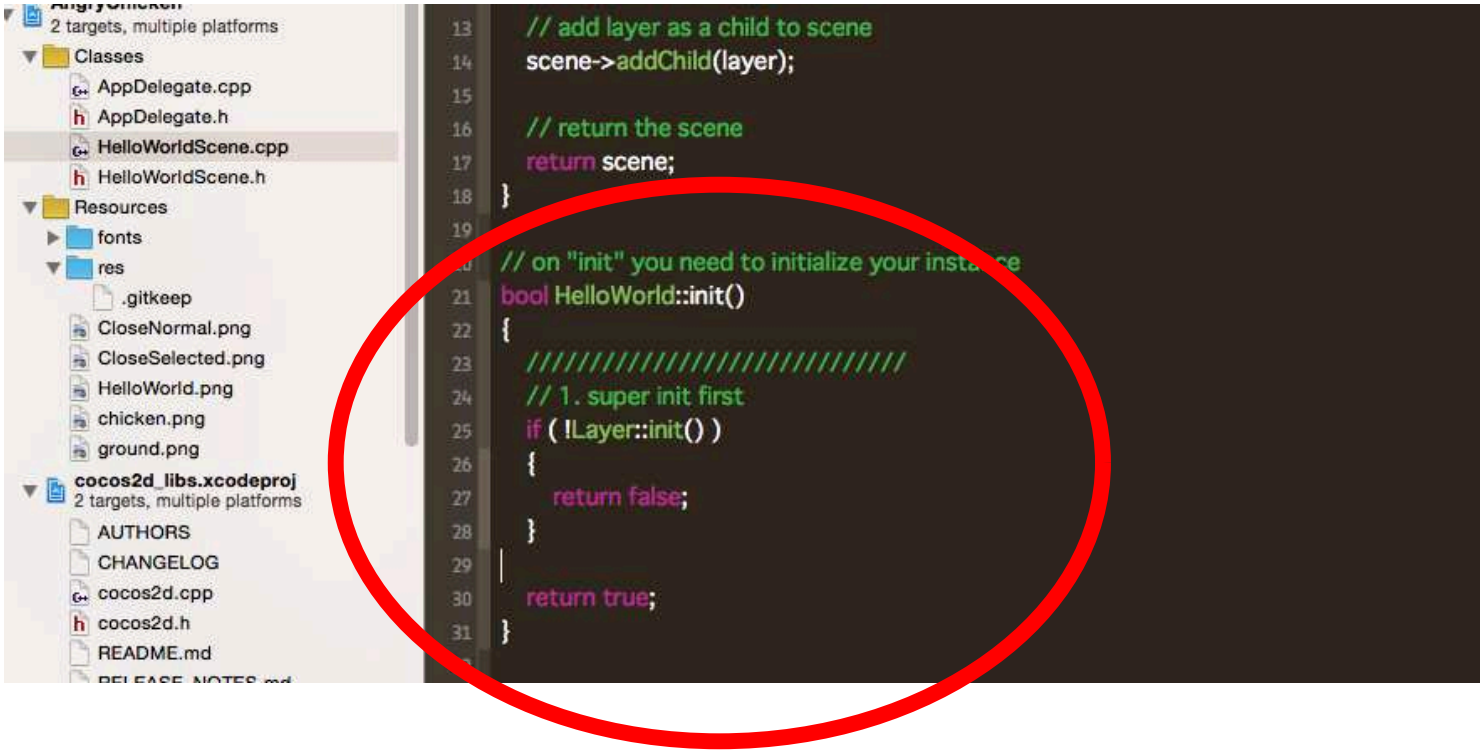

```
auto scene =
    Scene::createWithPhysics();
```

上記に変更

＊シーンとは？
cocos2d-xはゲーム中の機能的な１画面をシーンという単位で管理する。そしてそのためのクラスが存在する。代表例としては「メニュー、タイトル、メインゲーム中」をそれぞれシーンとして管理するのが一般的。

シーンの初期化処理の変更

HelloWorldScene.cppのinitメソッドの中身を上記のように変更



画像読み込みなど追記

画像の読み込みと画像に物理的パラメータを設定するコードを追加する。
createメソッドの return scene;のまえに以下のコードを追加します。

```
Size winSize = Director::getInstance()->getWinSize();

// 重力を設定
PhysicsWorld* world = scene->getPhysicsWorld();
Vect gravity;
gravity.setPoint(0, -150);
world->setGravity(gravity);

// 地面
Sprite* floor = Sprite::create("sample_floor.png");
floor->setPosition(Point(winSize.width / 2 , floor->getContentSize().height / 2));
PhysicsBody* floorPb = PhysicsBody::createBox(floor->getContentSize());
floorPb->setDynamic(false);
floor->setPhysicsBody(floorPb);
layer->addChild(floor);

// キャラクター
Sprite* character = Sprite::create("sample_chara01.png");
character->setPosition(Point(winSize.width / 5 , winSize.height / 2));
PhysicsBody* charaPb = PhysicsBody::createCircle(40);
charaPb->setMass(1.0f); // 重さを指定（ここが無いと後で飛ばせなくなる）
character->setPhysicsBody(charaPb);
character->setTag(1);
layer->addChild(character);
```

ソースの説明①

```
Size winSize = Director::getInstance()->getWinSize();
```

描画領域サイズの取得＝端末の解像度

```
PhysicsWorld* world = scene->getPhysicsWorld();
Vect gravity;
gravity.setPoint(0, -150);
world->setGravity(gravity);
```

重力の設定。つまり下方向に150の力がこの世界では働くことになる。

ソースの説明②

```
Sprite* floor = Sprite::create("ground.png");
floor->setPosition(Point(winSize.width / 2 , floor->getContentSize().height / 2));
PhysicsBody* floorPb = PhysicsBody::createBox(floor->getContentSize());
floorPb->setDynamic(false);
floor->setPhysicsBody(floorPb);
layer->addChild(floor);
```

地面の画像を生成する。その後、物理演算オブジェクトを生成して、画像と紐づける。ここで重要なのは、地面は重力の影響を受けない。こと。つまり「静止した状態の、物理的な性質をもった物体」ということなのでsetDynamic(false)となる。

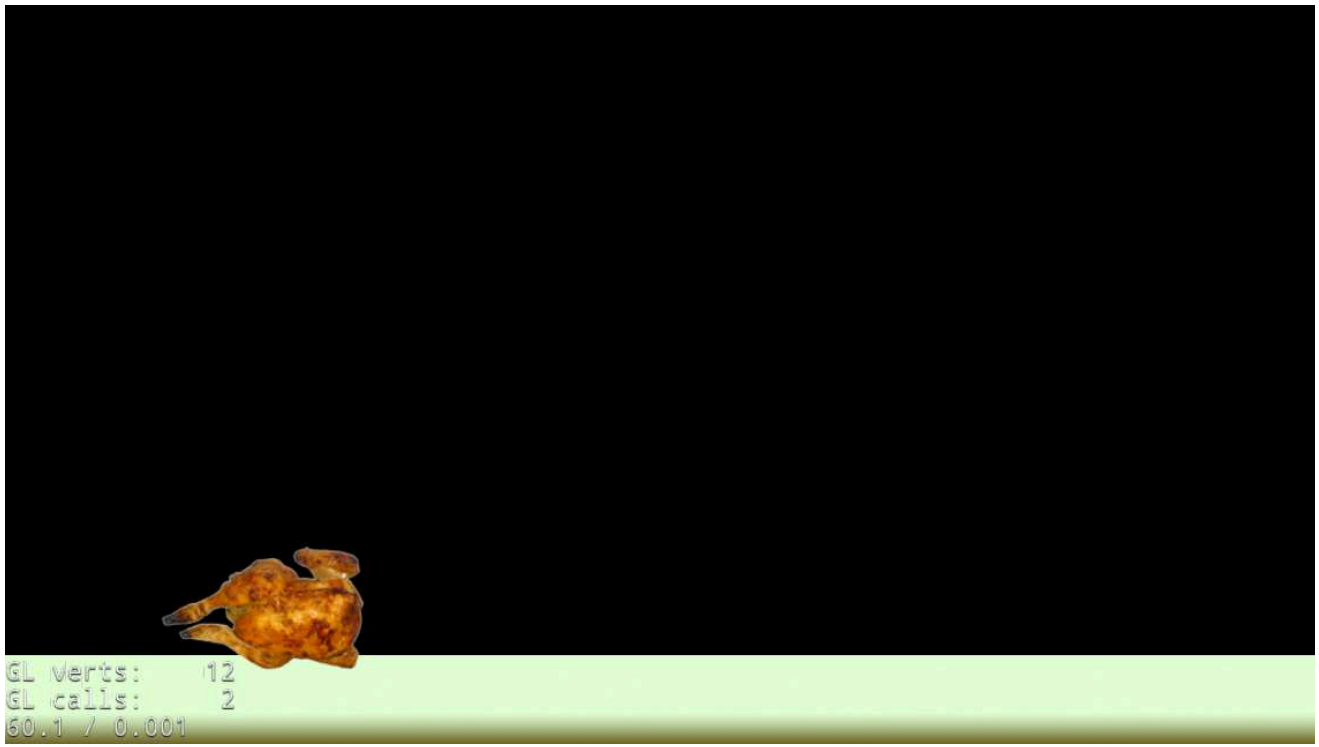
ソースの説明③

```
// キャラクター
Sprite* character = Sprite::create("chicken.png");
character->setPosition(Point(winSize.width / 5 , winSize.height / 2));
PhysicsBody* charaPb = PhysicsBody::createCircle(40);
charaPb->setMass(1.0f); // 重さを指定（ここが無いと後で飛ばせなくなる）
character->setPhysicsBody(charaPb);
character->setTag(1);
layer->addChild(character);
```

チキン画像を生成して、物理演算オブジェクトを生成する。
物理演算オブジェクトは円としている。

実行してみよう

- 最初に指示したような状態になっているはず



次回は物理演算Chipmunk
ひっぱって飛ばす編

ご清聴ありがとうございました。