Cocos2d-xで作る物理演算ゲーム

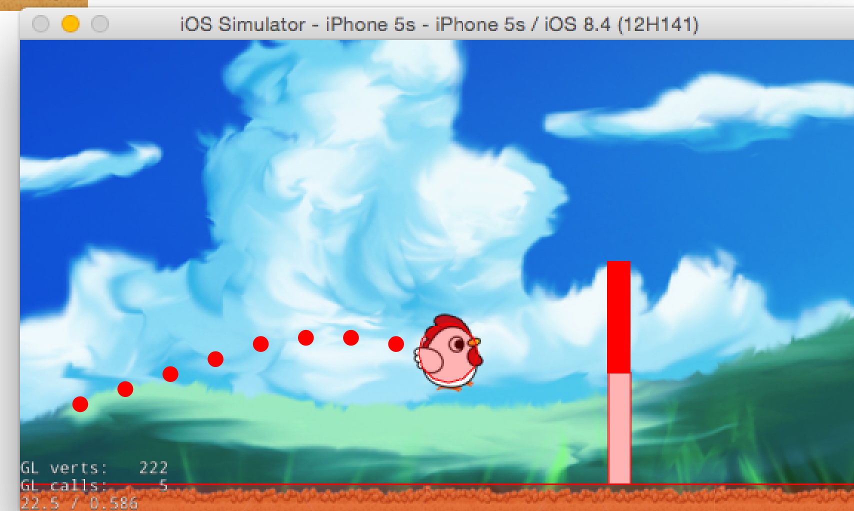
軌跡の点線を入れてみる②編

＝ 2016年4月30日 ＝

前回のあらすじ

チキンを引っ張り・・・

飛ばすと・・・軌跡が！！！



んが！！？？

ゲームが終わるまで点がずっと追加されてしまう。これを今回はなおします。

ちょっと実行して現状確認してみましょう。

ここまでのソースプログラムはここ

http://monolizm.com/sab/src/AngryChicken17.zip

GETだぜ！

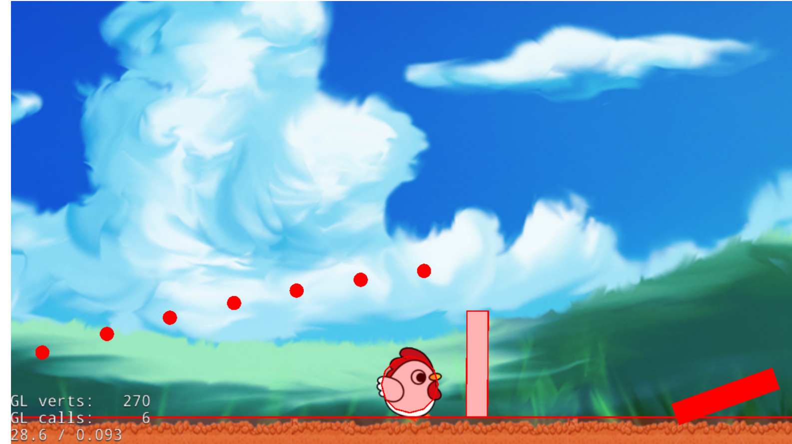
今回は軌跡の点線を
いれてみる②。点線をある条件
を満たしたら点線追加を中止する

チキンを引っ張り・・・

飛ばすと・・・軌跡が！！！

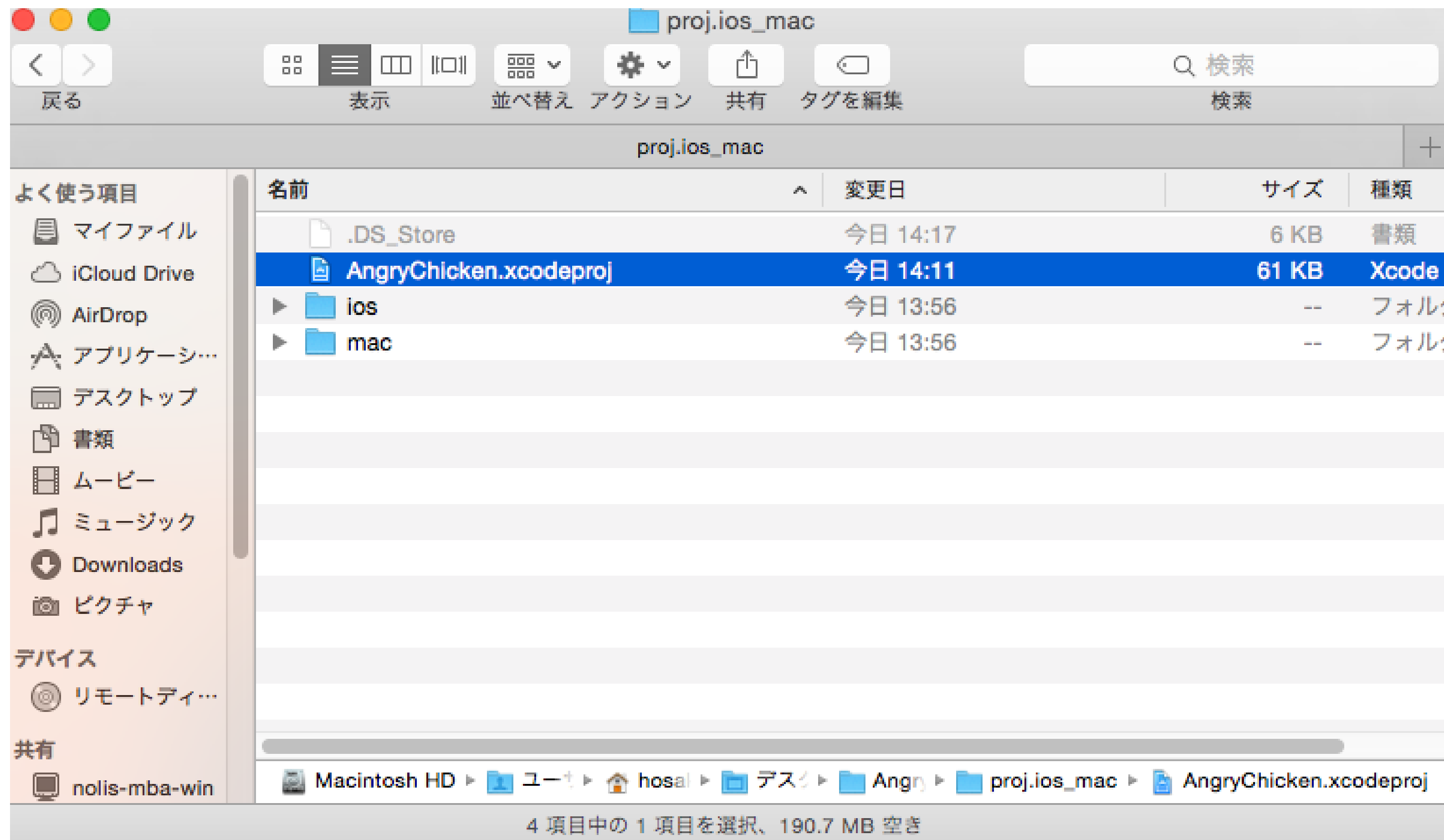
コレ

ぶつかった後は点線でない！



まずは起動しよう

AngryChickenをxcodeで実行。



まずは必要な処理を考えてみる

①発射後に「鶏が何かに衝突したら」を判定
②「真」なら点線追加をやめる

①発射後何かと衝突したら判定

根本の衝突判定は「第12回　～壊す対象を置いて当たり判定をとろう編～)」ですでにやっている。物体と物体が衝突したら以下のメソッドがすでに呼び出されるようになっている。

```
//onContactBegin——接触した際のイベント
bool MainGame::_onContactBegin(PhysicsContact& contact)
{
    //log("collision begijn!!");
    return true;
}

//onContactPreSolve——接触する直前に発生するイベント
bool MainGame::_onContactPreSolve(PhysicsContact& contact, PhysicsContactPreSolve& solve)
{
    //log("collision pre begijn!!");
    return true;
}

//onContactPostSolve——離れる直前に発生するイベント
void MainGame::_onContactPostSolve(PhysicsContact& contact, const PhysicsContactPostSolve& solve)
{
    //log("collision pre end!!");
}

//onContactSeparate——離れた際のイベント
void MainGame::_onContactSeparate(PhysicsContact& contact)
{
    //log("collision end!!");
}
```

なのでこのメソッドが呼び出されたら、点線追加をやめればいいわけで。
んで、_onContactBeginを使ってみよう。

こんな感じになる

_onContactBeginの内容を掘り下げてみよう。つまり使い方。こゆときはまず見るべきは「引数」である。

```
PhysicsContact& contact
```

PhysicsContactクラスの参照型が渡されている。このクラスはどんなクラスなのか？ググってみたりリファレンスで調べてみた。

リファレンス↓が説明が不親切でここだけではよくわからない
http://www.cocos2d-x.org/reference/native-cpp/V3.2/d5/d22/classcocos2d_1_1_physics_contact.html

調べた結果、ざっくりいうと「衝突情報」が格納されたクラス。ってことは衝突情報がイロイロ入ってくるということで、「何と何が衝突したか」も取れる・・・はず。

```
getShapeA()
getShapeB()
```

これにより剛体が取れるので、これを利用すれば「鳥とそれ以外」つまり、鳥が何かとぶつかったという判定はできる。

こうなった

```
bool MainGame::_onContactBegin(PhysicsContact& contact)
{
    //log("collision begijn!!");

    // 衝突した者同士のノードを取得
    auto nodeA = contact.getShapeA()->getBody()->getNode();
    auto nodeB = contact.getShapeB()->getBody()->getNode();
    if ( nodeA->getTag() == CHAR_OBJTAG || nodeB->getTag() == CHAR_OBJTAG )
    {// もし衝突したのが鳥だったら
        if ( this->_touchEndedFlag == true )
        {// 弾かれて、初めての衝突をしたら点線処理をやめる。

            // ここに点線処理をやめさせるための何を記述

        }
    }

    return true;
}
```

これで判定はOK。_touchEndedFlagを判定しているのは、これがないとゲームが始まってすぐに点線がでなくなってしまうことを避けるためだ。つまり弾いてから後に点線を消すべきだからだ。

次に具体的にどうやって点線を途中で止めるかを考える。

以下は現在の一定間隔で点を打つコード部分。実際の点描画のところは赤文字のところだ。

```
void MainGame::update(float delta)
{
～～～～～
～～～～～
	if ( this->_touchEndedFlag == true )
	{// 飛ばし済み
		this->_cFrame++;
		if ( this->_cFrame > 3 )
		{
			this->_cFrame = 0;
			// 点線を一定間隔で打つ
			auto* draw = dynamic_cast<DrawNode*>(this->getChildByTag(LOCUS_OBJTAG));
			if ( draw != nullptr )
			{//
				auto* charSprite = (Sprite*)this->getChildByTag(CHAR_OBJTAG);
				draw->drawDot(charSprite->getPosition(), 10, Color4F(1.f,0.f,0.f,1.f));
			}
		}
～～～～
～～～～
}
```

かんたんな話、赤字部分を通らなければよいことになる。ここまでくれば簡単。フラグを用意してあげればよい。

というわけでこうなる。

```
void MainGame::update(float delta)
{
～～～～
            this->_cFrame = 0;
            // 点線を一定間隔で打つ
            auto* draw = dynamic_cast<DrawNode*>(this->getChildByTag(LOCUS_OBJTAG));
            if ( draw != nullptr )
            {//
              if ( this->_dotFlag == true )
              {
                auto* charSprite = (Sprite*)this->getChildByTag(CHAR_OBJTAG);
                draw->drawDot(charSprite->getPosition(), 10, Color4F(1.f,0.f,0.f,1.f));
              }            }
          }
～～～～
}
```

```
bool MainGame::_onContactBegin(PhysicsContact& contact)
{
    //log("collision begijn!!");

    // 衝突した者同士のノードを取得
    auto nodeA = contact.getShapeA()->getBody()->getNode();
    auto nodeB = contact.getShapeB()->getBody()->getNode();
    if ( nodeA->getTag() == CHAR_OBJTAG || nodeB->getTag() == CHAR_OBJTAG )
    {// もし衝突したのが鳥だったら
        if ( this->_touchEndedFlag == true )
        {// 弾かれて、初めての衝突をしたら点線処理をやめる。
         // 点線をつけないようにフラグをfalseに。
         this->_dotFlag = false;
        }
    }

    return true;
}
```

```
void MainGame::_commonInit()
{
    // 初期化
    this->_touchEndedFlag = false;  // 0218
    this->_dotFlag = true;
～～～～
```

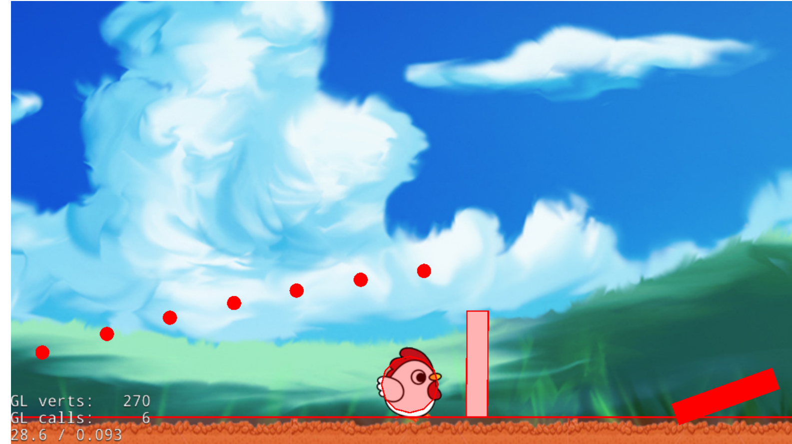
＊あらたな変数として_dotFlagを追加。

実行してみよう！

こうなる

チキンを引っ張り･･･

ぶつかった後は点線でない！



飛ばすと･･･軌跡が！！！

ぼやき

今回のように「点線の追加中止」みたいな場合は、点線クラスを作って、その中で変数などを持たせた方がソースが綺麗なわけで。（将来性を考えて今からクラス分けしてもいいのだけど、今はそこまで考えてません）
「クラスを作るほどでない程度にクラスを拡張したい」という要望はよくあるかと。objective-c, swiftでいうエクステンションという機能がありますが、C++にはない。（別名で何か似たような機能があるのかな？）

っというわけで、むきになってNodeクラスにxxxUserDataというメソッドがあったので、実際のソースはそれを使うことに。bool型のアドレスを保持しています。

```
void setUserData(void*);      //任意のデータへのポインタを保持
void* getUserData();          //任意のデータへのポインタを返す
```

```
draw->setUserData(new bool(true));
(*(bool*)draw->getUserData())
```

可読性が下がるし、メモリリークも怖いので本当は嫌ですが、興味本位で使ってみてます。
以上、ぼやきでした。

本題戻って・・・もう少し改良したい

点線が出るのは、次のショットに生かすため。
ということで、次は1つ前のゲームの点線を残す処理をやってみる

まとめ

NodeにはxxxUserDataというメソッドがあるので、ノードと紐づけたいデータがある場合はこれを使おう。

次回は物理演算Chipmunk
軌跡の点線を入れてみる③編

ここまでのソースプログラムはここ

http:／／monolizm.com／sab／src／AngryChicken18.zip

ご清聴ありがとうございました。