```
//
//  AppDelegate.h
//  BreakOut
//
//  Created by cj on 5/7/13.
//  Copyright (c) 2013 cjdesch. All rights reserved.
//

#import <Cocoa/Cocoa.h>

@interface AppDelegate : NSObject <NSApplicationDelegate>

@property (assign) IBOutlet NSWindow *window;

@end
```

```
//
//  AppDelegate.m
//  BreakOut
//
//  Created by cj on 5/7/13.
//  Copyright (c) 2013 cjdesch. All rights reserved.
//

#import "AppDelegate.h"

@implementation AppDelegate

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
}

@end
```

```
/*
* Copyright (c) 2006-2007 Erin Catto http://www.gphysics.com
*
* iPhone port by Simon Oliver - http://www.simonoliver.com - http://www.handcircus.com
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

//
// File modified for cocos2d integration
// http://www.cocos2d-iphone.org
//

#ifndef GLES_RENDER_H
#define GLES_RENDER_H

#import "cocos2d.h"

#ifdef __CC_PLATFORM_IOS
#import <OpenGLES/EAGL.h>
#elif defined(__CC_PLATFORM_MAC)
#import <OpenGL/OpenGL.h>
#endif

#include "Box2D.h"

struct b2AABB;

// This class implements debug drawing callbacks that are invoked
// inside b2World::Step.
class GLESDebugDraw : public b2Draw
{
        float32 mRatio;
        CCGLProgram *mShaderProgram;
        GLint           mColorLocation;

        void initShader( void );
public:
        GLESDebugDraw();

        GLESDebugDraw( float32 ratio );

        void DrawPolygon(const b2Vec2* vertices, int32 vertexCount, const b2Color& color);

        void DrawSolidPolygon(const b2Vec2* vertices, int32 vertexCount, const b2Color& color);

        void DrawCircle(const b2Vec2& center, float32 radius, const b2Color& color);

        void DrawSolidCircle(const b2Vec2& center, float32 radius, const b2Vec2& axis, const b2Color&
color);

        void DrawSegment(const b2Vec2& p1, const b2Vec2& p2, const b2Color& color);

        void DrawTransform(const b2Transform& xf);

    void DrawPoint(const b2Vec2& p, float32 size, const b2Color& color);

    void DrawString(int x, int y, const char* string, ...);

    void DrawAABB(b2AABB* aabb, const b2Color& color);
};


#endif // GLES_RENDER_H
```

```
/*
 * Copyright (c) 2006-2007 Erin Catto http://www.gphysics.com
 *
 * iPhone port by Simon Oliver - http://www.simonoliver.com - http://www.handcircus.com
 *
 * This software is provided 'as-is', without any express or implied
 * warranty.  In no event will the authors be held liable for any damages
 * arising from the use of this software.
 * Permission is granted to anyone to use this software for any purpose,
 * including commercial applications, and to alter it and redistribute it
 * freely, subject to the following restrictions:
 * 1. The origin of this software must not be misrepresented; you must not
 * claim that you wrote the original software. If you use this software
 * in a product, an acknowledgment in the product documentation would be
 * appreciated but is not required.
 * 2. Altered source versions must be plainly marked as such, and must not be
 * misrepresented as being the original software.
 * 3. This notice may not be removed or altered from any source distribution.
 */

//
// File modified for cocos2d integration
// http://www.cocos2d-iphone.org
//

#import "cocos2d.h"
#include "GLES-Render.h"


#include <cstdio>
#include <cstdarg>

#include <cstring>

GLESDebugDraw::GLESDebugDraw()
: mRatio( 1.0f )
{
        this->initShader();
}

GLESDebugDraw::GLESDebugDraw( float32 ratio )
: mRatio( ratio )
{
        this->initShader();
}

void GLESDebugDraw::initShader( void )
{
        mShaderProgram = [[CCShaderCache sharedShaderCache] programForKey:kCCShader_Position_uColor];

        mColorLocation = glGetUniformLocation( mShaderProgram->program_, "u_color");
}

void GLESDebugDraw::DrawPolygon(const b2Vec2* old_vertices, int32 vertexCount, const b2Color& color)
{
        [mShaderProgram use];
        [mShaderProgram setUniformForModelViewProjectionMatrix];

        ccVertex2F vertices[vertexCount];

        for( int i=0;i<vertexCount;i++) {
                b2Vec2 tmp = old_vertices[i];
                tmp *= mRatio;
                vertices[i].x = tmp.x;
                vertices[i].y = tmp.y;
        }

        [mShaderProgram setUniformLocation:mColorLocation withF1:color.r f2:color.g f3:color.b f4:1];

        glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, vertices);
        glDrawArrays(GL_LINE_LOOP, 0, vertexCount);

        CC_INCREMENT_GL_DRAWS(1);

        CHECK_GL_ERROR_DEBUG();
}

void GLESDebugDraw::DrawSolidPolygon(const b2Vec2* old_vertices, int32 vertexCount, const b2Color& col
or)
{
        [mShaderProgram use];
        [mShaderProgram setUniformForModelViewProjectionMatrix];

        ccVertex2F vertices[vertexCount];

        for( int i=0;i<vertexCount;i++) {
                b2Vec2 tmp = old_vertices[i];
                tmp = old_vertices[i];
                tmp *= mRatio;
                vertices[i].x = tmp.x;
                vertices[i].y = tmp.y;
        }

        [mShaderProgram setUniformLocation:mColorLocation withF1:color.r*0.5f f2:color.g*0.5f f3:color.b*0.5f f4:0.5f];

        glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, vertices);

        glDrawArrays(GL_TRIANGLE_FAN, 0, vertexCount);

        [mShaderProgram setUniformLocation:mColorLocation withF1:color.r f2:color.g f3:color.b f4:1];
        glDrawArrays(GL_LINE_LOOP, 0, vertexCount);

        CC_INCREMENT_GL_DRAWS(2);

        CHECK_GL_ERROR_DEBUG();
}

void GLESDebugDraw::DrawCircle(const b2Vec2& center, float32 radius, const b2Color& color)
{
        [mShaderProgram use];
        [mShaderProgram setUniformForModelViewProjectionMatrix];

        const float32 k_segments = 16.0f;
        int vertexCount=16;
        const float32 k_increment = 2.0f * b2_pi / k_segments;
        float32 theta = 0.0f;

        GLfloat                         glVertices[vertexCount*2];
        for (int32 i = 0; i < k_segments; ++i)
        {
                b2Vec2 v = center + radius * b2Vec2(cosf(theta), sinf(theta));
                glVertices[i*2]=v.x * mRatio;
                glVertices[i*2+1]=v.y * mRatio;
                theta += k_increment;
        }

        [mShaderProgram setUniformLocation:mColorLocation withF1:color.r f2:color.g f3:color.b f4:1];
        glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, glVertices);

        glDrawArrays(GL_LINE_LOOP, 0, vertexCount);

        CC_INCREMENT_GL_DRAWS(1);

        CHECK_GL_ERROR_DEBUG();
}

void GLESDebugDraw::DrawSolidCircle(const b2Vec2& center, float32 radius, const b2Vec2& axis, const b2Color& color)
{
        [mShaderProgram use];
        [mShaderProgram setUniformForModelViewProjectionMatrix];

        const float32 k_segments = 16.0f;
        int vertexCount=16;
        const float32 k_increment = 2.0f * b2_pi / k_segments;
        float32 theta = 0.0f;

        GLfloat                         glVertices[vertexCount*2];
        for (int32 i = 0; i < k_segments; ++i)
        {
                b2Vec2 v = center + radius * b2Vec2(cosf(theta), sinf(theta));
                glVertices[i*2]=v.x * mRatio;
                glVertices[i*2+1]=v.y * mRatio;
                theta += k_increment;
        }


        [mShaderProgram setUniformLocation:mColorLocation withF1:color.r*0.5f f2:color.g*0.5f f3:color.b*0.5f f4:0.5f];
        glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, glVertices);
        glDrawArrays(GL_TRIANGLE_FAN, 0, vertexCount);


        [mShaderProgram setUniformLocation:mColorLocation withF1:color.r f2:color.g f3:color.b f4:1];
        glDrawArrays(GL_LINE_LOOP, 0, vertexCount);

        // Draw the axis line
        DrawSegment(center,center+radius*axis,color);

        CC_INCREMENT_GL_DRAWS(2);

        CHECK_GL_ERROR_DEBUG();
}

void GLESDebugDraw::DrawSegment(const b2Vec2& p1, const b2Vec2& p2, const b2Color& color)
{
        [mShaderProgram use];
        [mShaderProgram setUniformForModelViewProjectionMatrix];

        [mShaderProgram setUniformLocation:mColorLocation withF1:color.r f2:color.g f3:color.b f4:1];

        GLfloat                         glVertices[] = {
                p1.x * mRatio, p1.y * mRatio,
                p2.x * mRatio, p2.y * mRatio
        };

        glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, glVertices);

        glDrawArrays(GL_LINES, 0, 2);

        CC_INCREMENT_GL_DRAWS(1);

        CHECK_GL_ERROR_DEBUG();
}

void GLESDebugDraw::DrawTransform(const b2Transform& xf)
{
        b2Vec2 p1 = xf.p, p2;
        const float32 k_axisScale = 0.4f;
        p2 = p1 + k_axisScale * xf.q.GetXAxis();
        DrawSegment(p1, p2, b2Color(1,0,0));

        p2 = p1 + k_axisScale * xf.q.GetYAxis();
        DrawSegment(p1,p2,b2Color(0,1,0));
}

void GLESDebugDraw::DrawPoint(const b2Vec2& p, float32 size, const b2Color& color)
{
        [mShaderProgram use];
        [mShaderProgram setUniformForModelViewProjectionMatrix];

        [mShaderProgram setUniformLocation:mColorLocation withF1:color.r f2:color.g f3:color.b f4:1];

//      glPointSize(size);

        GLfloat                         glVertices[] = {
                p.x * mRatio, p.y * mRatio
        };

        glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, glVertices);

        glDrawArrays(GL_POINTS, 0, 1);
//      glPointSize(1.0f);

        CC_INCREMENT_GL_DRAWS(1);

        CHECK_GL_ERROR_DEBUG();
}

void GLESDebugDraw::DrawString(int x, int y, const char *string, ...)
{
        //      NSLog(@"DrawString: unsupported: %s", string);
        //printf(string);
        /* Unsupported as yet. Could replace with bitmap font renderer at a later date */
}

void GLESDebugDraw::DrawAABB(b2AABB* aabb, const b2Color& color)
{
        [mShaderProgram use];
        [mShaderProgram setUniformForModelViewProjectionMatrix];

        [mShaderProgram setUniformLocation:mColorLocation withF1:color.r f2:color.g f3:color.b f4:1];

        GLfloat                         glVertices[] = {
                aabb->lowerBound.x * mRatio, aabb->lowerBound.y * mRatio,
                aabb->upperBound.x * mRatio, aabb->lowerBound.y * mRatio,
                aabb->upperBound.x * mRatio, aabb->upperBound.y * mRatio,
                aabb->lowerBound.x * mRatio, aabb->upperBound.y * mRatio
        };

        glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, glVertices);
        glDrawArrays(GL_LINE_LOOP, 0, 8);

        CC_INCREMENT_GL_DRAWS(1);

        CHECK_GL_ERROR_DEBUG();
}
```

```
//
//  HelloWorldLayer.h
//  LeapPuzz
//
//  Created by cj on 2/3/13.
//  Copyright __MyCompanyName__ 2013. All rights reserved.
//


// When you import this file, you import all the cocos2d classes
#import "cocos2d.h"
#import "Box2D.h"
#import "GLES-Render.h"
#import "LeapObjectiveC.h"
#import "RedDot.h"

// HelloWorldLayer
@interface HelloWorldLayer : CCLayer <LeapDelegate>
{

    LeapController *controller;

        CCTexture2D *spriteTexture_;    // weak ref
        b2World* world;                                 // strong ref
        GLESDebugDraw *m_debugDraw;             // strong ref

    CCSprite* targetSprite;
    b2MouseJoint *_mouseJoint;
    b2World* _world;
    b2Body *_groundBody;


    NSMutableDictionary* trackableList;

}
@end
```

```
//
//  HelloWorldLayer.mm
//  LeapPuzz
//
//  Created by cj on 2/3/13.
//  Copyright __MyCompanyName__ 2013. All rights reserved.
//

// Import the interfaces
#import "HelloWorldLayer.h"
#import "PhysicsSprite.h"
//Pixel to metres ratio. Box2D uses metres as the unit for measurement.
//This ratio defines how many pixels correspond to 1 Box2D "metre"
//Box2D is optimized for objects of 1x1 metre therefore it makes sense
//to define the ratio so that your most common object type is 1x1 metre.
#define PTM_RATIO 32

enum {
        kTagParentNode = 1,
};



#pragma mark - HelloWorldLayer

@interface HelloWorldLayer()
-(void) initPhysics;
-(void) addNewSpriteAtPosition:(CGPoint)p;
-(void) createResetButton;
@end

@implementation HelloWorldLayer

-(id) init
{
        if( (self=[super init])) {

                // enable events

#ifdef __IPHONE_OS_VERSION_MAX_ALLOWED
                self.isTouchEnabled = YES;
                self.isAccelerometerEnabled = YES;
#elif defined(__MAC_OS_X_VERSION_MAX_ALLOWED)
                self.isMouseEnabled = YES;
#endif
                CGSize s = [CCDirector sharedDirector].winSize;

                // init physics
                [self initPhysics];

                // create reset button
                [self createResetButton];

                //Set up sprite

#if 1
                // Use batch node. Faster
                CCSpriteBatchNode *parent = [CCSpriteBatchNode batchNodeWithFile:@"blocks.png" capacit
y:100];
                spriteTexture_ = [parent texture];
#else
                // doesn't use batch node. Slower
                spriteTexture_ = [[CCTextureCache sharedTextureCache] addImage:@"blocks.png"];
                CCNode *parent = [CCNode node];
#endif
                [self addChild:parent z:0 tag:kTagParentNode];


                [self addNewSpriteAtPosition:ccp(s.width/2, s.height/2)];

                CCLabelTTF *label = [CCLabelTTF labelWithString:@"LeapPuzz" fontName:@"Marker Felt" fo
ntSize:32];
                [self addChild:label z:0];
                [label setColor:ccc3(0,0,255)];
                label.position = ccp( s.width/2, s.height-50);

                [self scheduleUpdate];

        trackableList = [[NSMutableDictionary alloc] init];

        [self run];


        }
        return self;
}

- (void)run
{
    controller = [[LeapController alloc] init];
    [controller addDelegate:self];
    NSLog(@"running");
}

#pragma mark - SampleDelegate Callbacks

- (void)onInit:(LeapController *)aController
{
    NSLog(@"Initialized");
}

- (void)onConnect:(LeapController *)aController
{
    NSLog(@"Connected");
}

- (void)onDisconnect:(LeapController *)aController
{
    NSLog(@"Disconnected");
}

- (void)onExit:(LeapController *)aController
{
    NSLog(@"Exited");
}

- (void)onFrame:(LeapController *)aController
{
    // Get the most recent frame and report some basic information
    LeapFrame *frame = [aController frame:0];
    /*
    NSLog(@"Frame id: %lld, timestamp: %lld, hands: %ld, fingers: %ld, tools: %ld",
          [frame id], [frame timestamp], [[frame hands] count],
          [[frame fingers] count], [[frame tools] count]);

     */
    if ([[frame hands] count] != 0) {
        // Get the first hand
        LeapHand *hand = [[frame hands] objectAtIndex:0];


        // Check if the hand has any fingers
        NSArray *fingers = [hand fingers];

        if ([fingers count] != 0) {

            // Calculate the hand’s average finger tip position
            LeapVector *avgPos = [[LeapVector alloc] init];
            for (int i = 0; i < [fingers count]; i++) {
                LeapFinger *finger = [fingers objectAtIndex:i];
                avgPos = [avgPos plus:[finger tipPosition]];


                NSString* fingerID = [NSString stringWithFormat:@"%d", finger.id];

                //Check if the Finger ID exists in the list already
                if ([trackableList objectForKey:fingerID]) {

                    //If it does exist update the position on the screen
                    RedDot* sprite = [trackableList objectForKey:fingerID];
                    sprite.position = [self covertLeapCoordinates:CGPointMake(finger.tipPosition.x, finger.tipPosition.y)];
                    sprite.updated = TRUE;


                }else{

                    NSLog(@"x %0.0f y %0.0f z %0.0f", finger.tipPosition.x, finger.tipPosition.y, finger.tipPosition.z);
                  // CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
```

```
                    //CGPoint mouseLocation = [self convertToNodeSpace:point];

                    //Add it to the dictionary
                    RedDot* redDot = [self addRedDot:CGPointMake(finger.tipPosition.x, finger.tipPosit
ion.y) finger:fingerID];
                    [trackableList setObject:redDot forKey:fingerID];
                }
            }

            avgPos = [avgPos divide:[fingers count]];

            //NSLog(@"Hand has %ld fingers, average finger tip position %@", [fingers count], avgPos);
            for (LeapFinger* finger in fingers){

                //NSLog(@"Finger ID %d %ld", finger.id, (unsigned long)[finger hash]);
            }

        }

        //
        [self checkFingerExists];

        // Get the hand’s sphere radius and palm position
        /*
        NSLog(@"Hand sphere radius: %f mm, palm position: %@",
              [hand sphereRadius], [hand palmPosition]);
        */
        // Get the hand’s normal vector and direction
        const LeapVector *normal = [hand palmNormal];
        const LeapVector *direction = [hand direction];

        /*
        // Calculate the hand’s pitch, roll, and yaw angles
        NSLog(@"Hand pitch: %f degrees, roll: %f degrees, yaw: %f degrees\n",
              [direction pitch] * LEAP_RAD_TO_DEG,
              [normal roll] * LEAP_RAD_TO_DEG,
              [direction yaw] * LEAP_RAD_TO_DEG);
         */
    }
}


- (void)moveRedDot{


}

//Cycle through all the trackable dots and check if the fingers still exist.
//If they don’t, delete them.
- (void)checkFingerExists{

    for (id key in [trackableList allKeys]) {
        RedDot* sprite = [trackableList objectForKey:key];
        if (sprite.updated) {
            sprite.updated = FALSE;
            return;
        }else{
            CCNode *parent = [self getChildByTag:kTagParentNode];
            [trackableList removeObjectForKey:key];
            [parent removeChild:sprite cleanup:YES];

        }
    }
}


#pragma mark -

-(void) createResetButton
{
        CCMenuItemLabel *reset = [CCMenuItemFont itemWithString:@"Reset" block:^(id sender){
                CCScene *s = [CCScene node];
                id child = [HelloWorldLayer node];
                [s addChild:child];
                [[CCDirector sharedDirector] replaceScene: s];
        }];

        CCMenu *menu = [CCMenu menuWithItems:reset, nil];

        CGSize s = [[CCDirector sharedDirector] winSize];

        menu.position = ccp(s.width/2, 30);
        [self addChild: menu z:-1];

}

-(void) initPhysics
{

        CGSize s = [[CCDirector sharedDirector] winSize];

    //Gravity
        b2Vec2 gravity;
        gravity.Set(0.0f, 0.0f);
        world = new b2World(gravity);


        // Do we want to let bodies sleep?
        world->SetAllowSleeping(true);

        world->SetContinuousPhysics(true);

        m_debugDraw = new GLESDebugDraw( PTM_RATIO );
        world->SetDebugDraw(m_debugDraw);

    _world = world;

        uint32 flags = 0;
        flags += b2Draw::e_shapeBit;
        //              flags += b2Draw::e_jointBit;
        //              flags += b2Draw::e_aabbBit;
        //              flags += b2Draw::e_pairBit;
        //              flags += b2Draw::e_centerOfMassBit;
        m_debugDraw->SetFlags(flags);


        // Define the ground body.
        b2BodyDef groundBodyDef;
        groundBodyDef.position.Set(0, 0); // bottom-left corner

        // Call the body factory which allocates memory for the ground body
        // from a pool and creates the ground box shape (also from a pool).
        // The body is also added to the world.
        b2Body* groundBody = world->CreateBody(&groundBodyDef);

        // Define the ground box shape.
        b2EdgeShape groundBox;

        // bottom

        groundBox.Set(b2Vec2(0,0), b2Vec2(s.width/PTM_RATIO,0));
        groundBody->CreateFixture(&groundBox,0);

        // top
        groundBox.Set(b2Vec2(0,s.height/PTM_RATIO), b2Vec2(s.width/PTM_RATIO,s.height/PTM_RATIO));
        groundBody->CreateFixture(&groundBox,0);

        // left
        groundBox.Set(b2Vec2(0,s.height/PTM_RATIO), b2Vec2(0,0));
        groundBody->CreateFixture(&groundBox,0);

        // right
        groundBox.Set(b2Vec2(s.width/PTM_RATIO,s.height/PTM_RATIO), b2Vec2(s.width/PTM_RATIO,0));
        groundBody->CreateFixture(&groundBox,0);

    _groundBody = groundBody;
}

-(void) draw
{
        //
        // IMPORTANT:
        // This is only for debug purposes
        // It is recommend to disable it
        //
        [super draw];

        ccGLEnableVertexAttribs( kCCVertexAttribFlag_Position );

        kmGLPushMatrix();

        world->DrawDebugData();

        kmGLPopMatrix();
}

- (RedDot*)addRedDot:(CGPoint)p finger:(NSString*)fingerID{
    CCNode *parent = [self getChildByTag:kTagParentNode];
    int idx = (CCRANDOM_0_1() > .5 ? 0:1);
        int idy = (CCRANDOM_0_1() > .5 ? 0:1);

        //RedDot *sprite = [RedDot spriteWithFile:@"redcrosshair.png"];
    RedDot *sprite = [RedDot spriteWithTexture:spriteTexture_ rect:CGRectMake(32 * idx,32 * idy,32,32)];
        [parent addChild:sprite];
    sprite.updated = TRUE;
    sprite.fingerID = fingerID;
    sprite.position = ccp( p.x, p.y);

    return sprite;
}

- (CGPoint)covertLeapCoordinates:(CGPoint)p{

    CGSize s = [[CCDirector sharedDirector] winSize];
    float screenCenter = 0.0f;
    float xScale = 1.75f;
    float yScale = 1.25f;
    return CGPointMake((s.width/2)+ (( p.x - screenCenter) * xScale), p.y * yScale);
}

-(void) addNewSpriteAtPosition:(CGPoint)p
{
        CCLOG(@"Add sprite %0.2f x %02.f",p.x,p.y);
        CCNode *parent = [self getChildByTag:kTagParentNode];

        //We have a 64x64 sprite sheet with 4 different 32x32 images.  The following code is
        //just randomly picking one of the images
        int idx = (CCRANDOM_0_1() > .5 ? 0:1);
        int idy = (CCRANDOM_0_1() > .5 ? 0:1);
        PhysicsSprite *sprite = [PhysicsSprite spriteWithTexture:spriteTexture_ rect:CGRectMake(32 * idx,32 * idy,32,32)];
        [parent addChild:sprite];
        sprite.position = [self covertLeapCoordinates:p];
        //sprite.position = ccp( p.x, p.y);

        // Define the dynamic body.
        //Set up a 1m squared box in the physics world
        b2BodyDef bodyDef;
        bodyDef.type = b2_dynamicBody;
        bodyDef.position.Set(p.x/PTM_RATIO, p.y/PTM_RATIO);

    //bodyDef.userData  = (void *) CFBridgingRetain(sprite);
    bodyDef.userData  = (__bridge void *)sprite;
        b2Body *body = world->CreateBody(&bodyDef);

        // Define another box shape for our dynamic body.
        b2PolygonShape dynamicBox;
        dynamicBox.SetAsBox(.5f, .5f);//These are mid points for our 1m box

        // Define the dynamic body fixture.
        b2FixtureDef fixtureDef;
        fixtureDef.shape = &dynamicBox;
        fixtureDef.density = 1.0f;
        fixtureDef.friction = 0.3f;
        body->CreateFixture(&fixtureDef);

        [sprite setPhysicsBody:body];
}

-(void) addPieceAtPosition:(CGPoint)p
{
        CCLOG(@"Add sprite %0.2f x %02.f",p.x,p.y);
        CCNode *parent = [self getChildByTag:kTagParentNode];

        //We have a 64x64 sprite sheet with 4 different 32x32 images.  The following code is
        //just randomly picking one of the images
        int idx = (CCRANDOM_0_1() > .5 ? 0:1);
        int idy = (CCRANDOM_0_1() > .5 ? 0:1);
        PhysicsSprite *sprite = [PhysicsSprite spriteWithTexture:spriteTexture_ rect:CGRectMake(32 * i
dx,32 * idy,32,32)];
        [parent addChild:sprite];

        sprite.position = ccp( p.x, p.y);

        // Define the dynamic body.
        //Set up a 1m squared box in the physics world
        b2BodyDef bodyDef;
        bodyDef.type = b2_dynamicBody;
        bodyDef.position.Set(p.x/PTM_RATIO, p.y/PTM_RATIO);
        b2Body *body = world->CreateBody(&bodyDef);

        // Define another box shape for our dynamic body.
        b2PolygonShape dynamicBox;
        dynamicBox.SetAsBox(.5f, .5f);//These are mid points for our 1m box

        // Define the dynamic body fixture.
        b2FixtureDef fixtureDef;
        fixtureDef.shape = &dynamicBox;
        fixtureDef.density = 1.0f;
        fixtureDef.friction = 0.3f;
        body->CreateFixture(&fixtureDef);

        [sprite setPhysicsBody:body];
}

-(void) update: (ccTime) dt
{
        //It is recommended that a fixed time step is used with Box2D for stability
        //of the simulation, however, we are using a variable time step here.
        //You need to make an informed choice, the following URL is useful
        //http://gafferongames.com/game-physics/fix-your-timestep/

        int32 velocityIterations = 8;
        int32 positionIterations = 1;

        // Instruct the world to perform a single step of simulation. It is
        // generally best to keep the time step and iterations fixed.
        world->Step(dt, velocityIterations, positionIterations);
}

#ifdef __IPHONE_OS_VERSION_MAX_ALLOWED

- (void)ccTouchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
        //Add a new body/atlas sprite at the touched location
        for( UITouch *touch in touches ) {
                CGPoint location = [touch locationInView: [touch view]];

                location = [[CCDirector sharedDirector] convertToGL: location];

                [self addNewSpriteAtPosition: location];


        }
}

#elif defined(__MAC_OS_X_VERSION_MAX_ALLOWED)
/*
- (BOOL)ccTouchBegan:(UITouch *)touch withEvent:(UIEvent *)event {
    CGPoint touchLocation = [self convertTouchToNodeSpace:touch];
    [self selectSpriteForTouch:touchLocation];
    return TRUE;
}
 */




#pragma mark - Touch Handling

- (BOOL) ccMouseDown:(NSEvent *)event{

    if (_mouseJoint != NULL) return NO;


    CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
    CGPoint mouseLocation = [self convertToNodeSpace:point];
    CGPoint translation = (mouseLocation);
    CGPoint location = translation;
    //location = [[CCDirector sharedDirector] convertToGL:location];
    b2Vec2 locationWorld = b2Vec2(location.x/PTM_RATIO, location.y/PTM_RATIO);


    // Loop through all of the Box2D bodies in our Box2D world..
    for(b2Body *b = _world->GetBodyList(); b; b=b->GetNext()) {


        // See if there’s any user data attached to the Box2D body
        // There should be, since we set it in addBoxBodyForSprite

        if (b->GetUserData() != NULL) {
            // We know that the user data is a sprite since we set
            // it that way, so cast it...

            //PhysicsSprite *sprite = (PhysicsSprite *)CFBridgingRelease(b->GetUserData());


            for(b2Fixture *fixture = b->GetFixtureList(); fixture; fixture=fixture->GetNext()) {

                if(fixture->TestPoint(locationWorld)){
                    //NSLog(@"Touched itemType %d", sprite.itemType);
                    b2MouseJointDef md;
                    md.bodyA = _groundBody;
                    md.bodyB = b;
                    md.target = locationWorld;
                    md.collideConnected = true;
                    md.maxForce = 1000.0f * b->GetMass();

                    _mouseJoint = (b2MouseJoint *)_world->CreateJoint(&md);
                    b->SetAwake(true);
                }else{
                    //NSLog(@"NOT TOUCHED");
                }
            }
        }
    }
    return YES;
}

- (BOOL)ccMouseDragged:(NSEvent *)event {

    if (_mouseJoint == NULL) return NO;


    CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
    CGPoint mouseLocation = [self convertToNodeSpace:point];
    CGPoint translation = (mouseLocation);
    CGPoint location = translation;
    //location = [[CCDirector sharedDirector] convertToGL:location];
    b2Vec2 locationWorld = b2Vec2(location.x/PTM_RATIO, location.y/PTM_RATIO);

    _mouseJoint->SetTarget(locationWorld);

    return YES;

}

- (BOOL)ccMouseUp:(NSEvent *)event{
    if (_mouseJoint) {
        _world->DestroyJoint(_mouseJoint);
        _mouseJoint = NULL;

        //Check for any dangling mouse joints
        if(_world->GetJointCount() > 0){
            //NSLog(@"Found %d Extra Joints", _world->GetJointCount() );
            for(b2Joint *b = _world->GetJointList(); b; b=b->GetNext()) {
                //NSLog(@"Destproying the Dangling Joint");
                //Should check type first
                if(b){
                    _world->DestroyJoint(b);
                    b = NULL;
                    return YES;
                }
            }
        }
    }else{


        CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
        CGPoint mouseLocation = [self convertToNodeSpace:point];
        CGPoint translation = (mouseLocation);
        CGPoint location = translation;


                [self addNewSpriteAtPosition: location];

    }
    return YES;
}

#endif

@end
```

```
//
//  HelloWorldScene.h
//  Cocos2DBreakout2
//


#import "cocos2d.h"
#import "Box2D.h"
#import "MyContactListener.h"
#import "LeapObjectiveC.h"

@interface HelloWorld : CCLayer <LeapDelegate>{
    b2World *_world;
    b2Body *_groundBody;
    b2Body *_paddleBody;
    b2Fixture *_paddleFixture;
    b2Fixture *_ballFixture;
    b2Fixture *_bottomFixture;
    b2MouseJoint *_mouseJoint;
    b2MouseJoint *_fingerJoint;
    MyContactListener *_contactListener;

    LeapController *controller;
    NSMutableDictionary* trackableList;
    BOOL fingerTracked;
}

+ (id) scene;

@end
```

```
//
//  HelloWorldScene.m
//  Cocos2DBreakout2
//

#import "HelloWorldScene.h"

#import "SimpleAudioEngine.h"
#import "TrackedFinger.h"
#define PTM_RATIO 32

@implementation HelloWorld

+ (id)scene {

    CCScene *scene = [CCScene node];
    HelloWorld *layer = [HelloWorld node];
    [scene addChild:layer];
    return scene;

}

- (id)init {

    if ((self=[super init])) {

        CGSize s = [CCDirector sharedDirector].winSize;

        self.isMouseEnabled = YES;

        // Create a world
        b2Vec2 gravity = b2Vec2(0.0f, 0.0f);
        bool doSleep = true;
        _world = new b2World(gravity);

        // Define the ground body.
        b2BodyDef groundBodyDef;
        groundBodyDef.position.Set(0, 0); // bottom-left corner

        // Call the body factory which allocates memory for the ground body
        // from a pool and creates the ground box shape (also from a pool).
        // The body is also added to the world.
        b2Body* groundBody = _world->CreateBody(&groundBodyDef);

        // Define the ground box shape.
        b2EdgeShape groundBox;

        // bottom

        groundBox.Set(b2Vec2(0,0), b2Vec2(s.width/PTM_RATIO,0));
        groundBody->CreateFixture(&groundBox,0);

        // top
        groundBox.Set(b2Vec2(0,s.height/PTM_RATIO), b2Vec2(s.width/PTM_RATIO,s.height/PTM_RATIO));
        groundBody->CreateFixture(&groundBox,0);

        // left
        groundBox.Set(b2Vec2(0,s.height/PTM_RATIO), b2Vec2(0,0));
        groundBody->CreateFixture(&groundBox,0);

        // right
        groundBox.Set(b2Vec2(s.width/PTM_RATIO,s.height/PTM_RATIO), b2Vec2(s.width/PTM_RATIO,0));
        groundBody->CreateFixture(&groundBox,0);

        _groundBody = groundBody;

        // Create sprite and add it to the layer
        CCSprite *ball = [CCSprite spriteWithFile:@"Ball.png" rect:CGRectMake(0, 0, 52, 52)];
        ball.position = ccp(100, 100);
        ball.tag = 1;
        [self addChild:ball];

        // Create ball body
        b2BodyDef ballBodyDef;
        ballBodyDef.type = b2_dynamicBody;
        ballBodyDef.position.Set(100/PTM_RATIO, 100/PTM_RATIO);
        ballBodyDef.userData =  (__bridge void *) ball;
        b2Body * ballBody = _world->CreateBody(&ballBodyDef);

        // Create circle shape
        b2CircleShape circle;
        circle.m_radius = 26.0/PTM_RATIO;

        // Create shape definition and add to body
        b2FixtureDef ballShapeDef;
        ballShapeDef.shape = &circle;
        ballShapeDef.density = 1.0f;
        ballShapeDef.friction = 0.0f; // We don't want the ball to have friction!
        ballShapeDef.restitution = 1.0f;
        _ballFixture = ballBody->CreateFixture(&ballShapeDef);

        // Give shape initial impulse...
        b2Vec2 force = b2Vec2(10, 10);
        ballBody->ApplyLinearImpulse(force, ballBodyDef.position);

        // Create paddle and add it to the layer
        CCSprite *paddle = [CCSprite spriteWithFile:@"Paddle.png"];
        paddle.position = ccp(s.width/2, 50);
        [self addChild:paddle];

        // Create paddle body
        b2BodyDef paddleBodyDef;
        paddleBodyDef.type = b2_dynamicBody;
        paddleBodyDef.position.Set(s.width/2/PTM_RATIO, 50/PTM_RATIO);
        paddleBodyDef.userData =  (__bridge void *) paddle;
        _paddleBody = _world->CreateBody(&paddleBodyDef);

        // Create paddle shape
        b2PolygonShape paddleShape;
        paddleShape.SetAsBox(paddle.contentSize.width/PTM_RATIO/2,
                             paddle.contentSize.height/PTM_RATIO/2);

        // Create shape definition and add to body
        b2FixtureDef paddleShapeDef;
        paddleShapeDef.shape = &paddleShape;
        paddleShapeDef.density = 10.0f;
        paddleShapeDef.friction = 0.4f;
        paddleShapeDef.restitution = 0.1f;
        _paddleFixture = _paddleBody->CreateFixture(&paddleShapeDef);

        // Restrict paddle along the x axis
        b2PrismaticJointDef jointDef;
        b2Vec2 worldAxis(1.0f, 0.0f);
        jointDef.collideConnected = true;
        jointDef.Initialize(_paddleBody, _groundBody, _paddleBody->GetWorldCenter(), worldAxis);
        _world->CreateJoint(&jointDef);

        for(int i = 0; i < 10; i++) {
            static int padding=20;

            for (int j = 500; j < 1000; j+=100){
                // Create block and add it to the layer
                CCSprite *block = [CCSprite spriteWithFile:@"Block.png"];
                int xOffset = padding+block.contentSize.width/2+((block.contentSize.width+padding)*i);
                int yOffset = j; //padding+block.contentSize.height/2+((block.contentSize.height+padding)*i);
                block.position = ccp(xOffset, yOffset);
                block.tag = 2;
                [self addChild:block];

                // Create block body
                b2BodyDef blockBodyDef;
                blockBodyDef.type = b2_dynamicBody;
                blockBodyDef.position.Set(xOffset/PTM_RATIO, yOffset/PTM_RATIO);
                blockBodyDef.userData =  (__bridge void *) block;
                b2Body *blockBody = _world->CreateBody(&blockBodyDef);

                // Create block shape
                b2PolygonShape blockShape;
                blockShape.SetAsBox(block.contentSize.width/PTM_RATIO/2,
                                    block.contentSize.height/PTM_RATIO/2);

                // Create shape definition and add to body
                b2FixtureDef blockShapeDef;
                blockShapeDef.shape = &blockShape;
                blockShapeDef.density = 10.0;
                blockShapeDef.friction = 0.0;
                blockShapeDef.restitution = 0.1f;
                blockBody->CreateFixture(&blockShapeDef);
            }




        }



        // Create contact listener
        _contactListener = new MyContactListener();
        _world->SetContactListener(_contactListener);

        [[SimpleAudioEngine sharedEngine] playBackgroundMusic:@"background-music-aac.caf"];

        [self schedule:@selector(tick:)];

        [self run];
        fingerTracked = FALSE;

        [self addFingerJoint];

    }
    return self;

}


- (void)run
{
    controller = [[LeapController alloc] init];
    [controller addDelegate:self];
    NSLog(@"running");
}

#pragma mark - SampleDelegate Callbacks

- (void)onInit:(LeapController *)aController
{
    NSLog(@"Initialized");
}

- (void)onConnect:(LeapController *)aController
{
    NSLog(@"Connected");
}

- (void)onDisconnect:(LeapController *)aController
{
    NSLog(@"Disconnected");
}

- (void)onExit:(LeapController *)aController
{
    NSLog(@"Exited");
}

- (void)onFrame:(LeapController *)aController
{
    // Get the most recent frame and report some basic information
    LeapFrame *frame = [aController frame:0];
    /*
     NSLog(@"Frame id: %lld, timestamp: %lld, hands: %ld, fingers: %ld, tools: %ld",
     [frame id], [frame timestamp], [[frame hands] count],
     [[frame fingers] count], [[frame tools] count]);

     */
    if ([[frame hands] count] != 0) {
        // Get the first hand
        LeapHand *hand = [[frame hands] objectAtIndex:0];

        // Check if the hand has any fingers
        NSArray *fingers = [hand fingers];
        if ([fingers count] != 0) {
            // Calculate the hand's average finger tip position
            LeapVector *avgPos = [[LeapVector alloc] init];
            for (int i = 0; i < [fingers count]; i++) {
                LeapFinger *finger = [fingers objectAtIndex:i];
                avgPos = [avgPos plus:[finger tipPosition]];

                NSString* fingerID = [NSString stringWithFormat:@"%d", finger.id];
                /*
                //Check if the Finger ID exists in the list already
                if ([trackableList objectForKey:fingerID]) {

                    //If it does exist update the position on the screen
                    TrackedFinger* sprite = [trackableList objectForKey:fingerID];
                    sprite.updated = TRUE;


                }else{

                    //NSLog(@"x %0.0f y %0.0f z %0.0f", finger.tipPosition.x, finger.tipPosition.y, fi
nger.tipPosition.z);
                    // CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
                    //CGPoint mouseLocation = [self convertToNodeSpace:point];
                    //Add it to the dictionary
                    TrackedFinger* redDot = [[TrackedFinger alloc] initWithID:fingerID];
                    [trackableList setObject:redDot forKey:fingerID];
                }
                 */
            }

            avgPos = [avgPos divide:[fingers count]];
            //NSLog(@"x %0.0f y %0.0f z %0.0f", avgPos.x, avgPos.y, avgPos.z);
            [self fingerMoved:CGPointMake(avgPos.x, avgPos.y)];


            //NSLog(@"Hand has %ld fingers, average finger tip position %@", [fingers count], avgPos);
            for (LeapFinger* finger in fingers){

                //NSLog(@"Finger ID %d %ld", finger.id, (unsigned long)[finger hash]);
            }

        }

        //[self checkFingerExists];

        // Get the hand's sphere radius and palm position
        /*
         NSLog(@"Hand sphere radius: %f mm, palm position: %@",
         [hand sphereRadius], [hand palmPosition]);
         */
        // Get the hand's normal vector and direction
        const LeapVector *normal = [hand palmNormal];
        const LeapVector *direction = [hand direction];

        /*
         // Calculate the hand's pitch, roll, and yaw angles
         NSLog(@"Hand pitch: %f degrees, roll: %f degrees, yaw: %f degrees\n",
         [direction pitch] * LEAP_RAD_TO_DEG,
         [normal roll] * LEAP_RAD_TO_DEG,
         [direction yaw] * LEAP_RAD_TO_DEG);
         */
    }
}


- (void)tick:(ccTime) dt {

    bool blockFound = false;
    _world->Step(dt, 10, 10);
    for(b2Body *b = _world->GetBodyList(); b; b=b->GetNext()) {
        if (b->GetUserData() != NULL) {
            CCSprite *sprite = (__bridge CCSprite *)b->GetUserData();
            if (sprite.tag == 2) {
                blockFound = true;
            }

            if (sprite.tag == 1) {
                static int maxSpeed = 10;

                b2Vec2 velocity = b->GetLinearVelocity();
                float32 speed = velocity.Length();

                // When the ball is greater than max speed, slow it down by
                // applying linear damping.  This is better for the simulation
                // than raw adjustment of the velocity.
                if (speed > maxSpeed) {
                    b->SetLinearDamping(0.5);
                } else if (speed < maxSpeed) {
                    b->SetLinearDamping(0.0);
                }

            }

            sprite.position = ccp(b->GetPosition().x * PTM_RATIO,
                                  b->GetPosition().y * PTM_RATIO);
            sprite.rotation = -1 * CC_RADIANS_TO_DEGREES(b->GetAngle());
        }
    }

    if (!blockFound) {
        /*
        GameOverScene *gameOverScene = [GameOverScene node];
        [gameOverScene.layer.label setString:@"You Win!"];
        [[CCDirector sharedDirector] replaceScene:gameOverScene];
         */
        NSLog(@"GameOver");
    }

    std::vector<b2Body *>toDestroy;
    std::vector<MyContact>::iterator pos;
    for(pos = _contactListener->_contacts.begin(); pos != _contactListener->_contacts.end(); ++pos) {
        MyContact contact = *pos;

        if ((contact.fixtureA == _bottomFixture && contact.fixtureB == _ballFixture) ||
            (contact.fixtureA == _ballFixture && contact.fixtureB == _bottomFixture)) {

            NSLog(@"GameOver");
        }

        b2Body *bodyA = contact.fixtureA->GetBody();
        b2Body *bodyB = contact.fixtureB->GetBody();
        if (bodyA->GetUserData() != NULL && bodyB->GetUserData() != NULL) {
            CCSprite *spriteA = (__bridge CCSprite *) bodyA->GetUserData();
            CCSprite *spriteB = (__bridge CCSprite *) bodyB->GetUserData();

            // Sprite A = ball, Sprite B = Block
            if (spriteA.tag == 1 && spriteB.tag == 2) {
                if (std::find(toDestroy.begin(), toDestroy.end(), bodyB) == toDestroy.end()) {
                    toDestroy.push_back(bodyB);
                }
            }
            // Sprite B = block, Sprite A = ball
            else if (spriteA.tag == 2 && spriteB.tag == 1) {
                if (std::find(toDestroy.begin(), toDestroy.end(), bodyA) == toDestroy.end()) {
                    toDestroy.push_back(bodyA);
                }
            }
        }
    }

    std::vector<b2Body *>::iterator pos2;
    for(pos2 = toDestroy.begin(); pos2 != toDestroy.end(); ++pos2) {
        b2Body *body = *pos2;
        if (body->GetUserData() != NULL) {
            CCSprite *sprite = (__bridge CCSprite *) body->GetUserData();
            [self removeChild:sprite cleanup:YES];
        }
        _world->DestroyBody(body);
    }

    if (toDestroy.size() > 0) {
        [[SimpleAudioEngine sharedEngine] playEffect:@"blip.caf"];
    }

}

- (CGPoint)covertLeapCoordinates:(CGPoint)p{

    CGSize s = [[CCDirector sharedDirector] winSize];
    float screenCenter = 0.0f;
    float xScale = 3.25f;
    float yScale = 1.25f;
    return CGPointMake((s.width/2)+ (( p.x - screenCenter) * xScale), p.y * yScale);
}


//Cycle through all the trackable dots and check if the fingers still exist.
//If they don't, delete them.
```

```
- (void)checkFingerExists{
    for (id key in [trackableList allKeys]) {
        TrackedFinger* sprite = [trackableList objectForKey:key];
        if (sprite.updated) {
            sprite.updated = FALSE;
            return;
        }else{

            [trackableList removeObjectForKey:key];

        }
    }

    if ([trackableList count] == 0){
        [self fingerLost];
    }
}

- (void)addFingerJoint{

    b2MouseJointDef md;
    md.bodyA = _groundBody;
    md.bodyB = _paddleBody;

    md.target = _paddleBody->GetPosition();
    md.collideConnected = true;
    md.maxForce = 1000.0f * _paddleBody->GetMass();

    _fingerJoint = (b2MouseJoint *)_world->CreateJoint(&md);
    _paddleBody->SetAwake(true);

}



- (void)fingerMoved:(CGPoint)point{


    if (_fingerJoint == NULL) return;



    CGPoint location = [self covertLeapCoordinates:point];
    NSLog(@"Dragged %0.0f , %0.0f ", location.x, location.y);
    b2Vec2 locationWorld = b2Vec2(location.x/PTM_RATIO, location.y/PTM_RATIO);

    _fingerJoint->SetTarget(locationWorld);
}

- (void)fingerLost{

    if (_fingerJoint) {
        _world->DestroyJoint(_fingerJoint);
        _fingerJoint = NULL;
    }

}


//- (void)ccTouchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
- (BOOL) ccMouseDown:(NSEvent *)event{

    if (_mouseJoint != NULL) return NO;

    if (_fingerJoint) {

        //_fingerJoint->SetMaxForce(0);
        _world->DestroyJoint(_fingerJoint);
        _fingerJoint = NULL;
    }


    /*
    UITouch *myTouch = [touches anyObject];
    CGPoint location = [myTouch locationInView:[myTouch view]];
    location = [[CCDirector sharedDirector] convertToGL:location];
    */
    CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
    CGPoint mouseLocation = [self convertToNodeSpace:point];
    CGPoint translation = (mouseLocation);
    CGPoint location = translation;

    b2Vec2 locationWorld = b2Vec2(location.x/PTM_RATIO, location.y/PTM_RATIO);

    if (_paddleFixture->TestPoint(locationWorld)) {
        b2MouseJointDef md;
        md.bodyA = _groundBody;
        md.bodyB = _paddleBody;
        md.target = locationWorld;
        md.collideConnected = true;
        md.maxForce = 1000.0f * _paddleBody->GetMass();

        _mouseJoint = (b2MouseJoint *)_world->CreateJoint(&md);
        _paddleBody->SetAwake(true);
    }

}

//-(void)ccTouchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
- (BOOL)ccMouseDragged:(NSEvent *)event {

    if (_mouseJoint == NULL) return NO;
    CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
    CGPoint mouseLocation = [self convertToNodeSpace:point];
    CGPoint translation = (mouseLocation);
    CGPoint location = translation;
    b2Vec2 locationWorld = b2Vec2(location.x/PTM_RATIO, location.y/PTM_RATIO);

    _mouseJoint->SetTarget(locationWorld);

}

//-(void)ccTouchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event {
- (BOOL)ccMouseUp:(NSEvent *)event{

    if (_mouseJoint) {
        _world->DestroyJoint(_mouseJoint);
        _mouseJoint = NULL;

        [self addFingerJoint];
    }else{

        //_fingerJoint->SetMaxForce(1000.0f * _paddleBody->GetMass());
        //_fingerJoint->

    }

}

@end
```

```
//
//  main.m
//  BreakOut
//
//  Created by cj on 5/7/13.
//  Copyright (c) 2013 cjdesch. All rights reserved.
//

#import <Cocoa/Cocoa.h>

int main(int argc, char *argv[])
{
    return NSApplicationMain(argc, (const char **)argv);
}
```

```
//
//  PhysicsSprite.h
//  LeapPuzz
//
//  Created by cj on 2/8/13.
//
//

#import <Foundation/Foundation.h>
#import "cocos2d.h"
#import "Box2D.h"
#import "GLES-Render.h"
#import "LeapObjectiveC.h"


@interface PhysicsSprite :  CCSprite <CCMouseEventDelegate>
{       CGPoint target;
        uint ticker;
    bool hasTarget;
        b2Body *body_;  // strong ref
}

-(void) setPhysicsBody:(b2Body*)body;
-(void) setTarget:(CGPoint)p;
-(void) delTarget;

@end
```

```
//
//  PhysicsSprite.m
//  LeapPuzz
//
//  Created by cj on 2/8/13.
//
//

#import "PhysicsSprite.h"
#define PTM_RATIO 32

enum {
        kTagParentNode = 1,
};

@implementation PhysicsSprite
-(void) setPhysicsBody:(b2Body *)body
{
    //[[CCEventDispatcher sharedDispatcher] addMouseDelegate:self priority:0];
    //[[[CCDirector sharedDirector] eventDispatcher] addMouseDelegate:self priority:-1];
    hasTarget = NO;
    //[[CCTouchDispatcher sharedDispatcher] addTargetedDelegate:self priority:50 swallowsTouches:YES];
        body_ = body;
}

// this method will only get called if the sprite is batched.
// return YES if the physics values (angles, position ) changed
// If you return NO, then nodeToParentTransform won't be called.
-(BOOL) dirty
{
        return YES;
}

-(void) setTarget:(CGPoint)p
{
    hasTarget = YES;
    target = p;
}

-(void) delTarget
{
    hasTarget = NO;
}




- (BOOL)ccMouseDragged:(NSEvent *)event{

    //NSLog(@"Mouse dragged in Sprite");
    if (hasTarget){
        NSLog(@"Mouse dragged in Sprite");
        CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
        //CGPoint mouseLocation = [self convertToNodeSpace:point];
        CGPoint translation = (point);

        //NSLog(@"Dragged %0.0f , %0.0f ", translation.x, translation.y);
        //self.position = translation;
        //body_->Get
        self.position = ccp(translation.x, translation.y);
    }

    return YES;

}

- (BOOL)ccMouseDown:(NSEvent *)event{

    //NSLog(@"Mouse Down");


    CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
    //CGPoint mouseLocation = [self convertToNodeSpace:point];
    CGPoint translation = (point);
    //NSLog(@"Dragged %0.0f , %0.0f ", translation.x, translation.y);
    //NSLog(@"Bouding Box %0.0f %0.0f %0.0f %0.0f", self.boundingBox.origin.x, self.boundingBox.origin
.y, self.boundingBox.size.height,self.boundingBox.size.width );

    if (CGRectContainsPoint([self boundingBox], translation)){
        NSLog(@"Sprite inside Touched");
        [self setTarget:translation];
        return YES;

    }
    /*
     CCNode *parent = [self getChildByTag:kTagParentNode];
     for (PhysicsSprite *sprite in parent.children){


     if (CGRectContainsPoint([sprite boundingBox], translation)){
     NSLog(@"Sprite Touched");

     [self setTarget:translation ];

     //Move Sprite with

     //sprite.position = translation;

     }

     }
     */

    return NO;
}

- (BOOL)ccMouseMoved:(NSEvent *)event{


    NSLog(@"Mouse Moved in Sprite");

    return YES;
}

- (BOOL)ccMouseUp:(NSEvent *)event{

    [self delTarget];

    return YES;

}

// returns the transform matrix according the Chipmunk Body values
-(CGAffineTransform) nodeToParentTransform
{
        b2Vec2 pos  = body_->GetPosition();

        float x = pos.x * PTM_RATIO;
        float y = pos.y * PTM_RATIO;

        if ( ignoreAnchorPointForPosition_ ) {
                x += anchorPointInPoints_.x;
                y += anchorPointInPoints_.y;
        }

        // Make matrix
        float radians = body_->GetAngle();
        float c = cosf(radians);
        float s = sinf(radians);

        if( ! CGPointEqualToPoint(anchorPointInPoints_, CGPointZero) ){
                x += c*-anchorPointInPoints_.x + -s*-anchorPointInPoints_.y;
                y += s*-anchorPointInPoints_.x + c*-anchorPointInPoints_.y;
        }

        // Rot, Translate Matrix
        transform_ = CGAffineTransformMake( c,  s,
                                                                   -s,  c,
                                                                   x,   y );

        return transform_;
}



@end
```

```
//
//  PongScene.h
//  LeapPuzz
//
//  Created by cj on 2/12/13.
//
//

#import "cocos2d.h"
#import "Box2D.h"
#import "GLES-Render.h"
#import "LeapObjectiveC.h"
#import "RedDot.h"

@interface PongScene : CCLayer <LeapDelegate> {

    LeapController *controller;

        CCTexture2D *spriteTexture_;    // weak ref
        b2World* world;                                // strong ref
        GLESDebugDraw *m_debugDraw;             // strong ref

    CCSprite* targetSprite;
    b2MouseJoint *_mouseJoint;
    b2World* _world;
    b2Body *_groundBody;

    NSMutableDictionary* trackableList;
}

@end
```

```
//
//  PongScene.m
//  LeapPuzz
//
//  Created by cj on 2/12/13.
//
//

#import "PongScene.h"

@implementation PongScene

@end
```