```
 1: //
 2: //  AppDelegate.h
 3: //  PhysicsFingerPaint
 4: //
 5: //  Created by cj on 5/8/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import <Cocoa/Cocoa.h>
10:
11: @interface AppDelegate : NSObject <NSApplicationDelegate>
12:
13: @property (assign) IBOutlet NSWindow *window;
14:
15: @end
```

```
 1: //
 2: //  AppDelegate.m
 3: //  PhysicsFingerPaint
 4: //
 5: //  Created by cj on 5/8/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import "AppDelegate.h"
10:
11: @implementation AppDelegate
12:
13: - (void)applicationDidFinishLaunching:(NSNotification *)aNotification
14: {
15:     // Insert code here to initialize your application
16: }
17:
18: @end
```

```
 1: //
 2: //  DrawScene.h
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 2/20/13.
 6: //
 7: //
 8:
 9: #import <Foundation/Foundation.h>
10: #import "cocos2d.h"
11: #import "Box2D.h"
12: #import "GLES-Render.h"
13: #import "LeapObjectiveC.h"
14: #import "RedDot.h"
15: @interface DrawScene : CCLayer{
16:     LeapController *controller;
17:
18:         CCTexture2D *spriteTexture_;    // weak ref
19:         b2World* world;                                 // strong ref
20:         GLESDebugDraw *m_debugDraw;             // strong ref
21:
22:     CCSprite* targetSprite;
23:     b2MouseJoint *_mouseJoint;
24:     b2World* _world;
25:     b2Body *_groundBody;
26:
27:
28:     NSMutableDictionary* trackableList;
29: }
30:
31: @end
```

```
    1: //
    2: //  DrawScene.m
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 2/20/13.
    6: //
    7: //
    8:
    9: #import "DrawScene.h"
   10: #import "SimplePointObject.h"
   11:
   12: #define PTM_RATIO 32
   13:
   14: enum {
   15:     kTagParentNode = 1,
   16: };
   17: @implementation DrawScene
   18:
   19: -(id) init
   20: {
   21:         if( (self=[super init])) {
   22:
   23:                 // enable events
   24:
   25: #ifdef __IPHONE_OS_VERSION_MAX_ALLOWED
   26:                 self.isTouchEnabled = YES;
   27:                 self.isAccelerometerEnabled = YES;
   28: #elif defined(__MAC_OS_X_VERSION_MAX_ALLOWED)
   29:                 self.isMouseEnabled = YES;
   30: #endif
   31:                 CGSize s = [CCDirector sharedDirector].winSize;
   32:
   33:                 // init physics
   34:                 [self initPhysics];
   35:
   36:                 // create reset button
   37:                 [self createResetButton];
   38:
   39:                 //Set up sprite
   40:
   41: #if 1
   42:                 // Use batch node. Faster
   43:                 CCSpriteBatchNode *parent = [CCSpriteBatchNode batchNodeWithFile:@"blocks.png" capacit
y:100];
   44:                 spriteTexture_ = [parent texture];
   45: #else
   46:                 // doesn't use batch node. Slower
   47:                 spriteTexture_ = [[CCTextureCache sharedTextureCache] addImage:@"blocks.png"];
   48:                 CCNode *parent = [CCNode node];
   49: #endif
   50:                 [self addChild:parent z:0 tag:kTagParentNode];
   51:
   52:                 CCLabelTTF *label = [CCLabelTTF labelWithString:@"LeapPuzz" fontName:@"Marker Felt" fo
ntSize:32];
   53:                 [self addChild:label z:0];
   54:                 [label setColor:ccc3(0,0,255)];
   55:                 label.position = ccp( s.width/2, s.height-50);
   56:
   57:                 [self scheduleUpdate];
   58:
   59:         trackableList = [[NSMutableDictionary alloc] init];
   60:
   61:
   62:         }
   63:         return self;
   64: }
   65:
   66: #pragma mark -
   67:
   68: -(void) createResetButton
   69: {
   70:         CCMenuItemLabel *reset = [CCMenuItemFont itemWithString:@"Reset" block:^(id sender){
   71:                 CCScene *s = [CCScene node];
   72:                 id child = [DrawScene node];
   73:                 [s addChild:child];
   74:                 [[CCDirector sharedDirector] replaceScene: s];
   75:         }];
   76:
   77:         CCMenu *menu = [CCMenu menuWithItems:reset, nil];
   78:
```

```
79:         CGSize s = [[CCDirector sharedDirector] winSize];
80:
81:         menu.position = ccp(s.width/2, 30);
82:         [self addChild: menu z:-1];
83:
84: }
85:
86: -(void) initPhysics
87: {
88:
89:         CGSize s = [[CCDirector sharedDirector] winSize];
90:
91:     //Gravity
92:         b2Vec2 gravity;
93:         gravity.Set(0.0f, -10.0f);
94:         world = new b2World(gravity);
95:
96:
97:         // Do we want to let bodies sleep?
98:         world->SetAllowSleeping(true);
99:
100:        world->SetContinuousPhysics(true);
101:
102:        m_debugDraw = new GLESDebugDraw( PTM_RATIO );
103:        world->SetDebugDraw(m_debugDraw);
104:
105:    _world = world;
106:
107:        uint32 flags = 0;
108:        flags += b2Draw::e_shapeBit;
109:        //              flags += b2Draw::e_jointBit;
110:        //              flags += b2Draw::e_aabbBit;
111:        //              flags += b2Draw::e_pairBit;
112:        //              flags += b2Draw::e_centerOfMassBit;
113:        m_debugDraw->SetFlags(flags);
114:
115:
116:        // Define the ground body.
117:        b2BodyDef groundBodyDef;
118:        groundBodyDef.position.Set(0, 0); // bottom-left corner
119:
120:        // Call the body factory which allocates memory for the ground body
121:        // from a pool and creates the ground box shape (also from a pool).
122:        // The body is also added to the world.
123:        b2Body* groundBody = world->CreateBody(&groundBodyDef);
124:
125:        // Define the ground box shape.
126:        b2EdgeShape groundBox;
127:
128:        // bottom
129:
130:        groundBox.Set(b2Vec2(0,0), b2Vec2(s.width/PTM_RATIO,0));
131:        groundBody->CreateFixture(&groundBox,0);
132:
133:        // top
134:        groundBox.Set(b2Vec2(0,s.height/PTM_RATIO), b2Vec2(s.width/PTM_RATIO,s.height/PTM_RATIO));
135:        groundBody->CreateFixture(&groundBox,0);
136:
137:        // left
138:        groundBox.Set(b2Vec2(0,s.height/PTM_RATIO), b2Vec2(0,0));
139:        groundBody->CreateFixture(&groundBox,0);
140:
141:        // right
142:        groundBox.Set(b2Vec2(s.width/PTM_RATIO,s.height/PTM_RATIO), b2Vec2(s.width/PTM_RATIO,0));
143:        groundBody->CreateFixture(&groundBox,0);
144:
145:    _groundBody = groundBody;
146: }
147:
148:
149:
150: @end
```

```
 1: //
 2: //  FingerPaintingScene.h
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 2/18/13.
 6: //
 7: //
 8:
 9: #import <Foundation/Foundation.h>
10: #import "cocos2d.h"
11: #import "Box2D.h"
12: #import "GLES-Render.h"
13: #import "LeapObjectiveC.h"
14: #import "RedDot.h"
15:
16:
17: @interface FingerPaintingScene : CCLayer <LeapListener> {
18:
19:     LeapController *controller;
20:
21:         CCTexture2D *spriteTexture_;    // weak ref
22:         b2World* world;                                 // strong ref
23:         GLESDebugDraw *m_debugDraw;             // strong ref
24:
25:     CCSprite* targetSprite;
26:     b2MouseJoint *_mouseJoint;
27:     b2World* _world;
28:     b2Body *_groundBody;
29:
30:     CIColor* brushColor;
31:
32:
33:     NSMutableDictionary* trackableList;
34:     NSMutableDictionary* brushesList;
35:
36:     NSTimer* updateDraw;
37:
38:
39:     RedDot* mouseCursor;
40:
41:
42:
43: }
44:
45:
46:
47: @end
```

```
    1: //
    2: //  FingerPaintingScene.m
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 2/18/13.
    6: //
    7: //
    8:
    9: #import "FingerPaintingScene.h"
   10: #import "SimplePoint.h"
   11:
   12: #define PTM_RATIO 32
   13:
   14: enum {
   15:         kTagParentNode = 1,
   16: };
   17:
   18:
   19:
   20: @implementation FingerPaintingScene
   21:
   22: -(id) init
   23: {
   24:         if( (self=[super init])) {
   25:
   26:                 // enable events
   27:
   28:                 self.isMouseEnabled = YES;
   29:
   30:                 CGSize s = [CCDirector sharedDirector].winSize;
   31:
   32:
   33:                 // create reset button
   34:                 [self createResetButton];
   35:
   36:                 //Set up sprite
   37:
   38: #if 1
   39:                 // Use batch node. Faster
   40:                 CCSpriteBatchNode *parent = [CCSpriteBatchNode batchNodeWithFile:@"blocks.png" capacit
y:100];
   41:                 spriteTexture_ = [parent texture];
   42: #else
   43:                 // doesn't use batch node. Slower
   44:                 spriteTexture_ = [[CCTextureCache sharedTextureCache] addImage:@"blocks.png"];
   45:                 CCNode *parent = [CCNode node];
   46: #endif
   47:                 [self addChild:parent z:0 tag:kTagParentNode];
   48:
   49:
   50:                 CCLabelTTF *label = [CCLabelTTF labelWithString:@"MotionStreak" fontName:@"Marker Felt
" fontSize:32];
   51:                 [self addChild:label z:0];
   52:                 [label setColor:ccc3(0,0,255)];
   53:                 label.position = ccp( s.width/2, s.height-50);
   54:
   55:                 //[self scheduleUpdate];
   56:
   57:
   58:         updateDraw = [NSTimer scheduledTimerWithTimeInterval:0.1
   59:                                       target:self
   60:                                     selector:@selector(updateDrawing:)
   61:                                     userInfo:nil
   62:                                      repeats:YES];
   63:
   64:         trackableList = [[NSMutableDictionary alloc] init];
   65:         brushesList = [[NSMutableDictionary alloc] init];
   66:
   67:         [self run];
   68:
   69:
   70:
   71:         }
   72:         return self;
   73: }
   74:
   75: - (void)updateDrawing:(id)sender{
   76:
   77:
   78:     glLineWidth(5.f);
```

```
    ccDrawColor4B(0, 0, 255, 255);

    glEnable(GL_LINE_SMOOTH);


    for (id key in [trackableList allKeys]) {
        RedDot* sprite = [trackableList objectForKey:key];
        if ([sprite.path count] > 1){
            for (int i = 0;  i < [sprite.path count] -1; i++){
                SimplePoint* simplePoint =  [sprite.path objectAtIndex:i];
                SimplePoint* simplePointNext = [sprite.path objectAtIndex:i+1];

                NSLog(@"Dragged %0.0f , %0.0f ", simplePoint.x, simplePoint.y);
                ccDrawLine( ccp(simplePoint.x, simplePoint.y), ccp(simplePointNext.x, simplePoint.y));


                NSLog(@"Draw");
            }
            [sprite.path removeAllObjects];
        }

    }


    if (mouseCursor != nil){
        if ([mouseCursor.path count] > 1){
            for (int i = 0;  i < [mouseCursor.path count] -1; i++){
                SimplePoint* simplePoint =  [mouseCursor.path objectAtIndex:i];
                SimplePoint* simplePointNext = [mouseCursor.path objectAtIndex:i+1];

                NSLog(@"Dragged %0.0f , %0.0f ", simplePoint.x, simplePoint.y);
                ccDrawLine( ccp(simplePoint.x, simplePoint.y), ccp(simplePointNext.x, simplePoint.y));


                NSLog(@"Draw");
            }
            NSRange range = NSMakeRange(0, [mouseCursor.path count] -1);

            [mouseCursor.path removeObjectsInRange:range];

        }

    }

}


#pragma mark - SampleDelegate Callbacks
- (void)run
{
    controller = [[LeapController alloc] init];
    [controller addListener:self];
    NSLog(@"running");
}

- (void)onInit:(NSNotification *)notification{
    NSLog(@"Leap: Initialized");
}

- (void)onConnect:(NSNotification *)notification;
{
    NSLog(@"Leap: Connected");
    LeapController *aController = (LeapController *)[notification object];
    [aController enableGesture:LEAP_GESTURE_TYPE_CIRCLE enable:YES];
    [aController enableGesture:LEAP_GESTURE_TYPE_KEY_TAP enable:YES];
    [aController enableGesture:LEAP_GESTURE_TYPE_SCREEN_TAP enable:YES];
    [aController enableGesture:LEAP_GESTURE_TYPE_SWIPE enable:YES];
}

- (void)onDisconnect:(NSNotification *)notification{
    NSLog(@"Leap: Disconnected");
}

- (void)onExit:(NSNotification *)notification{
    NSLog(@"Leap: Exited");
}
- (void)onFrame:(NSNotification *)notification{

    ///NSLog(@"OnFrame");
    LeapController *aController = (LeapController *)[notification object];
```

```
159:     // Get the most recent frame and report some basic information
160:     LeapFrame *frame = [aController frame:0];
161:
162:
163:     /*
164:      NSLog(@"Frame id: %lld, timestamp: %lld, hands: %ld, fingers: %ld, tools: %ld",
165:      [frame id], [frame timestamp], [[frame hands] count],
166:      [[frame fingers] count], [[frame tools] count]);
167:      */
168:     if ([[frame hands] count] != 0) {
169:         // Get the first hand
170:         LeapHand *hand = [[frame hands] objectAtIndex:0];
171:
172:         // Check if the hand has any fingers
173:         NSArray *fingers = [hand fingers];
174:
175:         if ([fingers count] != 0) {
176:
177:             // Calculate the hand's average finger tip position
178:             LeapVector *avgPos = [[LeapVector alloc] init];
179:             for (int i = 0; i < [fingers count]; i++) {
180:                 LeapFinger *finger = [fingers objectAtIndex:i];
181:                 avgPos = [avgPos plus:[finger tipPosition]];
182:
183:                 if (avgPos.z < 0){
184:                     NSString* fingerID = [NSString stringWithFormat:@"%d", finger.id];
185:
186:                     //Check if the Finger ID exists in the list already
187:                     if ([trackableList objectForKey:fingerID]) {
188:
189:                         //If it does exist update the position on the screen
190:                         RedDot* sprite = [trackableList objectForKey:fingerID];
191:                         sprite.position = [self covertLeapCoordinates:CGPointMake(finger.tipPosition.x
, finger.tipPosition.y)];
192:                         sprite.updated = TRUE;
193:
194:                         CCMotionStreak* streak = [self getMotionStreak:[sprite.fingerID intValue] with
Sprite:sprite];
195:                         streak.position = sprite.position;
196:
197:                         SimplePoint* simplePoint = [[SimplePoint alloc] initWithPosition:sprite.positi
on];
198:                         [sprite.path addObject:simplePoint];
199:
200:
201:                     }else{
202:
203:                         NSLog(@"x %0.0f y %0.0f z %0.0f", finger.tipPosition.x, finger.tipPosition.y,
finger.tipPosition.z);
204:                         // CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
205:                         //CGPoint mouseLocation = [self convertToNodeSpace:point];
206:
207:                         //Add it to the dictionary
208:                         RedDot* redDot = [self addRedDot:[self covertLeapCoordinates:CGPointMake(finge
r.tipPosition.x, finger.tipPosition.y)] finger:fingerID];
209:                         [trackableList setObject:redDot forKey:fingerID];
210:
211:                         //NSMutableArray* myDraw = [[NSMutableArray alloc]init];
212:
213:                     }
214:                 }
215:
216:             }
217:
218:             avgPos = [avgPos divide:[fingers count]];
219:
220:             //NSLog(@"Hand has %ld fingers, average finger tip position %@", [fingers count], avgPos);
221:             for (LeapFinger* finger in fingers){
222:
223:                 //NSLog(@"Finger ID %d %ld", finger.id, (unsigned long)[finger hash]);
224:             }
225:
226:         }
227:
228:         //
229:         [self checkFingerExists];
230:
231:         // Get the hand's sphere radius and palm position
232:         /*
233:          NSLog(@"Hand sphere radius: %f mm, palm position: %@",
```

```
234:          [hand sphereRadius], [hand palmPosition]);
235:          */
236:         // Get the hand's normal vector and direction
237:         //const LeapVector *normal = [hand palmNormal];
238:         //const LeapVector *direction = [hand direction];
239:
240:         /*
241:          // Calculate the hand's pitch, roll, and yaw angles
242:          NSLog(@"Hand pitch: %f degrees, roll: %f degrees, yaw: %f degrees\n",
243:          [direction pitch] * LEAP_RAD_TO_DEG,
244:          [normal roll] * LEAP_RAD_TO_DEG,
245:          [direction yaw] * LEAP_RAD_TO_DEG);
246:          */
247:     }
248:     /*
249:     NSArray *gestures = [frame gestures:nil];
250:     for (int g = 0; g < [gestures count]; g++) {
251:         LeapGesture *gesture = [gestures objectAtIndex:g];
252:         switch (gesture.type) {
253:             case LEAP_GESTURE_TYPE_CIRCLE: {
254:                 LeapCircleGesture *circleGesture = (LeapCircleGesture *)gesture;
255:                 // Calculate the angle swept since the last frame
256:                 float sweptAngle = 0;
257:                 if(circleGesture.state != LEAP_GESTURE_STATE_START) {
258:                     LeapCircleGesture *previousUpdate = (LeapCircleGesture *)[[aController frame:1] ge
sture:gesture.id];
259:                     sweptAngle = (circleGesture.progress - previousUpdate.progress) * 2 * LEAP_PI;
260:                 }
261:
262:                 NSLog(@"Circle id: %d, %@, progress: %f, radius %f, angle: %f degrees",
263:                       circleGesture.id, [FingerPaintingScene stringForState:gesture.state],
264:                       circleGesture.progress, circleGesture.radius, sweptAngle * LEAP_RAD_TO_DEG);
265:                 break;
266:             }
267:             case LEAP_GESTURE_TYPE_SWIPE: {
268:                 LeapSwipeGesture *swipeGesture = (LeapSwipeGesture *)gesture;
269:                 NSLog(@"Swipe id: %d, %@, position: %@, direction: %@, speed: %f",
270:                       swipeGesture.id, [FingerPaintingScene stringForState:swipeGesture.state],
271:                       swipeGesture.position, swipeGesture.direction, swipeGesture.speed);
272:                 break;
273:             }
274:             case LEAP_GESTURE_TYPE_KEY_TAP: {
275:                 LeapKeyTapGesture *keyTapGesture = (LeapKeyTapGesture *)gesture;
276:                 NSLog(@"Key Tap id: %d, %@, position: %@, direction: %@",
277:                       keyTapGesture.id, [FingerPaintingScene stringForState:keyTapGesture.state],
278:                       keyTapGesture.position, keyTapGesture.direction);
279:                 break;
280:             }
281:             case LEAP_GESTURE_TYPE_SCREEN_TAP: {
282:                 LeapScreenTapGesture *screenTapGesture = (LeapScreenTapGesture *)gesture;
283:                 NSLog(@"Screen Tap id: %d, %@, position: %@, direction: %@",
284:                       screenTapGesture.id, [FingerPaintingScene stringForState:screenTapGesture.state]
,
285:                       screenTapGesture.position, screenTapGesture.direction);
286:                 break;
287:             }
288:             default:
289:                 NSLog(@"Unknown gesture type");
290:                 break;
291:         }
292:     }
293:      */
294:
295: }
296:
297: //Cycle through all the trackable dots and check if the fingers still exist.
298: //If they don't, delete them.
299: - (void)checkFingerExists{
300:
301:     for (id key in [trackableList allKeys]) {
302:         RedDot* sprite = [trackableList objectForKey:key];
303:         if (sprite.updated) {
304:             sprite.updated = FALSE;
305:             //return;
306:         }else{
307:             CCNode *parent = [self getChildByTag:kTagParentNode];
308:             [trackableList removeObjectForKey:key];
309:             [parent removeChild:sprite cleanup:YES];
310:             //Get rid of the motion streak
311:             [self removeMotionStreak:[sprite.fingerID intValue]];
```

```
  312:
  313:         }
  314:     }
  315: }
  316:
  317:
  318: - (RedDot*)addRedDot:(CGPoint)p finger:(NSString*)fingerID{
  319:
  320:     CCNode *parent = [self getChildByTag:kTagParentNode];
  321:     int idx = (CCRANDOM_0_1() > .5 ? 0:1);
  322:         int idy = (CCRANDOM_0_1() > .5 ? 0:1);
  323:
  324:         //RedDot *sprite = [RedDot spriteWithFile:@"redcrosshair.png"];
  325:     RedDot *sprite = [RedDot spriteWithTexture:spriteTexture_ rect:CGRectMake(32 * idx,32 * idy,32,32)
];
  326:         [parent addChild:sprite];
  327:     sprite.updated = TRUE;
  328:     sprite.fingerID = fingerID;
  329:     sprite.position = ccp( p.x, p.y);
  330:     sprite.path = [[NSMutableArray alloc] init];
  331:     SimplePoint* simplePoint = [[SimplePoint alloc] initWithPosition:sprite.position];
  332:     [sprite.path addObject:simplePoint];
  333:
  334:     [self createMotionStreak:[sprite.fingerID intValue] withSprite:sprite];
  335:
  336:     return sprite;
  337: }
  338:
  339: - (CGPoint)covertLeapCoordinates:(CGPoint)p{
  340:
  341:     CGSize s = [[CCDirector sharedDirector] winSize];
  342:     float screenCenter = 0.0f;
  343:     float xScale = 1.75f;
  344:     float yScale = 1.25f;
  345:     return CGPointMake((s.width/2)+ (( p.x - screenCenter) * xScale), p.y * yScale);
  346: }
  347: #pragma mark -
  348:
  349: -(void) createResetButton
  350: {
  351:         CCMenuItemLabel *reset = [CCMenuItemFont itemWithString:@"Reset" block:^(id sender){
  352:                 CCScene *s = [CCScene node];
  353:                 id child = [FingerPaintingScene node];
  354:                 [s addChild:child];
  355:                 [[CCDirector sharedDirector] replaceScene: s];
  356:         }];
  357:
  358:         CCMenu *menu = [CCMenu menuWithItems:reset, nil];
  359:
  360:         CGSize s = [[CCDirector sharedDirector] winSize];
  361:
  362:         menu.position = ccp(s.width/2, 30);
  363:         [self addChild: menu z:-1];
  364:
  365: }
  366:
  367: - (void)createMotionStreak:(NSInteger)touchHash withSprite:(CCSprite*)sprite
  368: {
  369:     CCMotionStreak* streak = [CCMotionStreak streakWithFade:1.7f minSeg:10 width:32 color:ccc3(0, 255,
 255) texture:sprite.texture];
  370:     [self addChild:streak z:5 tag:touchHash];
  371: }
  372:
  373: - (void)removeMotionStreak:(NSInteger)touchHash
  374: {
  375:     [self removeChildByTag:touchHash cleanup:YES];
  376: }
  377:
  378: - (CCMotionStreak*)getMotionStreak:(NSInteger)touchHash withSprite:(CCSprite*)sprite
  379: {
  380:     CCNode* node = [self getChildByTag:touchHash];
  381:     if(![node isKindOfClass:[CCMotionStreak class]]) {
  382:         [self createMotionStreak:touchHash withSprite:sprite];
  383:     }
  384:     return (CCMotionStreak*)node;
  385: }
  386:
  387: - (void)addMotionStreakPoint:(CGPoint)point on:(NSInteger)touchHash withSprite:(CCSprite*)sprite
  388: {
  389:     CCMotionStreak* streak = [self getMotionStreak:touchHash withSprite:sprite];
```

```
390:     streak.position = point;
391:     //[streak.ribbon addPointAt:point width:32];
392: }
393:
394: #pragma mark - Gestures
395:
396: - (void)activateColorWheel{
397:
398: }
399:
400: - (void)deactivateColorWheel{
401:
402: }
403:
404:
405: - (void)draw {
406:     // ...
407:     [super draw];
408:
409:     // draw a simple line
410:     // The default state is:
411:     // Line Width: 1
412:     // color: 255,255,255,255 (white, non-transparent)
413:     // Anti-Aliased
414:     glLineWidth(5.f);
415:     ccDrawColor4B(0, 0, 255, 255);
416:
417:     glEnable(GL_LINE_SMOOTH);
418:
419:
420:     for (id key in [trackableList allKeys]) {
421:         RedDot* sprite = [trackableList objectForKey:key];
422:         if ([sprite.path count] > 1){
423:             for (int i = 0;  i < [sprite.path count] -1; i++){
424:                 SimplePoint* simplePoint =  [sprite.path objectAtIndex:i];
425:                 SimplePoint* simplePointNext = [sprite.path objectAtIndex:i+1];
426:
427:                 NSLog(@"Dragged %0.0f , %0.0f ", simplePoint.x, simplePoint.y);
428:                 ccDrawLine( ccp(simplePoint.x, simplePoint.y), ccp(simplePointNext.x, simplePoint.y));
429:
430:
431:                 NSLog(@"Draw");
432:             }
433:             [sprite.path removeAllObjects];
434:         }
435:
436:     }
437:
438:     // ...
439: }
440:
441:
442:
443:
444: + (NSString *)stringForState:(LeapGestureState)state
445: {
446:     switch (state) {
447:         case LEAP_GESTURE_STATE_INVALID:
448:             return @"STATE_INVALID";
449:         case LEAP_GESTURE_STATE_START:
450:             return @"STATE_START";
451:         case LEAP_GESTURE_STATE_UPDATE:
452:             return @"STATE_UPDATED";
453:         case LEAP_GESTURE_STATE_STOP:
454:             return @"STATE_STOP";
455:         default:
456:             return @"STATE_INVALID";
457:     }
458: }
459:
460:
461:
462: - (void)beginFingerDraw:(id)sender{
463:
464:     //TrackedFinger* trackedFinger = (TrackedFinger*)[sender object];
465:     //[self beginDraw:trackedFinger.position];
466:
467: }
468:
469: - (void)updateFingerDraw:(id)sender{
```

```
470:     //TrackedFinger* trackedFinger = (TrackedFinger*)[sender object];
471:     //[self updateDraw:trackedFinger.position];
472:
473: }
474:
475: - (void)endFingerDraw:(id)sender{
476:     //TrackedFinger* trackedFinger = (TrackedFinger*)[sender object];
477:     //[self endDraw:trackedFinger.position];
478: }
479:
480: //The further negative, the thicker the line.
481: - (void)beginDraw:(CGPoint)point{
482:
483:
484:
485: }
486:
487: - (void)updateDraw:(CGPoint)point{
488:
489:
490: }
491:
492: - (void)endDraw:(CGPoint)point{
493:
494: }
495:
496:
497: #pragma mark - Mouse Handling
498: - (BOOL)ccMouseDown:(NSEvent *)event{
499:
500:     if (mouseCursor == nil){
501:
502:
503:         CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
504:         mouseCursor = [self addRedDot:point finger:@"mouse"];
505:
506:     }
507:
508:
509:         return YES;
510: }
511:
512: - (BOOL)ccMouseDragged:(NSEvent *)event {
513:
514:
515:     if (mouseCursor != nil){
516:         CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
517:         mouseCursor.position = point;
518:         SimplePoint* simplePoint = [[SimplePoint alloc] initWithPosition:point];
519:         [mouseCursor.path addObject:simplePoint];
520:
521:     }
522:
523:     return YES;
524:
525: }
526:
527: - (BOOL)ccMouseUp:(NSEvent *)event{
528:
529:     if (mouseCursor != nil){
530:         CCNode *parent = [self getChildByTag:kTagParentNode];
531:         [parent removeChild:mouseCursor cleanup:nil];
532:         mouseCursor =nil;
533:     }
534:
535:
536:
537:     return YES;
538: }
539:
540:
541: @end
```

```
 1: //
 2: //  GeometryDrawScene.h
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 2/19/13.
 6: //
 7: //
 8:
 9: #import <Foundation/Foundation.h>
10: #import "cocos2d.h"
11: #import "Box2D.h"
12: #import "GLES-Render.h"
13: #import "LeapObjectiveC.h"
14: #import "RedDot.h"
15:
16: @interface GeometryDrawScene : CCLayer{
17:
18:     LeapController *controller;
19:
20:         CCTexture2D *spriteTexture_;    // weak ref
21:         b2World* world;                                 // strong ref
22:         GLESDebugDraw *m_debugDraw;             // strong ref
23:
24:     CCSprite* targetSprite;
25:     b2MouseJoint *_mouseJoint;
26:     b2World* _world;
27:     b2Body *_groundBody;
28:
29: //
30:     CCRenderTexture *target;
31:     CCSprite *brush;
32:
33:     CGPoint previousLocation;
34:         b2Body* currentPlatformBody;
35:
36:         //std::vector<cocos2d::CCPoint> plataformPoints;
37:     NSMutableArray* plataformPoints;
38:
39:
40:     NSMutableDictionary* trackableList;
41: }
42:
43: @end
```

```
    1: //
    2: //  GeometryDrawScene.m
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 2/19/13.
    6: //
    7: //
    8:
    9: #import "GeometryDrawScene.h"
   10: #import "SimplePointObject.h"
   11:
   12: #define PTM_RATIO 32
   13:
   14: enum {
   15:     kTagParentNode = 1,
   16: };
   17:
   18: @implementation GeometryDrawScene
   19: -(id) init
   20: {
   21:         if( (self=[super init])) {
   22:
   23:                 // enable events
   24:
   25: #ifdef __IPHONE_OS_VERSION_MAX_ALLOWED
   26:                 self.isTouchEnabled = YES;
   27:                 self.isAccelerometerEnabled = YES;
   28: #elif defined(__MAC_OS_X_VERSION_MAX_ALLOWED)
   29:                 self.isMouseEnabled = YES;
   30: #endif
   31:                 CGSize s = [CCDirector sharedDirector].winSize;
   32:
   33:                 // init physics
   34:                 [self initPhysics];
   35:
   36:                 // create reset button
   37:                 [self createResetButton];
   38:
   39:                 //Set up sprite
   40:
   41: #if 1
   42:                 // Use batch node. Faster
   43:                 CCSpriteBatchNode *parent = [CCSpriteBatchNode batchNodeWithFile:@"blocks.png" capacit
y:100];
   44:                 spriteTexture_ = [parent texture];
   45: #else
   46:                 // doesn't use batch node. Slower
   47:                 spriteTexture_ = [[CCTextureCache sharedTextureCache] addImage:@"blocks.png"];
   48:                 CCNode *parent = [CCNode node];
   49: #endif
   50:                 [self addChild:parent z:0 tag:kTagParentNode];
   51:
   52:                 plataformPoints = [[NSMutableArray alloc] init];
   53:
   54:
   55:                 CCLabelTTF *label = [CCLabelTTF labelWithString:@"LeapPuzz" fontName:@"Marker Felt" fo
ntSize:32];
   56:                 [self addChild:label z:0];
   57:                 [label setColor:ccc3(0,0,255)];
   58:                 label.position = ccp( s.width/2, s.height-50);
   59:
   60:                 [self scheduleUpdate];
   61:
   62:         trackableList = [[NSMutableDictionary alloc] init];
   63:
   64:
   65:         target = [CCRenderTexture renderTextureWithWidth:s.width height: s.height pixelFormat:kCCTextu
re2DPixelFormat_RGBA8888];
   66:         target.position = ccp(s.width / 2, s.height / 2);
   67:         [self addChild:target];
   68:
   69:
   70:
   71:         brush = [CCSprite spriteWithFile:@"largeBrush.png"];
   72:
   73:
   74:
   75:         }
   76:         return self;
   77: }
```

```
78:
79:
80: #pragma mark -
81:
82: -(void) createResetButton
83: {
84:         CCMenuItemLabel *reset = [CCMenuItemFont itemWithString:@"Reset" block:^(id sender){
85:                 CCScene *s = [CCScene node];
86:                 id child = [GeometryDrawScene node];
87:                 [s addChild:child];
88:                 [[CCDirector sharedDirector] replaceScene: s];
89:         }];
90:
91:         CCMenu *menu = [CCMenu menuWithItems:reset, nil];
92:
93:         CGSize s = [[CCDirector sharedDirector] winSize];
94:
95:         menu.position = ccp(s.width/2, 30);
96:         [self addChild: menu z:-1];
97:
98: }
99:
100: -(void) initPhysics
101: {
102:
103:         CGSize s = [[CCDirector sharedDirector] winSize];
104:
105:     //Gravity
106:         b2Vec2 gravity;
107:         gravity.Set(0.0f, -10.0f);
108:         world = new b2World(gravity);
109:
110:
111:         // Do we want to let bodies sleep?
112:         world->SetAllowSleeping(true);
113:
114:         world->SetContinuousPhysics(true);
115:
116:         m_debugDraw = new GLESDebugDraw( PTM_RATIO );
117:         world->SetDebugDraw(m_debugDraw);
118:
119:     _world = world;
120:
121:         uint32 flags = 0;
122:         flags += b2Draw::e_shapeBit;
123:         //              flags += b2Draw::e_jointBit;
124:         //              flags += b2Draw::e_aabbBit;
125:         //              flags += b2Draw::e_pairBit;
126:         //              flags += b2Draw::e_centerOfMassBit;
127:         m_debugDraw->SetFlags(flags);
128:
129:
130:         // Define the ground body.
131:         b2BodyDef groundBodyDef;
132:         groundBodyDef.position.Set(0, 0); // bottom-left corner
133:
134:         // Call the body factory which allocates memory for the ground body
135:         // from a pool and creates the ground box shape (also from a pool).
136:         // The body is also added to the world.
137:         b2Body* groundBody = world->CreateBody(&groundBodyDef);
138:
139:         // Define the ground box shape.
140:         b2EdgeShape groundBox;
141:
142:         // bottom
143:
144:         groundBox.Set(b2Vec2(0,0), b2Vec2(s.width/PTM_RATIO,0));
145:         groundBody->CreateFixture(&groundBox,0);
146:
147:         // top
148:         groundBox.Set(b2Vec2(0,s.height/PTM_RATIO), b2Vec2(s.width/PTM_RATIO,s.height/PTM_RATIO));
149:         groundBody->CreateFixture(&groundBox,0);
150:
151:         // left
152:         groundBox.Set(b2Vec2(0,s.height/PTM_RATIO), b2Vec2(0,0));
153:         groundBody->CreateFixture(&groundBox,0);
154:
155:         // right
156:         groundBox.Set(b2Vec2(s.width/PTM_RATIO,s.height/PTM_RATIO), b2Vec2(s.width/PTM_RATIO,0));
157:         groundBody->CreateFixture(&groundBox,0);
```

```
158:
159:     _groundBody = groundBody;
160: }
161:
162: -(void) draw
163: {
164:         //
165:         // IMPORTANT:
166:         // This is only for debug purposes
167:         // It is recommend to disable it
168:         //
169:         [super draw];
170:
171:         ccGLEnableVertexAttribs( kCCVertexAttribFlag_Position );
172:
173:         kmGLPushMatrix();
174:
175:         world->DrawDebugData();
176:
177:         kmGLPopMatrix();
178:
179: }
180:
181: //void HelloWorld::addRectangleBetweenPointsToBody(b2Body *body, CCPoint start, CCPoint end)
182: - (void)addRectangleBetweenPointsToBody:(b2Body *)body withStart:(CGPoint) start withEnd:(CGPoint)end{
183:
184:
185:     float distance = sqrt( pow(end.x - start.x, 2) + pow(end.y - start.y, 2));
186:
187:     float sx=start.x;
188:     float sy=start.y;
189:     float ex=end.x;
190:     float ey=end.y;
191:     float dist_x=sx-ex;
192:     float dist_y=sy-ey;
193:     float angle= atan2(dist_y,dist_x);
194:
195:     float px = (sx+ex)/2/PTM_RATIO - body->GetPosition().x;
196:     float py = (sy+ey)/2/PTM_RATIO - body->GetPosition().y;
197:
198:     float width = abs(distance)/PTM_RATIO;
199:
200:     float height =  brush.boundingBox.size.height/PTM_RATIO;
201:
202:     b2PolygonShape boxShape;
203:     boxShape.SetAsBox(width / 2, height / 2, b2Vec2(px,py),angle);
204:
205:
206:     b2FixtureDef boxFixtureDef;
207:     boxFixtureDef.shape = &boxShape;
208:     boxFixtureDef.density = 5;
209:
210:     boxFixtureDef.filter.categoryBits = 2;
211:
212:     body->CreateFixture(&boxFixtureDef);
213:
214: }
215:
216:
217: - (void)addRectangleBetweenPointsToDynamicBody:(b2Body *)body withStart:(CGPoint) start withEnd:(CGPoi
nt)end{
218:
219:     float distance = sqrt( pow(end.x - start.x, 2) + pow(end.y - start.y, 2));
220:
221:     float sx=start.x;
222:     float sy=start.y;
223:     float ex=end.x;
224:     float ey=end.y;
225:
226:
227:     float dist_x=abs(sx-ex);
228:     float dist_y=abs(sy-ey);
229:
230:
231:     float angle= atan2(dist_y,dist_x);
232:
233:     float px = (sx+ex)/2/(float)PTM_RATIO - body->GetPosition().x;
234:     float py = (sy+ey)/2/(float)PTM_RATIO - body->GetPosition().y;
235:
236:     float width = abs(distance)/(float)PTM_RATIO;
```

```
237:
238:     float height =  brush.boundingBox.size.height/PTM_RATIO;
239:
240:     b2PolygonShape boxShape;
241:
242:     boxShape.SetAsBox(width / 2, height / 2, b2Vec2(px,py),angle);
243:     b2FixtureDef boxFixtureDef;
244:     boxFixtureDef.shape = &boxShape;
245:     boxFixtureDef.density = 5;
246:
247:     boxFixtureDef.filter.categoryBits = 2;
248:
249:
250:     body->CreateFixture(&boxFixtureDef);
251:
252: }
253:
254: - (CGRect) getBodyRectangle:(b2Body*) body
255: //CCRect HelloWorld::getBodyRectangle(b2Body* body)
256: {
257:     CGSize s = [[CCDirector sharedDirector] winSize];
258:
259:
260:     float minX2 = s.width;
261:     float maxX2 = 0;
262:     float minY2 = s.height;
263:     float maxY2 = 0;
264:
265:     const b2Transform& xf = body->GetTransform();
266:     for (b2Fixture* f = body->GetFixtureList(); f; f = f->GetNext())
267:     {
268:
269:         b2PolygonShape* poly = (b2PolygonShape*)f->GetShape();
270:         int32 vertexCount = poly->m_vertexCount;
271:         b2Assert(vertexCount <= b2_maxPolygonVertices);
272:
273:         for (int32 i = 0; i < vertexCount; ++i)
274:         {
275:             b2Vec2 vertex = b2Mul(xf, poly->m_vertices[i]);
276:
277:
278:             if(vertex.x < minX2)
279:             {
280:                 minX2 = vertex.x;
281:             }
282:
283:             if(vertex.x > maxX2)
284:             {
285:                 maxX2 = vertex.x;
286:             }
287:
288:             if(vertex.y < minY2)
289:             {
290:                 minY2 = vertex.y;
291:             }
292:
293:             if(vertex.y > maxY2)
294:             {
295:                 maxY2 = vertex.y;
296:             }
297:         }
298:     }
299:
300:     maxX2 *= PTM_RATIO;
301:     minX2 *= PTM_RATIO;
302:     maxY2 *= PTM_RATIO;
303:     minY2 *= PTM_RATIO;
304:
305:     float width2 = maxX2 - minX2;
306:     float height2 = maxY2 - minY2;
307:
308:     float remY2 = s.height - maxY2;
309:
310:     return CGRectMake(minX2, remY2, width2, height2);
311: }
312:
313:
314: -(void) update: (ccTime) dt{
315:
316:
```

```
  317:         //It is recommended that a fixed time step is used with Box2D for stability
  318:         //of the simulation, however, we are using a variable time step here.
  319:         //You need to make an informed choice, the following URL is useful
  320:         //http://gafferongames.com/game-physics/fix-your-timestep/
  321:
  322:         int32 velocityIterations = 8;
  323:         int32 positionIterations = 1;
  324:
  325:         // Instruct the world to perform a single step of simulation. It is
  326:         // generally best to keep the time step and iterations fixed.
  327:         world->Step(dt, velocityIterations, positionIterations);
  328:
  329:     //Iterate over the bodies in the physics world
  330:     for (b2Body* b = world->GetBodyList(); b; b = b->GetNext())
  331:     {
  332:         if (b->GetUserData() != Nil) {
  333:
  334:             //Synchronize the AtlasSprites position and rotation with the corresponding body
  335:
  336:             CCSprite* myActor = (__bridge CCSprite*)b->GetUserData();
  337:             myActor.position = CGPointMake( b->GetPosition().x * PTM_RATIO, b->GetPosition().y * PTM_R
ATIO);
  338:             myActor.rotation = ( -1 * CC_RADIANS_TO_DEGREES(b->GetAngle()) );
  339:         }
  340:     }
  341:
  342: }
  343:
  344: #pragma mark - Touch Handling
  345:
  346: - (BOOL) ccMouseDown:(NSEvent *)event{
  347:
  348:     //CGSize s = [[CCDirector sharedDirector] winSize];
  349:
  350:     CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
  351:     CGPoint location = point;
  352:
  353:     //b2Vec2 locationWorld = b2Vec2(location.x/PTM_RATIO, location.y/PTM_RATIO);
  354:
  355:     [plataformPoints removeAllObjects];
  356:
  357:
  358:     SimplePointObject* pointObject = [[SimplePointObject alloc] initWithPosition:location];
  359:     [plataformPoints addObject:pointObject];
  360:
  361:
  362:
  363:     previousLocation = location;
  364:
  365:     b2BodyDef myBodyDef;
  366:     myBodyDef.type = b2_staticBody;
  367:     myBodyDef.position.Set(location.x/PTM_RATIO,location.y/PTM_RATIO);
  368:     currentPlatformBody = world->CreateBody(&myBodyDef);
  369:
  370:     return YES;
  371: }
  372:
  373: - (BOOL)ccMouseDragged:(NSEvent *)event {
  374:
  375:     //CGSize s = [[CCDirector sharedDirector] winSize];
  376:
  377:     CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
  378:     CGPoint location = point;
  379:
  380:     //b2Vec2 locationWorld = b2Vec2(location.x/PTM_RATIO, location.y/PTM_RATIO);
  381:
  382:
  383:
  384:     //CCTouch *touch = (CCTouch *)touches->anyObject();
  385:     CGPoint start = location;
  386:     CGPoint end = previousLocation;
  387:
  388:     [target begin];
  389:
  390:
  391:     float distance = ccpDistance(start, end);
  392:
  393:     for (int i = 0; i < distance; i++)
  394:     {
  395:         float difx = end.x - start.x;
```

```
  396:         float dify = end.y - start.y;
  397:         float delta = (float)i / distance;
  398:         brush.position = ccp(start.x + (difx * delta), start.y + (dify * delta));
  399:
  400:                 //brush->setOpacity(0.1);
  401:
  402:         [brush visit];
  403:     }
  404:     [target end];
  405:
  406:
  407:     distance = sqrt( pow(location.x - previousLocation.x, 2) + pow(location.y - previousLocation.y, 2)
);
  408:
  409:         if(distance > 2)
  410:         {
  411:
  412:
  413:             [self addRectangleBetweenPointsToBody:currentPlatformBody withStart:previousLocation withE
nd:location];
  414:             SimplePointObject* pointObject = [[SimplePointObject alloc] initWithPosition:location];
  415:             [plataformPoints addObject:pointObject];
  416:
  417:             previousLocation = location;
  418:
  419:         }else{
  420:             //NSLog(@"Do Not add");
  421:         }
  422:
  423:
  424:     return YES;
  425:
  426: }
  427:
  428: - (BOOL)ccMouseUp:(NSEvent *)event{
  429:
  430:
  431:     b2BodyDef myBodyDef;
  432:     myBodyDef.type = b2_dynamicBody; //this will be a dynamic body
  433:
  434:     myBodyDef.position.Set(currentPlatformBody->GetPosition().x, currentPlatformBody->GetPosition().y)
; //set the starting position
  435:     myBodyDef.angle = 0;
  436:
  437:     b2Body* newBody = world->CreateBody(&myBodyDef);
  438:
  439:
  440:
  441:     for (int i=0; i < [plataformPoints count] - 1; i++){
  442:
  443:         SimplePointObject* startPoint = [plataformPoints objectAtIndex:i];
  444:         CGPoint start = startPoint.point;
  445:
  446:         SimplePointObject* endPoint = [plataformPoints objectAtIndex:i+1];
  447:         CGPoint end = endPoint.point;
  448:
  449:         [self addRectangleBetweenPointsToDynamicBody:newBody withStart:start withEnd:end];
  450:
  451:     }
  452:
  453:
  454:     world->DestroyBody(currentPlatformBody);
  455:
  456:
  457:     CGSize s = [[CCDirector sharedDirector] winSize];
  458:     CGRect bodyRectangle = [self getBodyRectangle:newBody];
  459:
  460:     CGImage *pImage = [target newCGImage];
  461:     CCTexture2D *tex = [[CCTextureCache sharedTextureCache] addCGImage:pImage forKey:nil];
  462:     CCSprite* sprite = [CCSprite spriteWithTexture:tex rect:bodyRectangle];
  463:
  464:     float anchorX = newBody->GetPosition().x * PTM_RATIO - bodyRectangle.origin.x;
  465:     float anchorY = bodyRectangle.size.height - (s.height - bodyRectangle.origin.y - newBody->GetPosit
ion().y * PTM_RATIO);
  466:
  467:     [sprite setAnchorPoint:ccp(anchorX / bodyRectangle.size.width,  anchorY / bodyRectangle.size.heigh
t)];
  468:
  469:     //myBodyDef.userData =  (__bridge void *)sprite;
  470:     newBody->SetUserData((__bridge void *)sprite);
```

```
  471:
  472:
  473:     [self addChild:sprite];
  474:     [self removeChild:target cleanup:YES];
  475:
  476:     target = [CCRenderTexture renderTextureWithWidth:s.width height:s.height pixelFormat:kCCTexture2DP
ixelFormat_RGBA8888];
  477:     target.position = ccp(s.width / 2, s.height / 2);
  478:     [self addChild:target z:5];
  479:
  480:     return YES;
  481: }
  482:
  483: @end
```

```
    1: /*
    2: * Copyright (c) 2006-2007 Erin Catto http://www.gphysics.com
    3: *
    4: * iPhone port by Simon Oliver - http://www.simonoliver.com - http://www.handcircus.com
    5: *
    6: * This software is provided 'as-is', without any express or implied
    7: * warranty.  In no event will the authors be held liable for any damages
    8: * arising from the use of this software.
    9: * Permission is granted to anyone to use this software for any purpose,
   10: * including commercial applications, and to alter it and redistribute it
   11: * freely, subject to the following restrictions:
   12: * 1. The origin of this software must not be misrepresented; you must not
   13: * claim that you wrote the original software. If you use this software
   14: * in a product, an acknowledgment in the product documentation would be
   15: * appreciated but is not required.
   16: * 2. Altered source versions must be plainly marked as such, and must not be
   17: * misrepresented as being the original software.
   18: * 3. This notice may not be removed or altered from any source distribution.
   19: */
   20:
   21: //
   22: // File modified for cocos2d integration
   23: // http://www.cocos2d-iphone.org
   24: //
   25:
   26: #ifndef GLES_RENDER_H
   27: #define GLES_RENDER_H
   28:
   29: #import "cocos2d.h"
   30:
   31: #ifdef __CC_PLATFORM_IOS
   32: #import <OpenGLES/EAGL.h>
   33: #elif defined(__CC_PLATFORM_MAC)
   34: #import <OpenGL/OpenGL.h>
   35: #endif
   36:
   37: #include "Box2D.h"
   38:
   39: struct b2AABB;
   40:
   41: // This class implements debug drawing callbacks that are invoked
   42: // inside b2World::Step.
   43: class GLESDebugDraw : public b2Draw
   44: {
   45:         float32 mRatio;
   46:         CCGLProgram *mShaderProgram;
   47:         GLint           mColorLocation;
   48:
   49:         void initShader( void );
   50: public:
   51:         GLESDebugDraw();
   52:
   53:         GLESDebugDraw( float32 ratio );
   54:
   55:         void DrawPolygon(const b2Vec2* vertices, int32 vertexCount, const b2Color& color);
   56:
   57:         void DrawSolidPolygon(const b2Vec2* vertices, int32 vertexCount, const b2Color& color);
   58:
   59:         void DrawCircle(const b2Vec2& center, float32 radius, const b2Color& color);
   60:
   61:         void DrawSolidCircle(const b2Vec2& center, float32 radius, const b2Vec2& axis, const b2Color&
color);
   62:
   63:         void DrawSegment(const b2Vec2& p1, const b2Vec2& p2, const b2Color& color);
   64:
   65:         void DrawTransform(const b2Transform& xf);
   66:
   67:     void DrawPoint(const b2Vec2& p, float32 size, const b2Color& color);
   68:
   69:     void DrawString(int x, int y, const char* string, ...);
   70:
   71:     void DrawAABB(b2AABB* aabb, const b2Color& color);
   72: };
   73:
   74:
   75: #endif // GLES_RENDER_H
```

```
    1: /*
    2:  * Copyright (c) 2006-2007 Erin Catto http://www.gphysics.com
    3:  *
    4:  * iPhone port by Simon Oliver - http://www.simonoliver.com - http://www.handcircus.com
    5:  *
    6:  * This software is provided 'as-is', without any express or implied
    7:  * warranty.  In no event will the authors be held liable for any damages
    8:  * arising from the use of this software.
    9:  * Permission is granted to anyone to use this software for any purpose,
   10:  * including commercial applications, and to alter it and redistribute it
   11:  * freely, subject to the following restrictions:
   12:  * 1. The origin of this software must not be misrepresented; you must not
   13:  * claim that you wrote the original software. If you use this software
   14:  * in a product, an acknowledgment in the product documentation would be
   15:  * appreciated but is not required.
   16:  * 2. Altered source versions must be plainly marked as such, and must not be
   17:  * misrepresented as being the original software.
   18:  * 3. This notice may not be removed or altered from any source distribution.
   19:  */
   20:
   21: //
   22: // File modified for cocos2d integration
   23: // http://www.cocos2d-iphone.org
   24: //
   25:
   26: #import "cocos2d.h"
   27: #include "GLES-Render.h"
   28:
   29:
   30: #include <cstdio>
   31: #include <cstdarg>
   32:
   33: #include <cstring>
   34:
   35: GLESDebugDraw::GLESDebugDraw()
   36: : mRatio( 1.0f )
   37: {
   38:         this->initShader();
   39: }
   40:
   41: GLESDebugDraw::GLESDebugDraw( float32 ratio )
   42: : mRatio( ratio )
   43: {
   44:         this->initShader();
   45: }
   46:
   47: void GLESDebugDraw::initShader( void )
   48: {
   49:         mShaderProgram = [[CCShaderCache sharedShaderCache] programForKey:kCCShader_Position_uColor];
   50:
   51:         mColorLocation = glGetUniformLocation( mShaderProgram->program_, "u_color");
   52: }
   53:
   54: void GLESDebugDraw::DrawPolygon(const b2Vec2* old_vertices, int32 vertexCount, const b2Color& color)
   55: {
   56:         [mShaderProgram use];
   57:         [mShaderProgram setUniformForModelViewProjectionMatrix];
   58:
   59:         ccVertex2F vertices[vertexCount];
   60:
   61:         for( int i=0;i<vertexCount;i++) {
   62:                 b2Vec2 tmp = old_vertices[i];
   63:                 tmp *= mRatio;
   64:                 vertices[i].x = tmp.x;
   65:                 vertices[i].y = tmp.y;
   66:         }
   67:
   68:         [mShaderProgram setUniformLocation:mColorLocation withF1:color.r f2:color.g f3:color.b f4:1];
   69:
   70:         glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, vertices);
   71:         glDrawArrays(GL_LINE_LOOP, 0, vertexCount);
   72:
   73:         CC_INCREMENT_GL_DRAWS(1);
   74:
   75:         CHECK_GL_ERROR_DEBUG();
   76: }
   77:
   78: void GLESDebugDraw::DrawSolidPolygon(const b2Vec2* old_vertices, int32 vertexCount, const b2Color& col
or)
   79: {
```

```
   80:         [mShaderProgram use];
   81:         [mShaderProgram setUniformForModelViewProjectionMatrix];
   82:
   83:         ccVertex2F vertices[vertexCount];
   84:
   85:         for( int i=0;i<vertexCount;i++) {
   86:                 b2Vec2 tmp = old_vertices[i];
   87:                 tmp = old_vertices[i];
   88:                 tmp *= mRatio;
   89:                 vertices[i].x = tmp.x;
   90:                 vertices[i].y = tmp.y;
   91:         }
   92:
   93:         [mShaderProgram setUniformLocation:mColorLocation withF1:color.r*0.5f f2:color.g*0.5f f3:color
.b*0.5f f4:0.5f];
   94:
   95:         glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, vertices);
   96:
   97:         glDrawArrays(GL_TRIANGLE_FAN, 0, vertexCount);
   98:
   99:         [mShaderProgram setUniformLocation:mColorLocation withF1:color.r f2:color.g f3:color.b f4:1];
  100:         glDrawArrays(GL_LINE_LOOP, 0, vertexCount);
  101:
  102:         CC_INCREMENT_GL_DRAWS(2);
  103:
  104:         CHECK_GL_ERROR_DEBUG();
  105: }
  106:
  107: void GLESDebugDraw::DrawCircle(const b2Vec2& center, float32 radius, const b2Color& color)
  108: {
  109:         [mShaderProgram use];
  110:         [mShaderProgram setUniformForModelViewProjectionMatrix];
  111:
  112:         const float32 k_segments = 16.0f;
  113:         int vertexCount=16;
  114:         const float32 k_increment = 2.0f * b2_pi / k_segments;
  115:         float32 theta = 0.0f;
  116:
  117:         GLfloat                         glVertices[vertexCount*2];
  118:         for (int32 i = 0; i < k_segments; ++i)
  119:         {
  120:                 b2Vec2 v = center + radius * b2Vec2(cosf(theta), sinf(theta));
  121:                 glVertices[i*2]=v.x * mRatio;
  122:                 glVertices[i*2+1]=v.y * mRatio;
  123:                 theta += k_increment;
  124:         }
  125:
  126:         [mShaderProgram setUniformLocation:mColorLocation withF1:color.r f2:color.g f3:color.b f4:1];
  127:         glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, glVertices);
  128:
  129:         glDrawArrays(GL_LINE_LOOP, 0, vertexCount);
  130:
  131:         CC_INCREMENT_GL_DRAWS(1);
  132:
  133:         CHECK_GL_ERROR_DEBUG();
  134: }
  135:
  136: void GLESDebugDraw::DrawSolidCircle(const b2Vec2& center, float32 radius, const b2Vec2& axis, const b2
Color& color)
  137: {
  138:         [mShaderProgram use];
  139:         [mShaderProgram setUniformForModelViewProjectionMatrix];
  140:
  141:         const float32 k_segments = 16.0f;
  142:         int vertexCount=16;
  143:         const float32 k_increment = 2.0f * b2_pi / k_segments;
  144:         float32 theta = 0.0f;
  145:
  146:         GLfloat                         glVertices[vertexCount*2];
  147:         for (int32 i = 0; i < k_segments; ++i)
  148:         {
  149:                 b2Vec2 v = center + radius * b2Vec2(cosf(theta), sinf(theta));
  150:                 glVertices[i*2]=v.x * mRatio;
  151:                 glVertices[i*2+1]=v.y * mRatio;
  152:                 theta += k_increment;
  153:         }
  154:
  155:
  156:         [mShaderProgram setUniformLocation:mColorLocation withF1:color.r*0.5f f2:color.g*0.5f f3:color
.b*0.5f f4:0.5f];
```

```
157:         glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, glVertices);
158:         glDrawArrays(GL_TRIANGLE_FAN, 0, vertexCount);
159:
160:
161:         [mShaderProgram setUniformLocation:mColorLocation withF1:color.r f2:color.g f3:color.b f4:1];
162:         glDrawArrays(GL_LINE_LOOP, 0, vertexCount);
163:
164:         // Draw the axis line
165:         DrawSegment(center,center+radius*axis,color);
166:
167:         CC_INCREMENT_GL_DRAWS(2);
168:
169:         CHECK_GL_ERROR_DEBUG();
170: }
171:
172: void GLESDebugDraw::DrawSegment(const b2Vec2& p1, const b2Vec2& p2, const b2Color& color)
173: {
174:         [mShaderProgram use];
175:         [mShaderProgram setUniformForModelViewProjectionMatrix];
176:
177:         [mShaderProgram setUniformLocation:mColorLocation withF1:color.r f2:color.g f3:color.b f4:1];
178:
179:         GLfloat                         glVertices[] = {
180:                 p1.x * mRatio, p1.y * mRatio,
181:                 p2.x * mRatio, p2.y * mRatio
182:         };
183:
184:         glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, glVertices);
185:
186:         glDrawArrays(GL_LINES, 0, 2);
187:
188:         CC_INCREMENT_GL_DRAWS(1);
189:
190:         CHECK_GL_ERROR_DEBUG();
191: }
192:
193: void GLESDebugDraw::DrawTransform(const b2Transform& xf)
194: {
195:         b2Vec2 p1 = xf.p, p2;
196:         const float32 k_axisScale = 0.4f;
197:         p2 = p1 + k_axisScale * xf.q.GetXAxis();
198:         DrawSegment(p1, p2, b2Color(1,0,0));
199:
200:         p2 = p1 + k_axisScale * xf.q.GetYAxis();
201:         DrawSegment(p1,p2,b2Color(0,1,0));
202: }
203:
204: void GLESDebugDraw::DrawPoint(const b2Vec2& p, float32 size, const b2Color& color)
205: {
206:         [mShaderProgram use];
207:         [mShaderProgram setUniformForModelViewProjectionMatrix];
208:
209:         [mShaderProgram setUniformLocation:mColorLocation withF1:color.r f2:color.g f3:color.b f4:1];
210:
211: //      glPointSize(size);
212:
213:         GLfloat                         glVertices[] = {
214:                 p.x * mRatio, p.y * mRatio
215:         };
216:
217:         glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, glVertices);
218:
219:         glDrawArrays(GL_POINTS, 0, 1);
220: //      glPointSize(1.0f);
221:
222:         CC_INCREMENT_GL_DRAWS(1);
223:
224:         CHECK_GL_ERROR_DEBUG();
225: }
226:
227: void GLESDebugDraw::DrawString(int x, int y, const char *string, ...)
228: {
229:         //      NSLog(@"DrawString: unsupported: %s", string);
230:         //printf(string);
231:         /* Unsupported as yet. Could replace with bitmap font renderer at a later date */
232: }
233:
234: void GLESDebugDraw::DrawAABB(b2AABB* aabb, const b2Color& color)
235: {
236:         [mShaderProgram use];
```

```
237:         [mShaderProgram setUniformForModelViewProjectionMatrix];
238:
239:         [mShaderProgram setUniformLocation:mColorLocation withF1:color.r f2:color.g f3:color.b f4:1];
240:
241:         GLfloat                         glVertices[] = {
242:                 aabb->lowerBound.x * mRatio, aabb->lowerBound.y * mRatio,
243:                 aabb->upperBound.x * mRatio, aabb->lowerBound.y * mRatio,
244:                 aabb->upperBound.x * mRatio, aabb->upperBound.y * mRatio,
245:                 aabb->lowerBound.x * mRatio, aabb->upperBound.y * mRatio
246:         };
247:
248:         glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, glVertices);
249:         glDrawArrays(GL_LINE_LOOP, 0, 8);
250:
251:         CC_INCREMENT_GL_DRAWS(1);
252:
253:         CHECK_GL_ERROR_DEBUG();
254: }
```

```
    1: //
    2: //  LPTool.h
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 3/29/13.
    6: //
    7: //
    8:
    9: #import <Foundation/Foundation.h>
   10: #import "cocos2d.h"
   11: #import "GLES-Render.h"
   12: /**
   13:  Extends CCSprite object with two properties for tracking sprites with pointable objects
   14:  */
   15:
   16: @interface LPTool : CCSprite
   17:
   18: @property (nonatomic, strong) NSString* toolID; /**< toolID is the ID number assigned by the LeapMotio
n SDK */
   19: @property (nonatomic, readwrite) BOOL updated; /**< updated is if the sprite has been updated in that
frame.*/
   20:
   21:
   22:
   23:
   24: @end
```

```
 1: //
 2: //  LPTool.m
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 3/29/13.
 6: //
 7: //
 8:
 9: #import "LPTool.h"
10:
11: @implementation LPTool
12:
13:
14:
15: @end
```

```
 1: //
 2: //  main.m
 3: //  PhysicsFingerPaint
 4: //
 5: //  Created by cj on 5/8/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import <Cocoa/Cocoa.h>
10:
11: int main(int argc, char *argv[])
12: {
13:     return NSApplicationMain(argc, (const char **)argv);
14: }
```