

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

```
1: //
2: // AppDelegate.m
3: // BreakOut
4: //
5: // Created by cj on 5/7/13.
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:  * 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 "GL ES-Render.h"
28:
29:
30: #include <stdio>
31: #include <stdarg>
32:
33: #include <string>
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          glVertexices[vertexCount*2];
118:     for (int32 i = 0; i < k_segments; ++i)
119:     {
120:         b2Vec2 v = center + radius * b2Vec2(cosf(theta), sinf(theta));
121:         glVertexices[i*2]=v.x * mRatio;
122:         glVertexices[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, glVertexices);
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          glVertexices[vertexCount*2];
147:     for (int32 i = 0; i < k_segments; ++i)
148:     {
149:         b2Vec2 v = center + radius * b2Vec2(cosf(theta), sinf(theta));
150:         glVertexices[i*2]=v.x * mRatio;
151:         glVertexices[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, glVertexices);
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          glVertexices[] = {
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, glVertexices);
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          glVertexices[] = {
214:             p.x * mRatio, p.y * mRatio
215:         };
216:
217:         glVertexAttribPointer(kCCVertexAttrib_Position, 2, GL_FLOAT, GL_FALSE, 0, glVertexices);
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          glVertexices[] = {
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, glVertexices);
249:         glDrawArrays(GL_LINE_LOOP, 0, 8);
250:
251:         CC_INCREMENT_GL_DRAWS(1);
252:
253:         CHECK_GL_ERROR_DEBUG();
254: }
```

```
1: //
2: // HelloWorldLayer.h
3: // LeapPuzz
4: //
5: // Created by cj on 2/3/13.
6: // Copyright __MyCompanyName__ 2013. All rights reserved.
7: //
8:
9:
10: // When you import this file, you import all the cocos2d classes
11: #import "cocos2d.h"
12: #import "Box2D.h"
13: #import "GLes-Render.h"
14: #import "LeapObjectiveC.h"
15: #import "RedDot.h"
16:
17: // HelloWorldLayer
18: @interface HelloWorldLayer : CCLayer <LeapDelegate>
19: {
20:
21:     LeapController *controller;
22:
23:     CCTexture2D *spriteTexture_;    // weak ref
24:     b2World* world;                // strong ref
25:     GLESDebugDraw *m_debugDraw;    // strong ref
26:
27:     CCSprite* targetSprite;
28:     b2MouseJoint *_mouseJoint;
29:     b2World* _world;
30:     b2Body *_groundBody;
31:
32:
33:     NSMutableDictionary* trackableList;
34:
35: }
36: @end
37:
```



```

1: //
2: //  HelloWorldLayer.mm
3: //  LeapPuzz
4: //
5: //  Created by cj on 2/3/13.
6: //  Copyright __MyCompanyName__ 2013. All rights reserved.
7: //
8:
9: // Import the interfaces
10: #import "HelloWorldLayer.h"
11: #import "PhysicsSprite.h"
12: //Pixel to metres ratio. Box2D uses metres as the unit for measurement.
13: //This ratio defines how many pixels correspond to 1 Box2D "metre"
14: //Box2D is optimized for objects of 1x1 metre therefore it makes sense
15: //to define the ratio so that your most common object type is 1x1 metre.
16: #define PTM_RATIO 32
17:
18: enum {
19:     kTagParentNode = 1,
20: };
21:
22:
23:
24: #pragma mark - HelloWorldLayer
25:
26: @interface HelloWorldLayer()
27: -(void) initPhysics;
28: -(void) addNewSpriteAtPosition:(CGPoint)p;
29: -(void) createResetButton;
30: @end
31:
32: @implementation HelloWorldLayer
33:
34: -(id) init
35: {
36:     if( (self=[super init])) {
37:
38:         // enable events
39:
40: #ifdef __IPHONE_OS_VERSION_MAX_ALLOWED
41:         self.isTouchEnabled = YES;
42:         self.isAccelerometerEnabled = YES;
43: #elif defined(__MAC_OS_X_VERSION_MAX_ALLOWED)
44:         self.isMouseEnabled = YES;
45: #endif
46:
47:         CGSize s = [CCDirector sharedDirector].winSize;
48:
49:         // init physics
50:         [self initPhysics];
51:
52:         // create reset button
53:         [self createResetButton];
54:
55:         //Set up sprite
56: #if 1
57:         // Use batch node. Faster
58:         CCSpriteBatchNode *parent = [CCSpriteBatchNode batchNodeWithFile:@"blocks.png" capacit
y:100];
59:         spriteTexture_ = [parent texture];
60: #else
61:         // doesn't use batch node. Slower
62:         spriteTexture_ = [[CCTextureCache sharedTextureCache] addImage:@"blocks.png"];
63:         CCNode *parent = [CCNode node];
64: #endif
65:         [self addChild:parent z:0 tag:kTagParentNode];
66:
67:
68:         [self addNewSpriteAtPosition:ccp(s.width/2, s.height/2)];
69:
70:         CCLabelTTF *label = [CCLabelTTF labelWithString:@"LeapPuzz" fontName:@"Marker Felt" fo
ntSize:32];
71:         [self addChild:label z:0];
72:         [label setColor:ccc3(0,0,255)];
73:         label.position = ccp( s.width/2, s.height-50);
74:
75:         [self scheduleUpdate];
76:
77:         trackableList = [[NSMutableDictionary alloc] init];
78:

```

```

79:         [self run];
80:
81:
82:     }
83:     return self;
84: }
85:
86: - (void)run
87: {
88:     controller = [[LeapController alloc] init];
89:     [controller addDelegate:self];
90:     NSLog(@"running");
91: }
92:
93: #pragma mark - SampleDelegate Callbacks
94:
95: - (void)onInit:(LeapController *)aController
96: {
97:     NSLog(@"Initialized");
98: }
99:
100: - (void)onConnect:(LeapController *)aController
101: {
102:     NSLog(@"Connected");
103: }
104:
105: - (void)onDisconnect:(LeapController *)aController
106: {
107:     NSLog(@"Disconnected");
108: }
109:
110: - (void)onExit:(LeapController *)aController
111: {
112:     NSLog(@"Exited");
113: }
114:
115: - (void)onFrame:(LeapController *)aController
116: {
117:     // Get the most recent frame and report some basic information
118:     LeapFrame *frame = [aController frame:0];
119:     /*
120:     NSLog(@"Frame id: %lld, timestamp: %lld, hands: %ld, fingers: %ld, tools: %ld",
121:           [frame id], [frame timestamp], [[frame hands] count],
122:           [[frame fingers] count], [[frame tools] count]);
123:     */
124:     if ([[frame hands] count] != 0) {
125:         // Get the first hand
126:         LeapHand *hand = [[frame hands] objectAtIndex:0];
127:
128:         // Check if the hand has any fingers
129:         NSArray *fingers = [hand fingers];
130:
131:         if ([fingers count] != 0) {
132:
133:             // Calculate the hand's average finger tip position
134:             LeapVector *avgPos = [[LeapVector alloc] init];
135:             for (int i = 0; i < [fingers count]; i++) {
136:                 LeapFinger *finger = [fingers objectAtIndex:i];
137:                 avgPos = [avgPos plus:[finger tipPosition]];
138:             }
139:
140:             NSString* fingerID = [NSString stringWithFormat:@"%d", finger.id];
141:
142:             //Check if the Finger ID exists in the list already
143:             if ([trackableList objectForKey:fingerID]) {
144:
145:                 //If it does exist update the position on the screen
146:                 RedDot* sprite = [trackableList objectForKey:fingerID];
147:                 sprite.position = [self covertLeapCoordinates:CGPointMake(finger.tipPosition.x, fi
148: nger.tipPosition.y)];
149:                 sprite.updated = TRUE;
150:             }else{
151:                 NSLog(@"x %0.0f y %0.0f z %0.0f", finger.tipPosition.x, finger.tipPosition.y, fing
152 er.tipPosition.z);
153:                 // CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];

```

```

157:         //CGPoint mouseLocation = [self convertToNodeSpace:point];
158:
159:         //Add it to the dictionary
160:         RedDot* redDot = [self addRedDot:CGPointMake(finger.tipPosition.x, finger.tipPosit
ion.y) finger:fingerID];
161:         [trackableList setObject:redDot forKey:fingerID];
162:     }
163: }
164:
165:     avgPos = [avgPos divide:[fingers count]];
166:
167:     //NSLog(@"Hand has %ld fingers, average finger tip position %@", [fingers count], avgPos);
168:     for (LeapFinger* finger in fingers){
169:
170:         //NSLog(@"Finger ID %d %ld", finger.id, (unsigned long)[finger hash]);
171:     }
172: }
173: }
174:
175: //
176: [self checkFingerExists];
177:
178: // Get the hand's sphere radius and palm position
179: /*
180: NSLog(@"Hand sphere radius: %f mm, palm position: %@",
181:       [hand sphereRadius], [hand palmPosition]);
182: */
183: // Get the hand's normal vector and direction
184: const LeapVector *normal = [hand palmNormal];
185: const LeapVector *direction = [hand direction];
186:
187: /*
188: // Calculate the hand's pitch, roll, and yaw angles
189: NSLog(@"Hand pitch: %f degrees, roll: %f degrees, yaw: %f degrees\n",
190:       [direction pitch] * LEAP_RAD_TO_DEG,
191:       [normal roll] * LEAP_RAD_TO_DEG,
192:       [direction yaw] * LEAP_RAD_TO_DEG);
193: */
194: }
195: }
196:
197:
198: - (void)moveRedDot{
199:
200:
201: }
202:
203: //Cycle through all the trackable dots and check if the fingers still exist.
204: //If they don't, delete them.
205: - (void)checkFingerExists{
206:
207:     for (id key in [trackableList allKeys]) {
208:         RedDot* sprite = [trackableList objectForKey:key];
209:         if (sprite.updated) {
210:             sprite.updated = FALSE;
211:             return;
212:         }else{
213:             CCNode *parent = [self getChildByTag:kTagParentNode];
214:             [trackableList removeObjectForKey:key];
215:             [parent removeChild:sprite cleanup:YES];
216:         }
217:     }
218: }
219: }
220:
221:
222: #pragma mark -
223:
224: - (void) createResetButton
225: {
226:     CCMenuItemLabel *reset = [CCMenuItemFont itemWithString:@"Reset" block:^(id sender){
227:         CCScene *s = [CCScene node];
228:         id child = [HelloWorldLayer node];
229:         [s addChild:child];
230:         [[CCDirector sharedDirector] replaceScene: s];
231:     }];
232:
233:     CCMenu *menu = [CCMenu menuWithItems:reset, nil];
234:
235:     CGSize s = [[CCDirector sharedDirector] winSize];

```

```
236:
237:         menu.position = ccp(s.width/2, 30);
238:         [self addChild: menu z:-1];
239:
240:     }
241:
242: -(void) initPhysics
243: {
244:
245:         CGSize s = [[CCDirector sharedDirector] winSize];
246:
247:         //Gravity
248:         b2Vec2 gravity;
249:         gravity.Set(0.0f, 0.0f);
250:         world = new b2World(gravity);
251:
252:
253:         // Do we want to let bodies sleep?
254:         world->SetAllowSleeping(true);
255:
256:         world->SetContinuousPhysics(true);
257:
258:         m_debugDraw = new GLESDebugDraw( PTM_RATIO );
259:         world->SetDebugDraw(m_debugDraw);
260:
261:         _world = world;
262:
263:         uint32 flags = 0;
264:         flags += b2Draw::e_shapeBit;
265:         //             flags += b2Draw::e_jointBit;
266:         //             flags += b2Draw::e_aabbBit;
267:         //             flags += b2Draw::e_pairBit;
268:         //             flags += b2Draw::e_centerOfMassBit;
269:         m_debugDraw->SetFlags(flags);
270:
271:
272:         // Define the ground body.
273:         b2BodyDef groundBodyDef;
274:         groundBodyDef.position.Set(0, 0); // bottom-left corner
275:
276:         // Call the body factory which allocates memory for the ground body
277:         // from a pool and creates the ground box shape (also from a pool).
278:         // The body is also added to the world.
279:         b2Body* groundBody = world->CreateBody(&groundBodyDef);
280:
281:         // Define the ground box shape.
282:         b2EdgeShape groundBox;
283:
284:         // bottom
285:
286:         groundBox.Set(b2Vec2(0,0), b2Vec2(s.width/PTM_RATIO,0));
287:         groundBody->CreateFixture(&groundBox,0);
288:
289:         // top
290:         groundBox.Set(b2Vec2(0,s.height/PTM_RATIO), b2Vec2(s.width/PTM_RATIO,s.height/PTM_RATIO));
291:         groundBody->CreateFixture(&groundBox,0);
292:
293:         // left
294:         groundBox.Set(b2Vec2(0,s.height/PTM_RATIO), b2Vec2(0,0));
295:         groundBody->CreateFixture(&groundBox,0);
296:
297:         // right
298:         groundBox.Set(b2Vec2(s.width/PTM_RATIO,s.height/PTM_RATIO), b2Vec2(s.width/PTM_RATIO,0));
299:         groundBody->CreateFixture(&groundBox,0);
300:
301:         _groundBody = groundBody;
302:     }
303:
304: -(void) draw
305: {
306:         //
307:         // IMPORTANT:
308:         // This is only for debug purposes
309:         // It is recommend to disable it
310:         //
311:         [super draw];
312:
313:         ccGLEnableVertexAttribs( kCCVertexAttribFlag_Position );
314:
315:         kmGLPushMatrix();
```

```

316:
317:         world->DrawDebugData();
318:
319:         kmGLPopMatrix();
320:     }
321:
322: - (RedDot*)addRedDot:(CGPoint)p finger:(NSString*)fingerID{
323:     CCNode *parent = [self getChildByTag:kTagParentNode];
324:     int idx = (CCRANDOM_0_1() > .5 ? 0:1);
325:     int idy = (CCRANDOM_0_1() > .5 ? 0:1);
326:
327:     //RedDot *sprite = [RedDot spriteWithFile:@"redcrosshair.png"];
328:     RedDot *sprite = [RedDot spriteWithTexture:spriteTexture_ rect:CGRectMake(32 * idx, 32 * idy, 32, 32)
];
329:     [parent addChild:sprite];
330:     sprite.updated = TRUE;
331:     sprite.fingerID = fingerID;
332:     sprite.position = ccp( p.x, p.y);
333:
334:     return sprite;
335: }
336:
337: - (CGPoint)covertLeapCoordinates:(CGPoint)p{
338:
339:     CGSize s = [[CCDirector sharedDirector] winSize];
340:     float screenCenter = 0.0f;
341:     float xScale = 1.75f;
342:     float yScale = 1.25f;
343:     return CGPointMake((s.width/2)+ (( p.x - screenCenter) * xScale), p.y * yScale);
344: }
345:
346: -(void) addNewSpriteAtPosition:(CGPoint)p
347: {
348:     CCLOG(@"Add sprite %0.2f x %0.2f",p.x,p.y);
349:     CCNode *parent = [self getChildByTag:kTagParentNode];
350:
351:     //We have a 64x64 sprite sheet with 4 different 32x32 images. The following code is
352:     //just randomly picking one of the images
353:     int idx = (CCRANDOM_0_1() > .5 ? 0:1);
354:     int idy = (CCRANDOM_0_1() > .5 ? 0:1);
355:     PhysicsSprite *sprite = [PhysicsSprite spriteWithTexture:spriteTexture_ rect:CGRectMake(32 * i
dx, 32 * idy, 32, 32)];
356:     [parent addChild:sprite];
357:     sprite.position = [self covertLeapCoordinates:p];
358:     //sprite.position = ccp( p.x, p.y);
359:
360:     // Define the dynamic body.
361:     //Set up a 1m squared box in the physics world
362:     b2BodyDef bodyDef;
363:     bodyDef.type = b2_dynamicBody;
364:     bodyDef.position.Set(p.x/PTM_RATIO, p.y/PTM_RATIO);
365:
366:     //bodyDef.userData = (void *) CFBridgingRetain(sprite);
367:     bodyDef.userData = (__bridge void *)sprite;
368:     b2Body *body = world->CreateBody(&bodyDef);
369:
370:     // Define another box shape for our dynamic body.
371:     b2PolygonShape dynamicBox;
372:     dynamicBox.SetAsBox(.5f, .5f); //These are mid points for our 1m box
373:
374:     // Define the dynamic body fixture.
375:     b2FixtureDef fixtureDef;
376:     fixtureDef.shape = &dynamicBox;
377:     fixtureDef.density = 1.0f;
378:     fixtureDef.friction = 0.3f;
379:     body->CreateFixture(&fixtureDef);
380:
381:     [sprite setPhysicsBody:body];
382: }
383:
384: -(void) addPieceAtPosition:(CGPoint)p
385: {
386:     CCLOG(@"Add sprite %0.2f x %0.2f",p.x,p.y);
387:     CCNode *parent = [self getChildByTag:kTagParentNode];
388:
389:     //We have a 64x64 sprite sheet with 4 different 32x32 images. The following code is
390:     //just randomly picking one of the images
391:     int idx = (CCRANDOM_0_1() > .5 ? 0:1);
392:     int idy = (CCRANDOM_0_1() > .5 ? 0:1);
393:     PhysicsSprite *sprite = [PhysicsSprite spriteWithTexture:spriteTexture_ rect:CGRectMake(32 * i

```

```
dx,32 * idy,32,32)];
394:     [parent addChild:sprite];
395:
396:     sprite.position = ccp( p.x, p.y);
397:
398:     // Define the dynamic body.
399:     //Set up a 1m squared box in the physics world
400:     b2BodyDef bodyDef;
401:     bodyDef.type = b2_dynamicBody;
402:     bodyDef.position.Set(p.x/PTM_RATIO, p.y/PTM_RATIO);
403:     b2Body *body = world->CreateBody(&bodyDef);
404:
405:     // Define another box shape for our dynamic body.
406:     b2PolygonShape dynamicBox;
407:     dynamicBox.SetAsBox(.5f, .5f);//These are mid points for our 1m box
408:
409:     // Define the dynamic body fixture.
410:     b2FixtureDef fixtureDef;
411:     fixtureDef.shape = &dynamicBox;
412:     fixtureDef.density = 1.0f;
413:     fixtureDef.friction = 0.3f;
414:     body->CreateFixture(&fixtureDef);
415:
416:     [sprite setPhysicsBody:body];
417: }
418:
419: -(void) update: (ccTime) dt
420: {
421:     //It is recommended that a fixed time step is used with Box2D for stability
422:     //of the simulation, however, we are using a variable time step here.
423:     //You need to make an informed choice, the following URL is useful
424:     //http://gafferongames.com/game-physics/fix-your-timestep/
425:
426:     int32 velocityIterations = 8;
427:     int32 positionIterations = 1;
428:
429:     // Instruct the world to perform a single step of simulation. It is
430:     // generally best to keep the time step and iterations fixed.
431:     world->Step(dt, velocityIterations, positionIterations);
432: }
433:
434: #ifdef __IPHONE_OS_VERSION_MAX_ALLOWED
435:
436: - (void)ccTouchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
437: {
438:     //Add a new body/atlas sprite at the touched location
439:     for( UITouch *touch in touches ) {
440:         CGPoint location = [touch locationInView: [touch view]];
441:
442:         location = [[CCDirector sharedDirector] convertToGL: location];
443:
444:         [self addNewSpriteAtPosition: location];
445:
446:
447:     }
448: }
449:
450: #elif defined(__MAC_OS_X_VERSION_MAX_ALLOWED)
451: /*
452: - (BOOL)ccTouchBegan:(UITouch *)touch withEvent:(UIEvent *)event {
453:     CGPoint touchLocation = [self convertTouchToNodeSpace:touch];
454:     [self selectSpriteForTouch:touchLocation];
455:     return TRUE;
456: }
457: */
458:
459:
460:
461:
462: #pragma mark - Touch Handling
463:
464: - (BOOL) ccMouseDown:(NSEvent *)event{
465:
466:     if (_mouseJoint != NULL) return NO;
467:
468:
469:     CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
470:     CGPoint mouseLocation = [self convertToNodeSpace:point];
471:     CGPoint translation = (mouseLocation);
472:     CGPoint location = translation;
```

```
473:     //location = [[CCDirector sharedDirector] convertToGL:location];
474:     b2Vec2 locationWorld = b2Vec2(location.x/PTM_RATIO, location.y/PTM_RATIO);
475:
476:
477:     // Loop through all of the Box2D bodies in our Box2D world..
478:     for(b2Body *b = _world->GetBodyList(); b; b=b->GetNext()) {
479:
480:
481:         // See if there's any user data attached to the Box2D body
482:         // There should be, since we set it in addBoxBodyForSprite
483:
484:         if (b->GetUserData() != NULL) {
485:             // We know that the user data is a sprite since we set
486:             // it that way, so cast it...
487:
488:             //PhysicsSprite *sprite = (PhysicsSprite *)CFBridgingRelease(b->GetUserData());
489:
490:
491:             for(b2Fixture *fixture = b->GetFixtureList(); fixture; fixture=fixture->GetNext()) {
492:
493:                 if(fixture->TestPoint(locationWorld)){
494:                     //NSLog(@"Touched itemType %d", sprite.itemType);
495:                     b2MouseJointDef md;
496:                     md.bodyA = _groundBody;
497:                     md.bodyB = b;
498:                     md.target = locationWorld;
499:                     md.collideConnected = true;
500:                     md.maxForce = 1000.0f * b->GetMass();
501:
502:                     _mouseJoint = (b2MouseJoint *)_world->CreateJoint(&md);
503:                     b->SetAwake(true);
504:                 }else{
505:                     //NSLog(@"NOT TOUCHED");
506:                 }
507:             }
508:         }
509:     }
510:     return YES;
511: }
512:
513: - (BOOL)ccMouseDragged:(NSEvent *)event {
514:
515:     if (_mouseJoint == NULL) return NO;
516:
517:
518:     CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
519:     CGPoint mouseLocation = [self convertToNodeSpace:point];
520:     CGPoint translation = (mouseLocation);
521:     CGPoint location = translation;
522:     //location = [[CCDirector sharedDirector] convertToGL:location];
523:     b2Vec2 locationWorld = b2Vec2(location.x/PTM_RATIO, location.y/PTM_RATIO);
524:
525:     _mouseJoint->SetTarget(locationWorld);
526:
527:     return YES;
528: }
529:
530:
531: - (BOOL)ccMouseUp:(NSEvent *)event{
532:     if (_mouseJoint) {
533:         _world->DestroyJoint(_mouseJoint);
534:         _mouseJoint = NULL;
535:
536:         //Check for any dangling mouse joints
537:         if(_world->GetJointCount() > 0){
538:             //NSLog(@"Found %d Extra Joints", _world->GetJointCount() );
539:             for(b2Joint *b = _world->GetJointList(); b; b=b->GetNext()) {
540:                 //NSLog(@"Destroying the Dangling Joint");
541:                 //Should check type first
542:                 if(b){
543:                     _world->DestroyJoint(b);
544:                     b = NULL;
545:                     return YES;
546:                 }
547:             }
548:         }
549:     }else{
550:
551:
552:         CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
```

```
553:         CGPoint mouseLocation = [self convertToNodeSpace:point];
554:         CGPoint translation = (mouseLocation);
555:         CGPoint location = translation;
556:
557:
558:         [self addNewSpriteAtPosition: location];
559:
560:     }
561:     return YES;
562: }
563:
564: #endif
565:
566: @end
```



```
1: //
2: //  HelloWorldScene.h
3: //  Cocos2DBreakout2
4: //
5:
6:
7: #import "cocos2d.h"
8: #import "Box2D.h"
9: #import "MyContactListener.h"
10: #import "LeapObjectiveC.h"
11:
12: @interface HelloWorld : CCLayer <LeapDelegate>{
13:     b2World *_world;
14:     b2Body *_groundBody;
15:     b2Body *_paddleBody;
16:     b2Fixture *_paddleFixture;
17:     b2Fixture *_ballFixture;
18:     b2Fixture *_bottomFixture;
19:     b2MouseJoint *_mouseJoint;
20:     b2MouseJoint *_fingerJoint;
21:     MyContactListener *_contactListener;
22:
23:     LeapController *_controller;
24:     NSMutableDictionary* trackableList;
25:     BOOL fingerTracked;
26: }
27:
28: + (id) scene;
29:
30: @end
```

```
1: //
2: //  HelloWorldScene.m
3: //  Cocos2DBreakout2
4: //
5:
6: #import "HelloWorldScene.h"
7:
8: #import "SimpleAudioEngine.h"
9: #import "TrackedFinger.h"
10: #define PTM_RATIO 32
11:
12: @implementation HelloWorld
13:
14: + (id)scene {
15:
16:     CCScene *scene = [CCScene node];
17:     HelloWorld *layer = [HelloWorld node];
18:     [scene addChild:layer];
19:     return scene;
20: }
21:
22:
23: - (id)init {
24:
25:     if ((self=[super init])) {
26:
27:         CGSize s = [CCDirector sharedDirector].winSize;
28:
29:         self.isMouseEnabled = YES;
30:
31:         // Create a world
32:         b2Vec2 gravity = b2Vec2(0.0f, 0.0f);
33:         bool doSleep = true;
34:         _world = new b2World(gravity);
35:
36:         // Define the ground body.
37:         b2BodyDef groundBodyDef;
38:         groundBodyDef.position.Set(0, 0); // bottom-left corner
39:
40:         // Call the body factory which allocates memory for the ground body
41:         // from a pool and creates the ground box shape (also from a pool).
42:         // The body is also added to the world.
43:         b2Body* groundBody = _world->CreateBody(&groundBodyDef);
44:
45:         // Define the ground box shape.
46:         b2EdgeShape groundBox;
47:
48:         // bottom
49:
50:         groundBox.Set(b2Vec2(0,0), b2Vec2(s.width/PTM_RATIO,0));
51:         groundBody->CreateFixture(&groundBox,0);
52:
53:         // top
54:         groundBox.Set(b2Vec2(0,s.height/PTM_RATIO), b2Vec2(s.width/PTM_RATIO,s.height/PTM_RATIO));
55:         groundBody->CreateFixture(&groundBox,0);
56:
57:         // left
58:         groundBox.Set(b2Vec2(0,s.height/PTM_RATIO), b2Vec2(0,0));
59:         groundBody->CreateFixture(&groundBox,0);
60:
61:         // right
62:         groundBox.Set(b2Vec2(s.width/PTM_RATIO,s.height/PTM_RATIO), b2Vec2(s.width/PTM_RATIO,0));
63:         groundBody->CreateFixture(&groundBox,0);
64:
65:         _groundBody = groundBody;
66:
67:         // Create sprite and add it to the layer
68:         CCSprite *ball = [CCSprite spriteWithFile:@"Ball.png" rect:CGRectMake(0, 0, 52, 52)];
69:         ball.position = ccp(100, 100);
70:         ball.tag = 1;
71:         [self addChild:ball];
72:
73:         // Create ball body
74:         b2BodyDef ballBodyDef;
75:         ballBodyDef.type = b2_dynamicBody;
76:         ballBodyDef.position.Set(100/PTM_RATIO, 100/PTM_RATIO);
77:         ballBodyDef.userData = (__bridge void *) ball;
78:         b2Body * ballBody = _world->CreateBody(&ballBodyDef);
79:
80:         // Create circle shape
```

```

81:         b2CircleShape circle;
82:         circle.m_radius = 26.0/PTM_RATIO;
83:
84:         // Create shape definition and add to body
85:         b2FixtureDef ballShapeDef;
86:         ballShapeDef.shape = &circle;
87:         ballShapeDef.density = 1.0f;
88:         ballShapeDef.friction = 0.0f; // We don't want the ball to have friction!
89:         ballShapeDef.restitution = 1.0f;
90:         _ballFixture = ballBody->CreateFixture(&ballShapeDef);
91:
92:         // Give shape initial impulse...
93:         b2Vec2 force = b2Vec2(10, 10);
94:         ballBody->ApplyLinearImpulse(force, ballBodyDef.position);
95:
96:         // Create paddle and add it to the layer
97:         CCSprite *paddle = [CCSprite spriteWithFile:@"Paddle.png"];
98:         paddle.position = ccp(s.width/2, 50);
99:         [self addChild:paddle];
100:
101:         // Create paddle body
102:         b2BodyDef paddleBodyDef;
103:         paddleBodyDef.type = b2_dynamicBody;
104:         paddleBodyDef.position.Set(s.width/2/PTM_RATIO, 50/PTM_RATIO);
105:         paddleBodyDef.userData = (__bridge void *) paddle;
106:         _paddleBody = _world->CreateBody(&paddleBodyDef);
107:
108:         // Create paddle shape
109:         b2PolygonShape paddleShape;
110:         paddleShape.SetAsBox(paddle.contentSize.width/PTM_RATIO/2,
111:                             paddle.contentSize.height/PTM_RATIO/2);
112:
113:         // Create shape definition and add to body
114:         b2FixtureDef paddleShapeDef;
115:         paddleShapeDef.shape = &paddleShape;
116:         paddleShapeDef.density = 10.0f;
117:         paddleShapeDef.friction = 0.4f;
118:         paddleShapeDef.restitution = 0.1f;
119:         _paddleFixture = _paddleBody->CreateFixture(&paddleShapeDef);
120:
121:         // Restrict paddle along the x axis
122:         b2PrismaticJointDef jointDef;
123:         b2Vec2 worldAxis(1.0f, 0.0f);
124:         jointDef.collideConnected = true;
125:         jointDef.Initialize(_paddleBody, _groundBody, _paddleBody->GetWorldCenter(), worldAxis);
126:         _world->CreateJoint(&jointDef);
127:
128:         for(int i = 0; i < 10; i++) {
129:             static int padding=20;
130:
131:             for (int j = 500; j < 1000; j+=100){
132:                 // Create block and add it to the layer
133:                 CCSprite *block = [CCSprite spriteWithFile:@"Block.png"];
134:                 int xOffset = padding+block.contentSize.width/2+((block.contentSize.width+padding)*i);
135:                 int yOffset = j; //padding+block.contentSize.height/2+((block.contentSize.height+padding)*i);
136:                 block.position = ccp(xOffset, yOffset);
137:                 block.tag = 2;
138:                 [self addChild:block];
139:
140:                 // Create block body
141:                 b2BodyDef blockBodyDef;
142:                 blockBodyDef.type = b2_dynamicBody;
143:                 blockBodyDef.position.Set(xOffset/PTM_RATIO, yOffset/PTM_RATIO);
144:                 blockBodyDef.userData = (__bridge void *) block;
145:                 b2Body *blockBody = _world->CreateBody(&blockBodyDef);
146:
147:                 // Create block shape
148:                 b2PolygonShape blockShape;
149:                 blockShape.SetAsBox(block.contentSize.width/PTM_RATIO/2,
150:                                     block.contentSize.height/PTM_RATIO/2);
151:
152:                 // Create shape definition and add to body
153:                 b2FixtureDef blockShapeDef;
154:                 blockShapeDef.shape = &blockShape;
155:                 blockShapeDef.density = 10.0;
156:                 blockShapeDef.friction = 0.0;
157:                 blockShapeDef.restitution = 0.1f;
158:                 blockBody->CreateFixture(&blockShapeDef);
159:             }

```

```
160:
161:
162:
163:
164:     }
165:
166:
167:
168:     // Create contact listener
169:     _contactListener = new MyContactListener();
170:     _world->SetContactListener(_contactListener);
171:
172:     [[SimpleAudioEngine sharedEngine] playBackgroundMusic:@"background-music-aac.caf"];
173:
174:     [self schedule:@selector(tick:)];
175:
176:     [self run];
177:     fingerTracked = FALSE;
178:
179:     [self addFingerJoint];
180:
181: }
182: return self;
183:
184: }
185:
186:
187: - (void)run
188: {
189:     controller = [[LeapController alloc] init];
190:     [controller addDelegate:self];
191:     NSLog(@"running");
192: }
193:
194: #pragma mark - SampleDelegate Callbacks
195:
196: - (void)onInit:(LeapController *)aController
197: {
198:     NSLog(@"Initialized");
199: }
200:
201: - (void)onConnect:(LeapController *)aController
202: {
203:     NSLog(@"Connected");
204: }
205:
206: - (void)onDisconnect:(LeapController *)aController
207: {
208:     NSLog(@"Disconnected");
209: }
210:
211: - (void)onExit:(LeapController *)aController
212: {
213:     NSLog(@"Exited");
214: }
215:
216: - (void)onFrame:(LeapController *)aController
217: {
218:     // Get the most recent frame and report some basic information
219:     LeapFrame *frame = [aController frame:0];
220:     /*
221:     NSLog(@"Frame id: %lld, timestamp: %lld, hands: %ld, fingers: %ld, tools: %ld",
222:     [frame id], [frame timestamp], [[frame hands] count],
223:     [[frame fingers] count], [[frame tools] count]);
224:     */
225:
226:     if ([[frame hands] count] != 0) {
227:         // Get the first hand
228:         LeapHand *hand = [[frame hands] objectAtIndex:0];
229:
230:         // Check if the hand has any fingers
231:         NSArray *fingers = [hand fingers];
232:         if ([fingers count] != 0) {
233:             // Calculate the hand's average finger tip position
234:             LeapVector *avgPos = [[LeapVector alloc] init];
235:             for (int i = 0; i < [fingers count]; i++) {
236:                 LeapFinger *finger = [fingers objectAtIndex:i];
237:                 avgPos = [avgPos plus:[finger tipPosition]];
238:
239:                 NSString* fingerID = [NSString stringWithFormat:@"%d", finger.id];
```

```

240:         /*
241:         //Check if the Finger ID exists in the list already
242:         if ([trackableList objectForKey:fingerID]) {
243:
244:             //If it does exist update the position on the screen
245:             TrackedFinger* sprite = [trackableList objectForKey:fingerID];
246:             sprite.updated = TRUE;
247:
248:
249:         }else{
250:
251:             //NSLog(@"x %0.0f y %0.0f z %0.0f", finger.tipPosition.x, finger.tipPosition.y, fi
nger.tipPosition.z);
252:             // CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
253:             //CGPoint mouseLocation = [self convertToNodeSpace:point];
254:             //Add it to the dictionary
255:             TrackedFinger* redDot = [[TrackedFinger alloc] initWithID:fingerID];
256:             [trackableList setObject:redDot forKey:fingerID];
257:         }
258:         */
259:     }
260:
261:     avgPos = [avgPos divide:[fingers count]];
262:     //NSLog(@"x %0.0f y %0.0f z %0.0f", avgPos.x, avgPos.y, avgPos.z);
263:     [self fingerMoved:CGPointMake(avgPos.x, avgPos.y)];
264:
265:
266:     //NSLog(@"Hand has %ld fingers, average finger tip position %@", [fingers count], avgPos);
267:     for (LeapFinger* finger in fingers){
268:
269:         //NSLog(@"Finger ID %d %ld", finger.id, (unsigned long)[finger hash]);
270:     }
271:
272: }
273:
274: //[[self checkFingerExists];
275:
276: // Get the hand's sphere radius and palm position
277: /*
278:     NSLog(@"Hand sphere radius: %f mm, palm position: %@",
279:     [hand sphereRadius], [hand palmPosition]);
280:     */
281: // Get the hand's normal vector and direction
282: const LeapVector *normal = [hand palmNormal];
283: const LeapVector *direction = [hand direction];
284:
285: /*
286:     // Calculate the hand's pitch, roll, and yaw angles
287:     NSLog(@"Hand pitch: %f degrees, roll: %f degrees, yaw: %f degrees\n",
288:     [direction pitch] * LEAP_RAD_TO_DEG,
289:     [normal roll] * LEAP_RAD_TO_DEG,
290:     [direction yaw] * LEAP_RAD_TO_DEG);
291:     */
292: }
293: }
294:
295:
296: - (void)tick:(ccTime) dt {
297:
298:     bool blockFound = false;
299:     _world->Step(dt, 10, 10);
300:     for(b2Body *b = _world->GetBodyList(); b; b=b->GetNext()) {
301:         if (b->GetUserData() != NULL) {
302:             CCSprite *sprite = ((__bridge CCSprite *)b->GetUserData());
303:             if (sprite.tag == 2) {
304:                 blockFound = true;
305:             }
306:
307:             if (sprite.tag == 1) {
308:                 static int maxSpeed = 10;
309:
310:                 b2Vec2 velocity = b->GetLinearVelocity();
311:                 float32 speed = velocity.Length();
312:
313:                 // When the ball is greater than max speed, slow it down by
314:                 // applying linear damping. This is better for the simulation
315:                 // than raw adjustment of the velocity.
316:                 if (speed > maxSpeed) {
317:                     b->SetLinearDamping(0.5);
318:                 } else if (speed < maxSpeed) {

```

```
319:         b->SetLinearDamping(0.0);
320:     }
321:
322: }
323:
324:     sprite.position = ccp(b->GetPosition().x * PTM_RATIO,
325:         b->GetPosition().y * PTM_RATIO);
326:     sprite.rotation = -1 * CC_RADIANS_TO_DEGREES(b->GetAngle());
327: }
328: }
329:
330: if (!blockFound) {
331:     /*
332:     GameOverScene *gameOverScene = [GameOverScene node];
333:     [gameOverScene.layer.label setString:@"You Win!"];
334:     [[CCDirector sharedDirector] replaceScene:gameOverScene];
335:     */
336:     NSLog(@"GameOver");
337: }
338:
339: std::vector<b2Body *>toDestroy;
340: std::vector<MyContact>::iterator pos;
341: for(pos = _contactListener->_contacts.begin(); pos != _contactListener->_contacts.end(); ++pos) {
342:     MyContact contact = *pos;
343:
344:     if ((contact.fixtureA == _bottomFixture && contact.fixtureB == _ballFixture) ||
345:         (contact.fixtureA == _ballFixture && contact.fixtureB == _bottomFixture)) {
346:
347:         NSLog(@"GameOver");
348:     }
349:
350:     b2Body *bodyA = contact.fixtureA->GetBody();
351:     b2Body *bodyB = contact.fixtureB->GetBody();
352:     if (bodyA->GetUserData() != NULL && bodyB->GetUserData() != NULL) {
353:         CCSprite *spriteA = (__bridge CCSprite *) bodyA->GetUserData();
354:         CCSprite *spriteB = (__bridge CCSprite *) bodyB->GetUserData();
355:
356:         // Sprite A = ball, Sprite B = Block
357:         if (spriteA.tag == 1 && spriteB.tag == 2) {
358:             if (std::find(toDestroy.begin(), toDestroy.end(), bodyB) == toDestroy.end()) {
359:                 toDestroy.push_back(bodyB);
360:             }
361:         }
362:         // Sprite B = block, Sprite A = ball
363:         else if (spriteA.tag == 2 && spriteB.tag == 1) {
364:             if (std::find(toDestroy.begin(), toDestroy.end(), bodyA) == toDestroy.end()) {
365:                 toDestroy.push_back(bodyA);
366:             }
367:         }
368:     }
369: }
370:
371: std::vector<b2Body *>::iterator pos2;
372: for(pos2 = toDestroy.begin(); pos2 != toDestroy.end(); ++pos2) {
373:     b2Body *body = *pos2;
374:     if (body->GetUserData() != NULL) {
375:         CCSprite *sprite = (__bridge CCSprite *) body->GetUserData();
376:         [self removeChild:sprite cleanup:YES];
377:     }
378:     _world->DestroyBody(body);
379: }
380:
381: if (toDestroy.size() > 0) {
382:     [[SimpleAudioEngine sharedEngine] playEffect:@"blip.caf"];
383: }
384:
385: }
386:
387: - (CGPoint)convertLeapCoordinates:(CGPoint)p{
388:
389:     CGSize s = [[CCDirector sharedDirector] winSize];
390:     float screenCenter = 0.0f;
391:     float xScale = 3.25f;
392:     float yScale = 1.25f;
393:     return CGPointMake((s.width/2)+ (( p.x - screenCenter) * xScale), p.y * yScale);
394: }
395:
396:
397: //Cycle through all the trackable dots and check if the fingers still exist.
398: //If they don't, delete them.
```

```
399: - (void)checkFingerExists{
400:     for (id key in [trackableList allKeys]) {
401:         TrackedFinger* sprite = [trackableList objectForKey:key];
402:         if (sprite.updated) {
403:             sprite.updated = FALSE;
404:             return;
405:         }else{
406:
407:             [trackableList removeObjectForKey:key];
408:
409:         }
410:     }
411:
412:     if ([trackableList count] == 0){
413:         [self fingerLost];
414:     }
415: }
416:
417: - (void)addFingerJoint{
418:
419:     b2MouseJointDef md;
420:     md.bodyA = _groundBody;
421:     md.bodyB = _paddleBody;
422:
423:     md.target = _paddleBody->GetPosition();
424:     md.collideConnected = true;
425:     md.maxForce = 1000.0f * _paddleBody->GetMass();
426:
427:     _fingerJoint = (b2MouseJoint *)_world->CreateJoint(&md);
428:     _paddleBody->SetAwake(true);
429:
430: }
431:
432:
433:
434: - (void)fingerMoved:(CGPoint)point{
435:
436:
437:     if (_fingerJoint == NULL) return;
438:
439:
440:
441:     CGPoint location = [self covertLeapCoordinates:point];
442:     NSLog(@"Dragged %0.0f , %0.0f ", location.x, location.y);
443:     b2Vec2 locationWorld = b2Vec2(location.x/PTM_RATIO, location.y/PTM_RATIO);
444:
445:     _fingerJoint->SetTarget(locationWorld);
446: }
447:
448: - (void)fingerLost{
449:
450:     if (_fingerJoint) {
451:         _world->DestroyJoint(_fingerJoint);
452:         _fingerJoint = NULL;
453:     }
454:
455: }
456:
457:
458: //- (void)ccTouchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
459: - (BOOL) ccMouseDown:(NSEvent *)event{
460:
461:     if (_mouseJoint != NULL) return NO;
462:
463:     if (_fingerJoint) {
464:
465:         //_fingerJoint->SetMaxForce(0);
466:         _world->DestroyJoint(_fingerJoint);
467:         _fingerJoint = NULL;
468:     }
469:
470:
471:     /*
472:     UITouch *myTouch = [touches anyObject];
473:     CGPoint location = [myTouch locationInView:[myTouch view]];
474:     location = [[CCDirector sharedDirector] convertToGL:location];
475:     */
476:     CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
477:     CGPoint mouseLocation = [self convertToNodeSpace:point];
478:     CGPoint translation = (mouseLocation);
```

```
479:     CGPoint location = translation;
480:
481:     b2Vec2 locationWorld = b2Vec2(location.x/PTM_RATIO, location.y/PTM_RATIO);
482:
483:     if (_paddleFixture->TestPoint(locationWorld)) {
484:         b2MouseJointDef md;
485:         md.bodyA = _groundBody;
486:         md.bodyB = _paddleBody;
487:         md.target = locationWorld;
488:         md.collideConnected = true;
489:         md.maxForce = 1000.0f * _paddleBody->GetMass();
490:
491:         _mouseJoint = (b2MouseJoint *)_world->CreateJoint(&md);
492:         _paddleBody->SetAwake(true);
493:     }
494:
495: }
496:
497: /(void)ccTouchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
498: - (BOOL)ccMouseDragged:(NSEvent *)event {
499:
500:     if (_mouseJoint == NULL) return NO;
501:     CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
502:     CGPoint mouseLocation = [self convertToNodeSpace:point];
503:     CGPoint translation = (mouseLocation);
504:     CGPoint location = translation;
505:     b2Vec2 locationWorld = b2Vec2(location.x/PTM_RATIO, location.y/PTM_RATIO);
506:
507:     _mouseJoint->SetTarget(locationWorld);
508:
509: }
510:
511: /(void)ccTouchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event {
512: - (BOOL)ccMouseUp:(NSEvent *)event{
513:
514:     if (_mouseJoint) {
515:         _world->DestroyJoint(_mouseJoint);
516:         _mouseJoint = NULL;
517:
518:         [self addFingerJoint];
519:     }else{
520:
521:         //_fingerJoint->SetMaxForce(1000.0f * _paddleBody->GetMass());
522:         //_fingerJoint->
523:
524:     }
525:
526: }
527:
528: @end
```



```
1: //
2: //  main.m
3: //  BreakOut
4: //
5: //  Created by cj on 5/7/13.
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: }
```

```
1: //
2: //  PhysicsSprite.h
3: //  LeapPuzz
4: //
5: //  Created by cj on 2/8/13.
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:
15:
16: @interface PhysicsSprite : CCSprite <CCMouseEventDelegate>
17: {
18:     CGPoint target;
19:     uint ticker;
20:     bool hasTarget;
21:     b2Body *body_; // strong ref
22: }
23: -(void) setPhysicsBody:(b2Body*)body;
24: -(void) setTarget:(CGPoint)p;
25: -(void) delTarget;
26:
27: @end
```

```
1: //
2: // PhysicsSprite.m
3: // LeapPuzz
4: //
5: // Created by cj on 2/8/13.
6: //
7: //
8:
9: #import "PhysicsSprite.h"
10: #define PTM_RATIO 32
11:
12: enum {
13:     kTagParentNode = 1,
14: };
15:
16: @implementation PhysicsSprite
17: -(void) setPhysicsBody:(b2Body *)body
18: {
19:     [[[CCEventDispatcher sharedDispatcher] addMouseDelegate:self priority:0];
20:     [[[CCDirector sharedDirector] eventDispatcher] addMouseDelegate:self priority:-1];
21:     hasTarget = NO;
22:     [[[CCTouchDispatcher sharedDispatcher] addTargetedDelegate:self priority:50 swallowsTouches:YES];
23:     body_ = body;
24: }
25:
26: // this method will only get called if the sprite is batched.
27: // return YES if the physics values (angles, position ) changed
28: // If you return NO, then nodeToParentTransform won't be called.
29: -(BOOL) dirty
30: {
31:     return YES;
32: }
33:
34: -(void) setTarget:(CGPoint)p
35: {
36:     hasTarget = YES;
37:     target = p;
38: }
39:
40: -(void) delTarget
41: {
42:     hasTarget = NO;
43: }
44:
45:
46:
47:
48: - (BOOL)ccMouseDragged:(NSEvent *)event{
49:
50:     NSLog(@"Mouse dragged in Sprite");
51:     if (hasTarget){
52:         NSLog(@"Mouse dragged in Sprite");
53:         CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
54:         //CGPoint mouseLocation = [self convertToNodeSpace:point];
55:         CGPoint translation = (point);
56:
57:         NSLog(@"Dragged %0.0f , %0.0f ", translation.x, translation.y);
58:         //self.position = translation;
59:         //body_->Get
60:         self.position = ccp(translation.x, translation.y);
61:     }
62:
63:     return YES;
64: }
65:
66:
67: - (BOOL)ccMouseDown:(NSEvent *)event{
68:
69:     NSLog(@"Mouse Down");
70:
71:
72:     CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
73:     //CGPoint mouseLocation = [self convertToNodeSpace:point];
74:     CGPoint translation = (point);
75:     NSLog(@"Dragged %0.0f , %0.0f ", translation.x, translation.y);
76:     NSLog(@"Bounding 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 );
77:
78:     if (CGRectContainsPoint([self boundingBox], translation)){
79:         NSLog(@"Sprite inside Touched");
```

```

80:         [self setTarget:translation];
81:         return YES;
82:     }
83: }
84: /*
85:     CCNode *parent = [self getChildByTag:kTagParentNode];
86:     for (PhysicsSprite *sprite in parent.children){
87:
88:
89:         if (CGRectContainsPoint([sprite boundingBox], translation)){
90:             NSLog(@"Sprite Touched");
91:
92:             [self setTarget:translation ];
93:
94:             //Move Sprite with
95:
96:             //sprite.position = translation;
97:
98:         }
99:     }
100: }
101: */
102:
103:     return NO;
104: }
105:
106: - (BOOL)ccMouseMoved:(NSEvent *)event{
107:
108:
109:     NSLog(@"Mouse Moved in Sprite");
110:
111:     return YES;
112: }
113:
114: - (BOOL)ccMouseUp:(NSEvent *)event{
115:
116:     [self delTarget];
117:
118:     return YES;
119: }
120: }
121:
122: // returns the transform matrix according the Chipmunk Body values
123: -(CGAffineTransform) nodeToParentTransform
124: {
125:     b2Vec2 pos = body_>GetPosition();
126:
127:     float x = pos.x * PTM_RATIO;
128:     float y = pos.y * PTM_RATIO;
129:
130:     if ( ignoreAnchorPointForPosition_ ) {
131:         x += anchorPointInPoints_.x;
132:         y += anchorPointInPoints_.y;
133:     }
134:
135:     // Make matrix
136:     float radians = body_>GetAngle();
137:     float c = cosf(radians);
138:     float s = sinf(radians);
139:
140:     if( ! CGPointEqualToPoint(anchorPointInPoints_, CGPointZero) ){
141:         x += c*-anchorPointInPoints_.x + -s*-anchorPointInPoints_.y;
142:         y += s*-anchorPointInPoints_.x + c*-anchorPointInPoints_.y;
143:     }
144:
145:     // Rot, Translate Matrix
146:     transform_ = CGAffineTransformMake( c,  s,
147:                                         -s,  c,
148:                                         x,   y );
149:
150:     return transform_;
151: }
152:
153:
154:
155: @end

```

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

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