```
 1: //
 2: //  AppDelegate.h
 3: //  HelloWorldBlocks
 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: #import "LeapObjectiveC.h"
11: @interface AppDelegate : NSObject <NSApplicationDelegate>
12:
13: @property (assign) IBOutlet NSWindow *window;
14:
15: @end
```

```
 1: //
 2: //  AppDelegate.m
 3: //  HelloWorldBlocks
 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: //  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:                 CGSize s = [CCDirector sharedDirector].winSize;
   47:
   48:                 // init physics
   49:                 [self initPhysics];
   50:
   51:                 // create reset button
   52:                 [self createResetButton];
   53:
   54:                 //Set up sprite
   55:
   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:      */
  125:     if ([[frame hands] count] != 0) {
  126:         // Get the first hand
  127:         LeapHand *hand = [[frame hands] objectAtIndex:0];
  128:
  129:
  130:         // Check if the hand has any fingers
  131:         NSArray *fingers = [hand fingers];
  132:
  133:         if ([fingers count] != 0) {
  134:
  135:             // Calculate the hand's average finger tip position
  136:             LeapVector *avgPos = [[LeapVector alloc] init];
  137:             for (int i = 0; i < [fingers count]; i++) {
  138:                 LeapFinger *finger = [fingers objectAtIndex:i];
  139:                 avgPos = [avgPos plus:[finger tipPosition]];
  140:
  141:
  142:                 NSString* fingerID = [NSString stringWithFormat:@"%d", finger.id];
  143:
  144:                 //Check if the Finger ID exists in the list already
  145:                 if ([trackableList objectForKey:fingerID]) {
  146:
  147:                     //If it does exist update the position on the screen
  148:                     RedDot* sprite = [trackableList objectForKey:fingerID];
  149:                     sprite.position = [self covertLeapCoordinates:CGPointMake(finger.tipPosition.x, fi
nger.tipPosition.y)];
  150:                     sprite.updated = TRUE;
  151:
  152:
  153:                 }else{
  154:
  155:                     NSLog(@"x %0.0f y %0.0f z %0.0f", finger.tipPosition.x, finger.tipPosition.y, fing
er.tipPosition.z);
  156:                    // 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 %02.f",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 %02.f",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(@"Destproying 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: //  main.m
 3: //  HelloWorldBlocks
 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: }
```