```
    1: //
    2: //  AppDelegate.h
    3: //  LeapPaint
    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 "cocos2d.h"
   11:
   12: /** Application Delegate
   13:  Creates app instance and binds libraries to interface builder xibs
   14:  Serves as an application wide callback object for events that affects the whole application, such as low-memory, etc.
   15:  */
   16: @interface AppDelegate : NSObject <NSApplicationDelegate>
   17: {
   18:
   19:     NSWindow    *window_;  /**< window is the main window to be displayed */
   20:     CCGLView    *glView_;  /**< glView is the embedded view in which cocos2d will run inside the window */
   21: }
   22: @property (strong) IBOutlet NSWindow *window;/**< window is the main window to be displayed */
   23: @property (strong) IBOutlet CCGLView *glView;/**< glView is the embedded view in which cocos2d will run inside the window */
   24: /** RunGameSceen sets up the Cocos2d environment and runs it in the application.
   25:  */
   26: - (void)runGameScene;
   27: /** Toggles from a window to full screen view point
   28:  @param sender is the action sending the command
   29:  @return IBAction binding to interface builder
   30:  */
   31: - (IBAction)toggleFullScreen:(id)sender;
   32:
   33: @end
```

```
 1: //
 2: //  AppDelegate.m
 3: //  LeapPaint
 4: //
 5: //  Created by cj on 5/7/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import "AppDelegate.h"
10: #import "GameScene.h"
11:
12: @implementation AppDelegate
13:
14: @synthesize window=window_, glView=glView_;
15:
16: - (void)applicationDidFinishLaunching:(NSNotification *)aNotification{
17:     // Insert code here to initialize your application
18:     [self runGameScene];
19: }
20: - (void)runGameScene{
21:
22:     CCDirectorMac *director = (CCDirectorMac*) [CCDirector sharedDirector];
23:
24:     //NSRect screensFrame = [[NSScreen mainScreen] frame];
25:     NSRect screensFrame = [[NSScreen mainScreen] visibleFrame];
26:     [glView_ setFrameSize:NSMakeSize(screensFrame.size.width,screensFrame.size.height)];
27:         // enable FPS and SPF
28:         [director setDisplayStats:YES];
29:         // connect the OpenGL view with the director
30:         [director setView:glView_];
31:         // EXPERIMENTAL stuff.
32:         // 'Effects' don't work correctly when autoscale is turned on.
33:         // Use kCCDirectorResize_NoScale if you don't want auto-scaling.
34:         [director setResizeMode:kCCDirectorResize_AutoScale];
35:
36:     //[glView_ setFrameSize:NSMakeSize(window_.frame.size.width,window_.frame.size.height-42)];
37:         // Enable "moving" mouse event. Default no.
38:         [window_ setAcceptsMouseMovedEvents:NO];
39:
40:         // Center main window
41:         [window_ center];
42:
43:     CCScene* scene  = [GameScene scene];
44:         [director runWithScene:scene];
45: }
46: #pragma mark AppDelegate - IBActions
47: - (IBAction)toggleFullScreen: (id)sender{
48:         CCDirectorMac *director = (CCDirectorMac*) [CCDirector sharedDirector];
49:         [director setFullScreen: ! [director isFullScreen] ];
50: }
51:
52:
53:
54:
55: @end
```

```
 1: //
 2: //  BackgroundLayer.h
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 4/9/13.
 6: //
 7: //
 8:
 9: #import <Foundation/Foundation.h>
10: #import "cocos2d.h"
11:
12:
13: /** Background Layer
14:     Displays a background image for the scene
15:  */
16: @interface BackgroundLayer : CCLayer
17:
18: @end
```

```
 1: //
 2: //  BackgroundLayer.m
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 4/9/13.
 6: //
 7: //
 8:
 9: #import "BackgroundLayer.h"
10:
11: @implementation BackgroundLayer
12:
13: /** init */
14: - (id)init
15: {
16:         if ((self = [super init]))
17:         {
18:                 // Get window size
19:                 CGSize size = [[CCDirector sharedDirector] winSize];
20:
21:                 // Add a button which takes us back to HelloWorldScene
22:
23:         // Add the generated background
24:         CCSprite *background = [CCSprite spriteWithFile:@"background-fullscreen.png"];
25:         [background setPosition:ccp(size.width / 2, size.height / 2)];
26:
27:         [self addChild:background];
28:
29:         }
30:         return self;
31: }
32:
33: @end
```

```
    1: //
    2: //  BrushSelectionLayer.h
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 4/9/13.
    6: //
    7: //
    8:
    9:
   10: #import "cocos2d.h"
   11: #import "CCControlExtension.h"
   12:
   13:
   14: /** BrushSelectionLayer Delegate
   15:  Provides a delegate interface for the layer to notify of actions
   16:  */
   17: @protocol BrushSelectionLayerDelegate <NSObject>
   18: /**
   19:  Calls back to notify that the layer can be hidden
   20:  */
   21: - (void)hidePanel;
   22: /**
   23:  Calls back to notify a new brushname has been selected
   24:  @param brushname is the name of the brush that has been selected.
   25:  */
   26: - (void)brushSelected:(NSString*)brushname;
   27: @end
   28:
   29: /** BrushSelectionLayer
   30:  This user interface layer provides a collection view of all the available brushes that can be selecte
d.
   31:  */
   32: @interface BrushSelectionLayer : CCLayer{
   33:
   34:     NSMutableDictionary* imageNamesDictionary; /**< imageNamesDictionary is the list of brush names av
ailable for selection */
   35:
   36: }
   37:
   38: @property (nonatomic, weak) id <BrushSelectionLayerDelegate> delegate;/**< delegate is the instance re
ference for triggering delegate call back functions */
   39: @property (nonatomic, readwrite) bool layerHidden;/**< layerHidded tracks the visibility state of the
layer */
   40:
   41:
   42: @end
```

```
    1: //
    2: //  BrushSelectionLayer.m
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 4/9/13.
    6: //
    7: //
    8:
    9: #import "BrushSelectionLayer.h"
   10:
   11: @implementation BrushSelectionLayer
   12: @synthesize delegate;
   13: @synthesize layerHidden;
   14: - (id)init
   15: {
   16:         if ((self = [super init]))
   17:         {
   18:                 // Get window size
   19:                 CGSize size = [[CCDirector sharedDirector] winSize];
   20:
   21:                 // Add a button which takes us back to HelloWorldScene
   22:         CCLabelTTF *titleButton = [CCLabelTTF labelWithString:@"BrushSelectionLayer" fontName:@"Marker
 Felt" fontSize:30];
   23:         titleButton.position =ccp(size.width / 2.0f , 125);
   24:
   25:
   26:         [self addChild:titleButton];
   27:         // Add the generated background
   28:         CCSprite *background = [CCSprite spriteWithFile:@"background-fullscreen.png"];
   29:         [background setPosition:ccp(size.width / 2, size.height / 2)];
   30:         self.layerHidden = true;
   31:         [self addChild:background];
   32:
   33:         [self buttoninit];
   34:
   35:         int brushCount = 30;
   36:         //NSMutableArray* menuItems = [[NSMutableArray alloc] init];
   37:         imageNamesDictionary = [[NSMutableDictionary alloc] init];
   38:         CCMenu *starMenu = [CCMenu menuWithItems:nil];
   39:         for (int i =0; i < brushCount; i++){
   40:             NSString* imagename = [NSString stringWithFormat:@"brush_%d.png",i];
   41:             CCMenuItem *starMenuItem = [CCMenuItemImage
   42:                                         itemWithNormalImage:imagename selectedImage:imagename
   43:                                         target:self selector:@selector(brushSelectedAction:)];
   44:             //starMenuItem.position = ccp(size.width /2, size.height /2);
   45:             starMenuItem.tag = i;
   46:             [imageNamesDictionary setObject:imagename forKey:[NSString stringWithFormat:@"%d",i]];
   47:
   48:
   49:
   50:             [starMenu addChild:starMenuItem];
   51:         }
   52:
   53:         //[starMenu alignItemsHorizontally];
   54:         NSNumber* itemsPerRow = [NSNumber numberWithInt:5];
   55:         [starMenu alignItemsInColumns:itemsPerRow,itemsPerRow,itemsPerRow,itemsPerRow,itemsPerRow,item
sPerRow, nil];
   56:
   57:
   58:
   59:
   60:
   61:         starMenu.position = ccp(size.width / 2, size.height / 2);
   62:         [self addChild:starMenu];
   63:
   64:         }
   65:         return self;
   66: }
   67:
   68:
   69:
   70: - (void)buttoninit{
   71:     CGSize screenSize = [[CCDirector sharedDirector] winSize];
   72:     // Add the button
   73:     CCScale9Sprite *backgroundButton = [CCScale9Sprite spriteWithFile:@"button.png"];
   74:     CCScale9Sprite *backgroundHighlightedButton = [CCScale9Sprite spriteWithFile:@"buttonHighlighted.p
ng"];
   75:
   76: #ifdef __IPHONE_OS_VERSION_MAX_ALLOWED
   77:     CCLabelTTF *titleButton = [CCLabelTTF labelWithString:@"Touch Me!" fontName:@"HelveticaNeue-Bold"
```

```
fontSize:30];
   78: #elif __MAC_OS_X_VERSION_MAX_ALLOWED
   79:     CCLabelTTF *titleButton = [CCLabelTTF labelWithString:@"Hide" fontName:@"Marker Felt" fontSize:30]
;
   80: #endif
   81:     [titleButton setColor:ccc3(159, 168, 176)];
   82:
   83:     CCControlButton *controlButton = [CCControlButton buttonWithLabel:titleButton
   84:                                                       backgroundSprite:backgroundButton];
   85:     [controlButton setBackgroundSprite:backgroundHighlightedButton forState:CCControlStateHighlighted]
;
   86:     [controlButton setTitleColor:ccWHITE forState:CCControlStateHighlighted];
   87:
   88:     controlButton.anchorPoint = ccp(0.5f, 1);
   89:     controlButton.position = ccp(screenSize.width / 2.0f, screenSize.height -100);
   90:     [self addChild:controlButton z:1];
   91:
   92:     // Add the black background
   93:     CCScale9Sprite *background = [CCScale9Sprite spriteWithFile:@"buttonBackground.png"];
   94:     [background setContentSize:CGSizeMake(300, 170)];
   95:     [background setPosition:ccp(screenSize.width / 2.0f, screenSize.height / 2.0f)];
   96:     //[self addChild:background];
   97:
   98:     // Sets up event handlers
   99:     [controlButton addTarget:self action:@selector(touchDownAction:) forControlEvents:CCControlEventTo
uchDown];
  100: }
  101:
  102: - (void)touchDownAction:(CCControlButton *)sender
  103: {
  104:
  105:
  106:     [self.delegate hidePanel];
  107: }
  108:
  109:
  110: - (void)brushSelectedAction:(id)sender
  111: {
  112:     NSLog(@"Selected Brush");
  113:
  114:     CCMenuItemImage* menuItem =     (CCMenuItemImage*)sender;
  115:     int i = menuItem.tag;
  116:     NSString* imageName = [imageNamesDictionary objectForKey:[NSString stringWithFormat:@"%d",i]];
  117:     [self.delegate brushSelected:imageName];
  118:     [self.delegate hidePanel];
  119:
  120: }
  121:
  122:
  123:
  124: @end
```

```
    1: //
    2: //  ControlsLayer.h
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 4/9/13.
    6: //
    7: //
    8:
    9: #import <Foundation/Foundation.h>
   10: #import "cocos2d.h"
   11: #import "CCControlExtension.h"
   12: #import "BrushSelectionLayer.h"
   13: #import "GameSettings.h"
   14: /**
   15:  Controls Layer Delegate
   16: Provides a delegate interface for the layer to notify of actions
   17:  */
   18: @protocol ControlsLayerDelegate <NSObject>
   19: /**
   20:  Callback with a change in color of the brush
   21:  @param color is the new selected color value
   22:  */
   23: - (void)changeColorControl:(ccColor3B)color;
   24: /**
   25:  Callback with a change in thickness of the brush
   26:  @param value is the new selected color value
   27:  */
   28: - (void)changeThicknessControl:(float)value;
   29: /**
   30:  Callback with a change in brush texture
   31:  @param brushname is the new selected brush value
   32:  */
   33: - (void)changeBrushControl:(NSString*)brushname;
   34: /**
   35:  Callback with a change in opacity
   36:  @param value is the new selected opacity value
   37:  */
   38: - (void)changeOpacityControl:(float)value;
   39: /**
   40:  Callback to notify to clear the drawing
   41:  */
   42: - (void)clearDrawing;
   43: /**
   44:  Callback with a change in color
   45:  @param mode is the  toggled eraser mode
   46:  TODO: Turn off eraser mode when new color is selected
   47:  */
   48: - (void)eraserMode:(bool)mode;
   49: @end
   50:
   51: /** Controls Layer
   52:     User inferface controls for operating buttons, switches, sliders
   53:  */
   54:
   55: @interface ControlsLayer : CCLayer <BrushSelectionLayerDelegate>{
   56:     CCLabelTTF *colorLabel;         /**< colorLabel displays name of color in hash value */
   57:     CCLabelTTF *displayValueLabel;  /**< displayValueLabel displays coordinate */
   58:     GameSettings* gameSettings;     /**< gameSettings global reference to shared settings instance */
   59: }
   60: @property (nonatomic, strong) CCControlSlider   *slider;                   /**< slider is the thickne
ss control of the brush */
   61: @property (nonatomic, strong) CCControlSlider   *opacitySlider;            /**< opacitySlider is the
opacity contro of the brush*/
   62: @property (nonatomic, strong) CCControlSwitch   *opacitySwitchControl;     /**< opacitySwitchControl
is the control for setting automatic or manual opacity control */
   63: @property (nonatomic, strong) CCLabelTTF        *opacitydisplayValueLabel;  /**<  opacitydisplayValueL
abel shows the state of the opacitySwitchControl*/
   64: @property (nonatomic, weak) id <ControlsLayerDelegate> delegate;           /**< delegate is the insta
nce reference for triggering delegate call back functions */
   65: @property (nonatomic,strong) BrushSelectionLayer *brushSelection;          /**< brushSelection layer
expands as a drawer to allow for brush selection */
   66: @property (nonatomic, strong) CCLabelTTF        *displayValueLabel;        /**< displayValueLabel dis
plays eraser toggle state */
   67: @property (nonatomic, strong) CCControlSwitch   *switchControl;            /**< switchControl is the
eraser toggle */
   68:
   69: /**
   70:  Recieves brushSizeControl delegate callbacks and updates values in the interface
   71:  @param sender is the object performing the callback
   72:  */
```

```
- (void)valueChanged:(CCControlSlider *)sender;

/**
 Recieves opacitySliderControl delegate callbacks and updates values in the interface
 @param sender is the object performing the callback
 */
- (void)opacitySliderChanged:(CCControlSlider *)sender;

/** Expands brushes panel*/
- (void)expandPanel;
/** Collapses Brushes Panel */
- (void)collapsePanel;

/** Creates and returns a new CCControlSwitch.
 @return a generate ControlSwitch
 */
- (CCControlSwitch *)makeControlSwitch;
/** Callback for the change value.
  @param sender is the object performing the callback*/
- (void)switchValueChanged:(CCControlSwitch *)sender;
/** Callback for opacity changing with the slider
  @param sender is the object performing the callback
 */
- (void)updateOpacitySlider:(float)value;

@end
```

```
    1: //
    2: //  ControlsLayer.m
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 4/9/13.
    6: //
    7: //
    8:
    9: #import "ControlsLayer.h"
   10:
   11: @implementation ControlsLayer
   12: @synthesize slider;
   13: @synthesize opacitySlider;
   14: @synthesize opacitydisplayValueLabel;
   15: @synthesize opacitySwitchControl;
   16: @synthesize delegate;
   17: @synthesize displayValueLabel;
   18: @synthesize switchControl;
   19: @synthesize brushSelection;
   20:
   21: - (id)init
   22: {
   23:         if ((self = [super init]))
   24:         {
   25:                 // Get window size
   26:                 CGSize screenSize = [[CCDirector sharedDirector] winSize];
   27:
   28:         gameSettings = [GameSettings sharedInstance];
   29:         [self sliderinit];
   30:         [self initEraserSwitch];
   31:         [self colorpickerinit];
   32:
   33:         [self initResetButton];
   34:         self.brushSelection = [BrushSelectionLayer node];
   35:         self.brushSelection.position = ccp(-screenSize.width,0);
   36:
   37:         self.brushSelection.delegate = self;
   38:
   39:         [self addChild:brushSelection z:10];
   40:         [self initBrushSelectionButton];
   41:         [self opacitySliderInit];
   42:         [self initOpacitySwitch];
   43:         }
   44:         return self;
   45: }
   46:
   47:
   48: - (void)sliderinit{
   49:
   50:     //Slideer
   51:     CGSize screenSize = [[CCDirector sharedDirector] winSize];
   52:     // Add the slider
   53:     self.slider                     = [CCControlSlider sliderWithBackgroundFile:@"sliderTrack.png"
   54:                                                                    progressFile:@"sliderProgress.png"
   55:                                                                       thumbFile:@"sliderThumb.png"];
   56:     self.slider.anchorPoint             = ccp(0.5f, 1.0f);
   57:     self.slider.minimumValue            = 0.0f; // Sets the min value of range
   58:     self.slider.maximumValue            = 5.0f; // Sets the max value of range
   59:     self.slider.position                = ccp(screenSize.width / 2.0f, 100);
   60:
   61:     CCLabelTTF *titleButton = [CCLabelTTF labelWithString:@"Size" fontName:@"Marker Felt" fontSize:30]
;
   62:     titleButton.position =ccp(screenSize.width / 2.0f , 125);
   63:
   64:     // Add the black background
   65:     CCScale9Sprite *background = [CCScale9Sprite spriteWithFile:@"buttonBackground.png"];
   66:     [background setContentSize:CGSizeMake(350,100)];
   67:     [background setPosition:ccp(screenSize.width / 2.0f, 100)];
   68:     [self addChild:background];
   69:
   70:     [self addChild:titleButton];
   71:     // When the value of the slider will change, the given selector will be call
   72:     [self.slider addTarget:self action:@selector(valueChanged:) forControlEvents:CCControlEventValueCh
anged];
   73:
   74:     [self addChild:self.slider z:0 tag:1];
   75: }
   76:
   77: - (void)valueChanged:(CCControlSlider *)sender{
   78:         // Change value of label.
```

```
   79:         //NSLog(@"slider value %@", [NSString stringWithFormat:@"Slider value = %.02f", sender.value])
;
   80:     [self.delegate changeThicknessControl:sender.value];
   81: }
   82:
   83: - (void)opacitySliderInit{
   84:
   85:     //Slideer
   86:     CGSize screenSize = [[CCDirector sharedDirector] winSize];
   87:     CCNode *layer                       = [CCNode node];
   88:     layer.position                      =ccp(screenSize.width / 2.0f + 200, 100);
   89:     [self addChild:layer z:3];
   90:
   91:     double layer_width = 0;
   92:
   93:
   94:     // Add the black background
   95:     CCScale9Sprite *background = [CCScale9Sprite spriteWithFile:@"buttonBackground.png"];
   96:     [background setContentSize:CGSizeMake(350,100)];
   97:     [background setPosition:ccp(background.contentSize.width / 2.0f, 0)];
   98:     [layer addChild:background];
   99:     layer_width += background.contentSize.width;
  100:
  101:     CCLabelTTF *titleButton = [CCLabelTTF labelWithString:@"Opacity" fontName:@"Marker Felt" fontSize:
30];
  102:     titleButton.position =ccp(layer_width / 2.0f , 25);
  103:
  104:
  105:     [layer addChild:titleButton];
  106:     // When the value of the slider will change, the given selector will be call
  107:
  108:     // Add the slider
  109:     self.opacitySlider                    = [CCControlSlider sliderWithBackgroundFile:@"sliderTrack.p
ng"
  110:                                                                                 progressFile:@"sliderProgres
s.png"
  111:                                                                                    thumbFile:@"sliderThumb.p
ng"];
  112:     self.opacitySlider.anchorPoint              = ccp(0.5f, 1.0f);
  113:     self.opacitySlider.minimumValue             = 0.0f; // Sets the min value of range
  114:     self.opacitySlider.maximumValue             = 100.0f; // Sets the max value of range
  115:     self.opacitySlider.position                 = ccp(layer_width  / 2.0f, 0);
  116:
  117:     [self.opacitySlider addTarget:self action:@selector(opacitySliderChanged:) forControlEvents:CCCont
rolEventValueChanged];
  118:
  119:     [layer addChild:self.opacitySlider z:0 tag:2];
  120: }
  121:
  122: - (void)opacitySliderChanged:(CCControlSlider *)sender{
  123:         // Change value of label.
  124:         //NSLog(@"slider value %@", [NSString stringWithFormat:@"Slider value = %.02f", sender.value])
;
  125:     [self.delegate changeOpacityControl:sender.value];
  126: }
  127:
  128: - (void)updateOpacitySlider:(float)value{
  129:     //ensure the value is within its bounds
  130:     if(value > self.opacitySlider.maximumValue){
  131:         //Max Value
  132:         self.opacitySlider.value = self.opacitySlider.maximumValue;
  133:     }else if(value < self.opacitySlider.minimumValue){
  134:         //Min Value
  135:         self.opacitySlider.value = self.opacitySlider.minimumValue;
  136:     }else{
  137:         self.opacitySlider.value = value;
  138:     }
  139: }
  140:
  141:
  142: - (void)initOpacitySwitch{
  143:     CGSize screenSize = [[CCDirector sharedDirector] winSize];
  144:
  145:     CCNode *layer               = [CCNode node];
  146:     //layer.position              = ccp (screenSize.width / 2, screenSize.height / 2);
  147:     layer.position = ccp(screenSize.width / 2.0f + 400, 125);
  148:     [self addChild:layer z:5];
  149:
  150:     double layer_width = 0;
  151:
```

```
  152:     // Add the black background for the text
  153:     CCScale9Sprite *background  = [CCScale9Sprite spriteWithFile:@"buttonBackground.png"];
  154:     background.contentSize      = CGSizeMake(80, 50);
  155:     background.position         = ccp(layer_width + background.contentSize.width / 2.0f, 0);
  156:     //[layer addChild:background];
  157:
  158:     layer_width += background.contentSize.width;
  159:     self.opacitydisplayValueLabel      = [CCLabelTTF labelWithString:@"Auto" fontName:@"Marker Felt" f
ontSize:30];
  160:
  161:     self.opacitydisplayValueLabel.position  = background.position;
  162:     //[layer addChild:self.opacitydisplayValueLabel];
  163:
  164:     // Create the switch
  165:     self.opacitySwitchControl          = [self makeControlSwitch];
  166:     self.opacitySwitchControl.position      = ccp (layer_width + 10 + self.opacitySwitchControl.conten
tSize.width / 2, 0);
  167:     self.opacitySwitchControl.on            = NO;
  168:     [layer addChild:self.opacitySwitchControl];
  169:
  170:     [self.opacitySwitchControl addTarget:self action:@selector(opacitySwitchValueChanged:) forControlE
vents:CCControlEventValueChanged];
  171:
  172:     // Set the layer size
  173:     layer.contentSize           = CGSizeMake(layer_width, 0);
  174:     layer.anchorPoint           = ccp (0.5f, 0.5f);
  175:
  176: }
  177:
  178:
  179:
  180:
  181: - (void)opacitySwitchValueChanged:(CCControlSwitch *)sender{
  182:     if ([sender isOn]){
  183:         //displayValueLabel.string    = @"Eraser";
  184:         gameSettings.depthOpacityMode  = true;
  185:     } else{
  186:         //displayValueLabel.string    = @"Eraser";
  187:         gameSettings.depthOpacityMode  = false;
  188:     }
  189: }
  190:
  191: #pragma mark - button
  192:
  193: - (void)buttoninit{
  194:     CGSize screenSize = [[CCDirector sharedDirector] winSize];
  195:     // Add the button
  196:     CCScale9Sprite *backgroundButton = [CCScale9Sprite spriteWithFile:@"button.png"];
  197:     CCScale9Sprite *backgroundHighlightedButton = [CCScale9Sprite spriteWithFile:@"buttonHighlighted.p
ng"];
  198:     CCLabelTTF *titleButton = [CCLabelTTF labelWithString:@"Touch Me!" fontName:@"Marker Felt" fontSiz
e:30];
  199:
  200:     [titleButton setColor:ccc3(159, 168, 176)];
  201:
  202:     CCControlButton *controlButton = [CCControlButton buttonWithLabel:titleButton
  203:                                                       backgroundSprite:backgroundButton];
  204:     [controlButton setBackgroundSprite:backgroundHighlightedButton forState:CCControlStateHighlighted]
;
  205:     [controlButton setTitleColor:ccWHITE forState:CCControlStateHighlighted];
  206:
  207:     controlButton.anchorPoint = ccp(0.5f, 1);
  208:     controlButton.position = ccp(screenSize.width / 2.0f, screenSize.height / 2.0f);
  209:     [self addChild:controlButton z:1];
  210:
  211:     // Add the black background
  212:     CCScale9Sprite *background = [CCScale9Sprite spriteWithFile:@"buttonBackground.png"];
  213:     [background setContentSize:CGSizeMake(300, 170)];
  214:     [background setPosition:ccp(screenSize.width / 2.0f, screenSize.height / 2.0f)];
  215:     [self addChild:background];
  216:
  217:     // Sets up event handlers
  218:     [controlButton addTarget:self action:@selector(touchDownAction:) forControlEvents:CCControlEventTo
uchDown];
  219: }
  220:
  221: - (void)touchDownAction:(CCControlButton *)sender
  222: {
  223:     NSLog(@"button value %@", [NSString stringWithFormat:@"Touch Down"]);
  224: }
```

```
  225:
  226: - (void)initEraserButton{
  227:     // Add the button
  228:     CCScale9Sprite *backgroundButton = [CCScale9Sprite spriteWithFile:@"button.png"];
  229:     CCScale9Sprite *backgroundHighlightedButton = [CCScale9Sprite spriteWithFile:@"buttonHighlighted.p
ng"];
  230:     CCLabelTTF *titleButton = [CCLabelTTF labelWithString:@"Eraser" fontName:@"Marker Felt" fontSize:3
0];
  231:
  232:     [titleButton setColor:ccc3(159, 168, 176)];
  233:
  234:     CCControlButton *controlButton = [CCControlButton buttonWithLabel:titleButton
  235:                                                       backgroundSprite:backgroundButton];
  236:     [controlButton setBackgroundSprite:backgroundHighlightedButton forState:CCControlStateHighlighted]
;
  237:     [controlButton setTitleColor:ccWHITE forState:CCControlStateHighlighted];
  238:
  239:     controlButton.anchorPoint = ccp(0.5f, 1);
  240:     controlButton.position = ccp(100, 100);
  241:     [self addChild:controlButton z:1];
  242:
  243:     // Add the black background
  244:     CCScale9Sprite *background = [CCScale9Sprite spriteWithFile:@"buttonBackground.png"];
  245:     background.anchorPoint =  ccp(0, 0);
  246:     [background setContentSize:CGSizeMake(100, 75)];
  247:     [background setPosition: ccp(50, 50)];
  248:     [self addChild:background];
  249:
  250:     // Sets up event handlers
  251:     [controlButton addTarget:self action:@selector(eraserAction:) forControlEvents:CCControlEventTouch
Down];
  252: }
  253:
  254: - (void)eraserAction:(CCControlButton *)sender
  255: {
  256:     [self.delegate changeColorControl:ccWHITE];
  257:
  258: }
  259: - (void)initResetButton{
  260:
  261:     CGSize screenSize = [[CCDirector sharedDirector] winSize];
  262:     // Add the button
  263:     CCScale9Sprite *backgroundButton = [CCScale9Sprite spriteWithFile:@"button.png"];
  264:     CCScale9Sprite *backgroundHighlightedButton = [CCScale9Sprite spriteWithFile:@"buttonHighlighted.p
ng"];
  265:     CCLabelTTF *titleButton = [CCLabelTTF labelWithString:@"Reset" fontName:@"Marker Felt" fontSize:30
];
  266:
  267:     [titleButton setColor:ccc3(159, 168, 176)];
  268:
  269:     CCControlButton *controlButton = [CCControlButton buttonWithLabel:titleButton
  270:                                                       backgroundSprite:backgroundButton];
  271:     [controlButton setBackgroundSprite:backgroundHighlightedButton forState:CCControlStateHighlighted]
;
  272:     [controlButton setTitleColor:ccWHITE forState:CCControlStateHighlighted];
  273:
  274:     controlButton.anchorPoint = ccp(0.5f, 1);
  275:     controlButton.position = ccp(100, screenSize.height -100);
  276:     [self addChild:controlButton z:1];
  277:
  278:     // Add the black background
  279:     CCScale9Sprite *background = [CCScale9Sprite spriteWithFile:@"buttonBackground.png"];
  280:     [background setContentSize:CGSizeMake(100, 75)];
  281:     [background setPosition:ccp(100, screenSize.height -125)];
  282:     [self addChild:background];
  283:
  284:     // Sets up event handlers
  285:     [controlButton addTarget:self action:@selector(resetAction:) forControlEvents:CCControlEventTouchD
own];
  286: }
  287:
  288:
  289: - (void)resetAction:(CCControlButton*)sender{
  290:     [self.delegate clearDrawing];
  291: }
  292:
  293: - (CCControlButton *)standardButtonWithTitle:(NSString *)title
  294: {
  295:     /** Creates and return a button with a default background and title color. */
  296:     CCScale9Sprite *backgroundButton = [CCScale9Sprite spriteWithFile:@"button.png"];
```

```
  297:     CCScale9Sprite *backgroundHighlightedButton = [CCScale9Sprite spriteWithFile:@"buttonHighlighted.p
ng"];
  298:     CCLabelTTF *titleButton = [CCLabelTTF labelWithString:title fontName:@"Marker Felt" fontSize:30];
  299:
  300:     [titleButton setColor:ccc3(159, 168, 176)];
  301:
  302:     CCControlButton *button = [CCControlButton buttonWithLabel:titleButton backgroundSprite:background
Button];
  303:     [button setBackgroundSprite:backgroundHighlightedButton forState:CCControlStateHighlighted];
  304:     [button setTitleColor:ccWHITE forState:CCControlStateHighlighted];
  305:
  306:     return button;
  307: }
  308:
  309:
  310: #pragma mark - ColorPicker
  311:
  312: - (void)colorpickerinit{
  313:     CGSize screenSize = [[CCDirector sharedDirector] winSize];
  314:     CCNode *layer                       = [CCNode node];
  315:     layer.position                      = ccp (screenSize.width -300 , screenSize.height / 2);
  316:     [self addChild:layer z:1];
  317:
  318:     double layer_width = 0;
  319:
  320:     // Add the black background for the text
  321:     CCScale9Sprite *background = [CCScale9Sprite spriteWithFile:@"buttonBackground.png"];
  322:     [background setContentSize:CGSizeMake(150, 50)];
  323:     [background setPosition:ccp(layer_width + background.contentSize.width / 2.0f, 0)];
  324:     //[layer addChild:background];
  325:
  326:     layer_width += background.contentSize.width;
  327:     colorLabel = [CCLabelTTF labelWithString:@"#color" fontName:@"Marker Felt" fontSize:30];
  328:     colorLabel.position = background.position;
  329:     //[layer addChild:colorLabel];
  330:
  331:     // Create the colour picker
  332:     CCControlColourPicker *colourPicker = [CCControlColourPicker colourPickerWithHueFile:@"hueBackgrou
nd.png"
  333:                                                                                 tintBackgroundFile:@"tintBackgro
und.png"
  334:                                                                                    tintOverlayFile:@"tintOverlay
.png"
  335:                                                                                         pickerFile:@"picker.png"
  336:                                                                                          arrowFile:@"arrow.png"]
;
  337:     colourPicker.color                  = ccc3(37, 46, 252);
  338:     colourPicker.position               = ccp (layer_width + colourPicker.contentSize.width / 2, 0);
  339:     colourPicker.arrowDirection         = CCControlColourPickerArrowDirectionLeft;
  340:
  341:     // Add it to the layer
  342:     [layer addChild:colourPicker];
  343:
  344: #if NS_BLOCKS_AVAILABLE
  345:     // Add block for value changed event
  346:     [colourPicker setBlock:^(id sender, CCControlEvent event)
  347:      {
  348:          [self colourValueChanged:sender];
  349:      }
  350:           forControlEvents:CCControlEventValueChanged];
  351: #else
  352:     // Add the target-action pair
  353:     [colourPicker addTarget:self action:@selector(colourValueChanged:) forControlEvents:CCControlEvent
ValueChanged];
  354: #endif
  355:
  356:     layer_width += colourPicker.contentSize.width;
  357:
  358:     // Set the layer size
  359:     layer.contentSize                   = CGSizeMake(layer_width, 0);
  360:     layer.anchorPoint                   = ccp (0.5f, 0.5f);
  361:
  362:     // Update the color text
  363:     [self colourValueChanged:colourPicker];
  364: }
  365:
  366: - (void)colourValueChanged:(CCControlColourPicker *)sender
  367: {
  368:     colorLabel.string   = [NSString stringWithFormat:@"#%02X%02X%02X",sender.color.r, sender.color.g,
sender.color.b];
```

```
  369:
  370:     [self.delegate changeColorControl:sender.color];
  371: }
  372: #pragma mark - Window Controls
  373:
  374:
  375: - (void)initEraserSwitch{
  376:
  377:
  378:
  379:     CCNode *layer               = [CCNode node];
  380:     //layer.position              = ccp (screenSize.width / 2, screenSize.height / 2);
  381:     layer.position = ccp(100, 100);
  382:     [self addChild:layer z:1];
  383:
  384:     double layer_width = 0;
  385:
  386:     // Add the black background for the text
  387:     CCScale9Sprite *background  = [CCScale9Sprite spriteWithFile:@"buttonBackground.png"];
  388:     background.contentSize      = CGSizeMake(80, 50);
  389:     background.position         = ccp(layer_width + background.contentSize.width / 2.0f, 0);
  390:     [layer addChild:background];
  391:
  392:     layer_width += background.contentSize.width;
  393:     self.displayValueLabel      = [CCLabelTTF labelWithString:@"Eraser" fontName:@"Marker Felt" fontSi
ze:30];
  394:
  395:     displayValueLabel.position  = background.position;
  396:     [layer addChild:displayValueLabel];
  397:
  398:     // Create the switch
  399:     self.switchControl          = [self makeControlSwitch];
  400:     switchControl.position      = ccp (layer_width + 10 + switchControl.contentSize.width / 2, 0);
  401:     switchControl.on            = NO;
  402:     [layer addChild:switchControl];
  403:
  404:     [switchControl addTarget:self action:@selector(switchValueChanged:) forControlEvents:CCControlEven
tValueChanged];
  405:
  406:     // Set the layer size
  407:     layer.contentSize           = CGSizeMake(layer_width, 0);
  408:     layer.anchorPoint           = ccp (0.5f, 0.5f);
  409:
  410:
  411: }
  412:
  413:
  414: - (CCControlSwitch *)makeControlSwitch
  415: {
  416:     return [CCControlSwitch switchWithMaskSprite:[CCSprite spriteWithFile:@"switch-mask.png"]
  417:                                        onSprite:[CCSprite spriteWithFile:@"switch-on.png"]
  418:                                       offSprite:[CCSprite spriteWithFile:@"switch-off.png"]
  419:                                     thumbSprite:[CCSprite spriteWithFile:@"switch-thumb.png"]
  420:                                         onLabel:[CCLabelTTF labelWithString:@"On" fontName:@"Arial-Bo
ldMT" fontSize:16]
  421:                                        offLabel:[CCLabelTTF labelWithString:@"Off" fontName:@"Arial-B
oldMT" fontSize:16]];
  422: }
  423:
  424:
  425: - (void)switchValueChanged:(CCControlSwitch *)sender{
  426:     if ([sender isOn]){
  427:         displayValueLabel.string    = @"Eraser";
  428:         [self.delegate eraserMode:true];
  429:     } else{
  430:         displayValueLabel.string    = @"Eraser";
  431:         [self.delegate eraserMode:false];
  432:     }
  433: }
  434:
  435:
  436: #pragma mark- Brush Selection Delegate
  437:
  438: - (void)initBrushSelectionButton{
  439:
  440:     CGSize screenSize = [[CCDirector sharedDirector] winSize];
  441:     // Add the button
  442:     CCScale9Sprite *backgroundButton = [CCScale9Sprite spriteWithFile:@"button.png"];
  443:     CCScale9Sprite *backgroundHighlightedButton = [CCScale9Sprite spriteWithFile:@"buttonHighlighted.p
ng"];
```

```
  444:     CCLabelTTF *titleButton = [CCLabelTTF labelWithString:@"Brushes" fontName:@"Marker Felt" fontSize:
30];
  445:
  446:     [titleButton setColor:ccc3(159, 168, 176)];
  447:
  448:     CCControlButton *controlButton = [CCControlButton buttonWithLabel:titleButton
  449:                                                       backgroundSprite:backgroundButton];
  450:     [controlButton setBackgroundSprite:backgroundHighlightedButton forState:CCControlStateHighlighted]
;
  451:     [controlButton setTitleColor:ccWHITE forState:CCControlStateHighlighted];
  452:
  453:     controlButton.anchorPoint = ccp(0.5f, 1);
  454:     controlButton.position = ccp(screenSize.width -100, screenSize.height -100);
  455:     [self addChild:controlButton z:1];
  456:
  457:     // Add the black background
  458:     CCScale9Sprite *background = [CCScale9Sprite spriteWithFile:@"buttonBackground.png"];
  459:     [background setContentSize:CGSizeMake(100, 75)];
  460:     [background setPosition:ccp(screenSize.width -100, screenSize.height -125)];
  461:     [self addChild:background];
  462:
  463:     // Sets up event handlers
  464:     [controlButton addTarget:self action:@selector(brushButtonAction:) forControlEvents:CCControlEvent
TouchDown];
  465: }
  466:
  467:
  468: - (void)brushButtonAction:(CCControlButton*)sender{
  469:     if (self.brushSelection.layerHidden) {
  470:         [self showBrushSelectionPanel];
  471:     }else{
  472:         [self hideBrushSelectionPanel];
  473:     }
  474:
  475: }
  476:
  477: - (void)showBrushSelectionPanel{
  478:     self.brushSelection.layerHidden = false;
  479:     [self.brushSelection runAction:[CCMoveTo actionWithDuration:2 position:ccp(0,0)]];
  480: }
  481:
  482: - (void)hideBrushSelectionPanel{
  483:     // Get window size
  484:     CGSize screenSize = [[CCDirector sharedDirector] winSize];
  485:
  486:     self.brushSelection.layerHidden = true;
  487:     [self.brushSelection runAction:[CCMoveTo actionWithDuration:2 position:ccp(-screenSize.width,0)]];
  488: }
  489:
  490: - (void)hidePanel{
  491:     [self hideBrushSelectionPanel];
  492: }
  493:
  494:
  495: - (void)brushSelected:(NSString *)brushname{
  496:     [self.delegate changeBrushControl:brushname];
  497: }
  498:
  499:
  500: @end
```

```
    1: //
    2: //  GameManager.h
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 4/2/13.
    6: //
    7: //
    8:
    9: #import <Foundation/Foundation.h>
   10: #import "cocos2d.h"
   11: #import "LeapObjectiveC.h"
   12: #import "HUDLayer.h"
   13:
   14:
   15: #import "SketchRenderTextureScene.h"
   16: #import "BackgroundLayer.h"
   17: #import "ControlsLayer.h"
   18: #import "GameSettings.h"
   19: #import "SimplePoint.h"
   20:
   21: /**
   22:  Core Application Management
   23:  Provides interfaces and controls the various inputs, controls and outputs
   24:
   25:  */
   26: @interface GameManager : CCScene <LeapListener, HUDDelegate, ControlsLayerDelegate>
   27: {
   28:     InputMode inputMode;    /**< colorLabel displays name of color in hash value */
   29:     LeapPointable* currentPointable; /**< colorLabel displays name of color in hash value */
   30:     CGPoint currentPoint;   /**< colorLabel displays name of color in hash value */
   31:     //Settings
   32:     BOOL painting;      /**< painting indicates wether or not the application is painting at that mome
nt*/
   33:
   34:     GameSettings* gameSettings; /**< gameSettings singleton to global seetings*/
   35:
   36:
   37:     int lastTag;                /**< lastTag is the last tag value tracked of a LeapPointable */
   38:     SimplePoint* lastPoint;     /**< lastPoint is the last known point on the screen of the LeapPointa
ble */
   39:     int framesSinceLastFound;   /**< framesSinceLastFound number of frames since last finding a LeapPo
intable */
   40:
   41:
   42: }
   43:
   44: @property (nonatomic,strong) HUDLayer* hudLayer;                    /**< hudLayer displays the icons f
or tracking where a leapPointable is pointing */
   45: @property (nonatomic,strong) SketchRenderTextureScene* textureScene;/**< textureScene is the drawing l
ayer */
   46: @property (nonatomic,strong) BackgroundLayer* backgroundLayer;      /**< backgroundLayer is the layer
for setting up the background */
   47: @property (nonatomic,strong) ControlsLayer* controlsLayer;          /**< controlsLayer is the layer fo
r managing interface controls */
   48:
   49: @property (nonatomic,strong) LeapController* controller;           /**< controller is the leapControl
ler */
   50: @property (nonatomic,strong) LeapScreen* leapScreen;               /**< leapScreen references the scr
een being used on the system */
   51:
   52: /**
   53:  Finds the percentage of a number between two values
   54:  If the number is greater or less than the range, that boundry of the range will be returned.
   55:  @param max is the top range value
   56:  @param min is the bottom range value
   57:  @param value is the number we are seeking the percentage from
   58:  @return the a percentage between 0 and 100%
   59:  */
   60: - (float)findPecentageDifference:(float)max withMin:(float)min withValue:(float)value;
   61:
   62: /**
   63:  Determines the opacity based upon the Z axis coordinate
   64:  @param value is the Z axis coordinate
   65:  @return the opacity value to set the brush at.
   66:  */
   67: - (float)opacityPercentage:(float)value;
   68:
   69: @end
```

```
    1: //
    2: //  GameManager.m
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 4/2/13.
    6: //
    7: //
    8:
    9: #import "GameManager.h"
   10:
   11: @implementation GameManager
   12:
   13: @synthesize hudLayer;
   14: @synthesize textureScene;
   15: @synthesize backgroundLayer;
   16: @synthesize controlsLayer;
   17: @synthesize controller;
   18: @synthesize leapScreen;
   19:
   20:
   21: // On "init" you need to initialize your instance
   22: -(id) init
   23: {
   24:         // always call "super" init
   25:         // Apple recommends to re-assign "self" with the "super" return value
   26:         if( (self=[super init])) {
   27:
   28:
   29:                 // create and initialize a Label
   30:                 CCLabelTTF *label = [CCLabelTTF labelWithString:@"Leap Paint" fontName:@"Marker Felt"
fontSize:64];
   31:
   32:                 // ask director the the window size
   33:                 CGSize size = [[CCDirector sharedDirector] winSize];
   34:
   35:         NSLog(@"Window size (pixels)-- Width: %0.0f Height: %0.0f",size.width, size.height);
   36:
   37:                 // position the label on the center of the screen
   38:                 label.position =  ccp( size.width /2 , size.height - 25 );
   39:                 // add the label as a child to this Layer
   40:
   41:                 [self addChild: label];
   42:         [self run];
   43:
   44:         inputMode = kPressKeyMode;
   45:         painting = false;
   46:
   47:         gameSettings = [GameSettings sharedInstance];
   48:
   49:         lastTag = -1;
   50:         lastPoint = [[SimplePoint alloc] initWithX:0.0f withY:0.0f withZ:0.0f];
   51:         framesSinceLastFound = 0;
   52:
   53:         }
   54:         return self;
   55: }
   56:
   57: #pragma mark - SampleDelegate Callbacks
   58:
   59: /**
   60:  LeapMotion SDK Delegate Callback
   61:  Init's a LeapMotion instance to initiate connection and tracking with the LeapMotion and assigns the
delegate or listener for the controller
   62:  */
   63: - (void)run
   64: {
   65:     controller = [[LeapController alloc] init];
   66:     [controller addListener:self];
   67:
   68: }
   69: /**
   70:  LeapMotion SDK Delegate Callback
   71:  Initialize
   72:  Verifies the LeapMotion has been initialized and any additional steps for setup can continue.
   73:  */
   74:
   75: - (void)onInit:(NSNotification *)notification{
   76:     NSLog(@"Leap: Initialized");
   77: }
   78: /**
```

```
   79:  LeapMotion SDK Delegate Callback
   80:  Connect
   81:  Verifies the LeapMotion is connected and additional steps for setup can continue.
   82:  Sets up the screens to be track intersecting vectors from pointables.
   83:  */
   84: - (void)onConnect:(NSNotification *)notification{
   85:     NSLog(@"Leap: Connected");
   86:     NSArray* screens = controller.locatedScreens;
   87:     if ([screens count] > 0){
   88:         leapScreen = [screens objectAtIndex:0];
   89:         NSLog(@"Screens: %0.0ld", (unsigned long)[screens count]);
   90:     }else{
   91:         NSLog(@"No Screens");
   92:     }
   93: }
   94: /**
   95:  LeapMotion SDK Delegate Callback
   96:  Disconnect
   97:  Notifies the application that the LeapMotion has been disconnected and hold or release any processes
in regard to the LeapMotion
   98:  */
   99: - (void)onDisconnect:(NSNotification *)notification{
  100:     NSLog(@"Leap: Disconnected");
  101: }
  102:
  103: /**
  104:  LeapMotion SDK Delegate Callback
  105:  Exits
  106:  Releases memory and sets object instances to nil (null)
  107:  */
  108: - (void)onExit:(NSNotification *)notification{
  109:     NSLog(@"Leap: Exited");
  110: }
  111: /**
  112:  LeapMotion SDK Delegate Callback
  113:  OnFrame Event notifies the application that an incoming frame has been processed and the data can be
used to control the application
  114:  */
  115: - (void)onFrame:(NSNotification *)notification{
  116:     LeapController *aController = (LeapController *)[notification object];
  117:     // Get the most recent frame and report some basic information
  118:     LeapFrame *frame = [aController frame:0];
  119:
  120:     //Try and find the same one as last time.
  121:     if ([[frame pointables] count] != 0) {
  122:         NSArray* leapPointables = [frame pointables];
  123:
  124:         LeapPointable* tool;
  125:         if (lastTag != -1){
  126:             for (LeapPointable* pointable in leapPointables){
  127:                 if (lastTag == pointable.id){
  128:                     tool = pointable;
  129:                     lastTag = pointable.id;
  130:                     break;
  131:                 }
  132:             }
  133:
  134:             //Find a new point able
  135:             if (tool == nil){
  136:                 tool = [self pointableClosestToScreen:leapPointables];
  137:                 lastTag = tool.id;
  138:             }
  139:         }else{
  140:             //Find a new pointable
  141:             tool = [self pointableClosestToScreen:leapPointables];
  142:             lastTag = tool.id;
  143:         }
  144:
  145:         //Get the screen
  146:         LeapVector* normalized = [leapScreen intersect:tool normalize:YES clampRatio:2.0];
  147:
  148:         if ([leapScreen isValid]){
  149:             double x = normalized.x * [leapScreen widthPixels];
  150:             double y = normalized.y * [leapScreen heightPixels];
  151:
  152:             CGPoint pointer = CGPointMake(x, y);
  153:
  154:             //Convert to Local coordinates from Screen Coordinates
  155:             CCDirector* director = [CCDirector sharedDirector];
  156:             NSPoint var = [director.view.window convertScreenToBase:pointer];
```

```
  157:
  158:             //Logging
  159:             //NSLog(@"x %0.0f y %0.0f z %0.0f Pointer: x %0.0f y %0.0f ", x, y,tool.tipPosition.z, var
.x, var.y);
  160:             //SimplePoint* simplePoint = [[SimplePoint alloc] initWithPosition:var withZ:tool.tipPosit
ion.z];
  161:             //[[NSNotificationCenter defaultCenter] postNotificationName:@"CoordHUDUpdate" object:simp
lePoint];
  162:
  163:             if (gameSettings.depthOpacityMode){
  164:                 float opacity = [self opacityPercentage:tool.tipPosition.z];
  165:                 //Update the controls
  166:                 [controlsLayer updateOpacitySlider:opacity];
  167:             }
  168:
  169:             if (inputMode == kDepthMode){
  170:
  171:                 if (tool.tipPosition.z > 0){
  172:                     painting = FALSE;
  173:                 }else{
  174:                     painting = TRUE;
  175:                 }
  176:             }
  177:
  178:             //Update the HUD View
  179:             [self.hudLayer toolMoved:var toolID:[NSString stringWithFormat:@"%0.0d",tool.id]];
  180:             if (painting){
  181:                 [self movedToolTexture:var tool:tool];
  182:             }else{
  183:                // NSLog(@"Not Painting");
  184:             }
  185:
  186:         }else{
  187:             NSLog(@"Leap Screen is invalid");
  188:         }
  189:     }else{
  190:         NSLog(@"No frame");
  191:         //Remove the marker from the HUD view
  192:         if (currentPointable != nil) {
  193:
  194:             [self endLineDrawingTexture:currentPoint tool:currentPointable];
  195:             [self.hudLayer endTrackingTool];
  196:         }
  197:
  198:         lastTag = -1;
  199:
  200:         framesSinceLastFound ++;
  201:         if (framesSinceLastFound > kMaxFrames){
  202:
  203:             framesSinceLastFound = 0;
  204:         }
  205:     }
  206: }
  207:
  208: #pragma mark - TextureScene
  209:
  210: /** Moves the tool on the screen when painting */
  211: - (void)movedToolTexture:(CGPoint)point tool:(LeapPointable*)pointable{
  212:
  213:     if (currentPointable != nil){
  214:
  215:         [self moveLineDrawingTexture:point tool:pointable];
  216:         currentPointable = pointable;
  217:     }else{
  218:         [self beginLineDrawingTexture:point tool:pointable];
  219:         currentPointable = pointable;
  220:     }
  221: }
  222:
  223: /** Begin drawing to the canvas
  224:  @param point is the current coordinate the LeapPointable is interescting with the screen
  225:  @param pointable is a reference to the pointable currently drawing
  226:  */
  227: - (void)beginLineDrawingTexture:(CGPoint)point tool:(LeapPointable*)pointable{
  228:
  229:     [self.textureScene beginDraw:point withZ:pointable.tipPosition.z];
  230:     currentPoint = point;
  231: }
  232: /** Update drawing with a moved image on the canvas
  233:   @param point is the current coordinate the LeapPointable is interescting with the screen
```

```
234:   @param pointable is a reference to the pointable currently drawing
235:  */
236: - (void)moveLineDrawingTexture:(CGPoint)point tool:(LeapPointable*)pointable{
237:
238:     [self.textureScene updateDraw:point withZ:pointable.tipPosition.z];
239:     currentPoint = point;
240: }
241: /** End the drawing
242:   @param point is the current coordinate the LeapPointable is interescting with the screen
243:   @param pointable is a reference to the pointable currently drawing
244:  */
245: - (void)endLineDrawingTexture:(CGPoint)point tool:(LeapPointable*)pointable{
246:     [self.textureScene endDraw:point];
247:     currentPointable = nil;
248: }
249:
250: #pragma mark - Keyboard Events
251:
252: /** Change Input Mode */
253: - (void)changeMode:(InputMode)mode{
254:     //NSLog(@"Changemode");
255:     inputMode = mode;
256:     gameSettings.inputMode = mode;
257: }
258: /** Change Paiting state
259:  @param paintState changes the painting sate
260:  */
261: - (void)painting:(BOOL)paintingState{
262:     painting = paintingState;
263:     gameSettings.painting = paintingState;
264: }
265: #pragma mark - ControlsDelegate
266:
267: /** Change the color of the brush
268:  Updates the HUD Layer and the Texture Layer
269:   @param color is the color to be changed
270:  */
271:
272: - (void)changeColorControl:(ccColor3B)color{
273:     [self.hudLayer changeColor:color];
274:     [self.textureScene changeColor:color];
275:
276: }
277: /** Change the thickness of the brush
278:  Updates the HUD Layer and the Texture Layer
279:  @param value is the thinkness(width) value
280:  */
281: - (void)changeThicknessControl:(float)value{
282:     [self.hudLayer changeScale:value];
283:     [self.textureScene changeScale:value];
284: }
285: /** Change the brush type
286:  Updates the HUD Layer and the Texture Layer
287:  @param brushname is the name of the brush to be changed
288:  */
289: - (void)changeBrushControl:(NSString *)brushname{
290:
291:     [self.hudLayer changeBrush:brushname];
292:     [self.textureScene changeBrush:brushname];
293: }
294: /** Change the opacity of the brush
295:  Updates the HUD Layer and the Texture Layer
296:  @param value is the opacity value
297:  */
298: - (void)changeOpacityControl:(float)value{
299:     [self.textureScene changeOpacity:value];
300: }
301:
302: /** Clears the drawing */
303: - (void)clearDrawing{
304:     [self.textureScene clearDrawing];
305:     //**Turns off eraser mode if it is on
306:     if (gameSettings.eraserMode){
307:         gameSettings.eraserMode = false;
308:
309:         //Update texture mode and update Controls layer
310:     }
311: }
312: /** Update the eraser mode
313:  Updates the HUD Layer and the Texture Layer
```

```
  314:  @param mode is the current state of the eraser
  315:  */
  316: - (void)eraserMode:(bool)mode{
  317:
  318:     [self.hudLayer erasingMode:mode];
  319:     [self.textureScene erasingMode:mode];
  320:
  321: }
  322:
  323: /** Return the Opacity value based on Z position */
  324: - (float)opacityPercentage:(float)value{
  325:     //NSLog(@"value %0.0f", value);
  326:     if (value < kOpMinRange){
  327:         return kOpMax;
  328:     }else if(value > kOpMaxRange){
  329:         return kOpMin;
  330:     }else {
  331:         float percentage = [self findPecentageDifference:kOpMaxRange withMin:kOpMinRange withValue:val
ue];
  332:         percentage = 100 - percentage;
  333:         return percentage;
  334:     }
  335: }
  336: /** Find the percentage between two numbers */
  337: - (float)findPecentageDifference:(float)max withMin:(float)min withValue:(float)value{
  338:     return (value - min)/(max - min)*100;
  339: }
  340: /**
  341:  Using all the pointables, gets the closest one to the screen
  342:  @param pointables is an array of pointables currently observered by the LeapMotion
  343:  @return pointable that is closest by the screen
  344:  */
  345: - (LeapPointable*)pointableClosestToScreen:(NSArray*)pointables{
  346:     LeapPointable* closestPointable;
  347:     for (LeapPointable*pointable in pointables){
  348:
  349:         //Check for the first iteration that the closest is not equal to nil
  350:         if (closestPointable != nil){
  351:             if (closestPointable.tipPosition.z > pointable.tipPosition.z){
  352:                 closestPointable = pointable;
  353:             }
  354:         }else{
  355:             closestPointable = pointable;
  356:         }
  357:     }
  358:     return closestPointable;
  359: }
  360:
  361: /**
  362:  Find the closest LeapPointable to the current last vector
  363:  @param leapVector is the position of the last pointbale
  364:  @param pointables is an array of pointables currently observered by the LeapMotion
  365:  @return LeapPointable closest to a leapVector
  366:  */
  367: - (LeapPointable*)pointableClosestToVector:(LeapVector*)leapVector withPointables:(NSArray*)pointables
{
  368:     LeapPointable* closestPointable;
  369:     //Check to make sure there is atleast one object in the array
  370:
  371:     //if the array is empty, throw an exception
  372:     if ([pointables count] == 0){
  373:         NSLog(@"Cannot pass item 0 array");
  374:         return nil;
  375:     }
  376:     //If there is only one object in the array, return it
  377:     else if ([pointables count] == 1){
  378:         return [pointables objectAtIndex:0];
  379:     }else{
  380:         //Get the distance for the first point
  381:         float minDistance = 0;
  382:         closestPointable = [pointables objectAtIndex:0];
  383:         minDistance = [leapVector distanceTo:closestPointable.tipPosition];
  384:
  385:         for (int i = 1; i < [pointables count]; i++){
  386:
  387:             LeapPointable* point = [pointables objectAtIndex:i];
  388:             float distance = [leapVector distanceTo:point.tipPosition];
  389:             if ( distance < minDistance){
  390:                 minDistance = distance;
  391:                 closestPointable = point;
```

```
392:             }
393:         }
394:     }
395:     return closestPointable;
396: }
397: @end
```

```
 1: //
 2: //  GameScene.h
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 4/1/13.
 6: //
 7: //
 8:
 9: #import <Foundation/Foundation.h>
10: #import "cocos2d.h"
11: #import "HUDLayer.h"
12: #import "GameManager.h"
13: #import "LeapObjectiveC.h"
14: #import "SketchRenderTextureScene.h"
15: #import "BackgroundLayer.h"
16: #import "ControlsLayer.h"
17:
18: /**
19:  GameScene
20:  Initializes and assembles all of the layers and gameobjects into the GameManager
21:  */
22: @interface GameScene : CCScene
23: /**
24:  Scene initializes each object and assigns interlinking pointers and delegates to each class
25:  @return scene for CCDirector to begin running
26:  */
27: +(CCScene *) scene;
28: @end
```

```
 1: //
 2: //  GameScene.m
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 4/1/13.
 6: //
 7: //
 8:
 9: #import "GameScene.h"
10:
11: @implementation GameScene
12:
13: +(CCScene *) scene
14: {
15:         // 'scene' is an autorelease object.
16:     GameManager*scene = [GameManager node];
17:
18:         HUDLayer* hudLayer = [HUDLayer node];
19:     BackgroundLayer* backgroundLayer = [BackgroundLayer node];
20:     ControlsLayer* controlsLayer = [ControlsLayer node];
21:     SketchRenderTextureScene* textureScene = [SketchRenderTextureScene node];
22:
23:     //setup delegates
24:     hudLayer.delegate = scene;
25:     controlsLayer.delegate = scene;
26:
27:     // add layer as a child to scene
28:     [scene addChild:backgroundLayer z:0];
29:     [scene addChild:controlsLayer z:3];
30:         [scene addChild:hudLayer z:5];
31:     [scene addChild:textureScene z:2];
32:     scene.hudLayer = hudLayer;
33:     scene.backgroundLayer = backgroundLayer;
34:     scene.controlsLayer = controlsLayer;
35:     scene.textureScene = textureScene;
36:
37:         // return the scene
38:         return scene;
39: }
40: @end
```

```
    1: //
    2: //  GameSettings.h
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 4/16/13.
    6: //
    7: //
    8:
    9: #import <Foundation/Foundation.h>
   10:
   11: #define kVelMax 1000
   12: #define kVelMin 0
   13: #define kOpMinRange -80
   14: #define kOpMaxRange 120
   15: #define kOpMin 0
   16: #define kOpMax 100
   17: #define kNormalizedVelMax 15
   18: #define kNormalizedVelMin 0
   19: #define kMaxFrames 1000
   20: extern int const BLOCK_SIZE;
   21:
   22: typedef enum {
   23:     kPressKeyMode,
   24:     kDepthMode,
   25: } InputMode;
   26:
   27: /**
   28:  GameSettings is a globally shared class instance which tracks all the game settings.
   29:  This class can be accessed by any object in the game.
   30:  */
   31: @interface GameSettings : NSObject
   32: @property (nonatomic,readwrite) BOOL depthOpacityMode;      /**< depthOpacityMode controls use of z ax
is control of opacity */
   33: @property (nonatomic,readwrite) BOOL painting;      /**< painting indicates wether or not the applicat
ion is painting at that moment*/
   34: @property (nonatomic,readwrite) BOOL eraserMode;            /**< eraserMode controls erasing on drawin
g canvas */
   35: @property (nonatomic,readwrite) InputMode inputMode;       /**< inputMode controller input mode for l
eapmotion */
   36: /** Singleton
   37:  Intiailizes and Returns a shared instance of the class
   38:  @return sharedInstance of the class.
   39:  */
   40: + (GameSettings *)sharedInstance;
   41: @end
```

```objc
//
//  GameSettings.m
//  LeapPuzz
//
//  Created by cj on 4/16/13.
//
//

#import "GameSettings.h"

//Constants
int const  BLOCK_SIZE = 128;

@implementation GameSettings
@synthesize depthOpacityMode;
@synthesize eraserMode;
@synthesize inputMode;
@synthesize painting;

/** Singleton SharedInstance
 Intiailizes and Returns a shared instance of the class
 */
+ (GameSettings *)sharedInstance{
    static GameSettings *sharedInstance;
    @synchronized(self)
    {
        if (!sharedInstance)
            sharedInstance = [[GameSettings alloc] init];
        return sharedInstance;
    }
}

/**
 Initialize the class and sets the default values
 */
- (id)init{
    if (self = [super init]) {
        // Init Defaults
        self.depthOpacityMode = false;
        self.painting = false;
    }
    return self;
}
@end
```

```
    1: //
    2: //  HUDLayer.h
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 4/1/13.
    6: //
    7: //
    8:
    9: #import <Foundation/Foundation.h>
   10: #import "cocos2d.h"
   11: #import "LPTool.h"
   12: #import "LeapObjectiveC.h"
   13: #import "SimplePoint.h"
   14: #import "GameSettings.h"
   15:
   16:
   17: /** HUD Delegate Protocol
   18:  User inferface controls for operating buttons, switches, sliders
   19:  */
   20: @protocol HUDDelegate <NSObject>
   21: /**
   22:  Calls back to notify a new input mode has been selected by the keyboard interface
   23:  @param mode is the state of the input mode
   24:  */
   25: - (void)changeMode:(InputMode)mode;
   26: /**
   27:  Calls back to notify a new change in painting state
   28:  @param paintingState
   29:  */
   30: - (void)painting:(BOOL)paintingState;
   31:
   32: @end
   33:
   34: /** HUD Layer
   35:  Tracks the position of the LeapCursor on the screen
   36:  */
   37: @interface HUDLayer : CCLayer  {
   38:     NSString* primaryToolID;    /**< primaryToolID stores the id tag to the pointable in reference*/
   39:     LPTool* primaryTool;        /**< primaryTool points to the current pointable  object*/
   40:     InputMode inputMode;        /**< inputMode is the current mode of input*/
   41:
   42:     ccColor3B lastColor;        /**< lastColor is the lastColor to be selected */
   43:     ccColor3B previousColor;    /**< previousColor is the color before the lastcolor to be selected */
   44:     NSString* lastBrush;        /**< lastBrush is last brush to be selected */
   45:     float lastScale;            /**< lastScale is last scale to be selected */
   46:
   47:     CCSprite* paintingIndicator; /**< paintingIndicator shows the state at which the object is current
ly paintg */
   48:     BOOL eraseMode;             /**< eraseMode determines weather the pointable is painting or erasing
 */
   49:
   50:
   51:     GameSettings* gameSettings; /**< gameSettings singleton to global seetings*/
   52:
   53:
   54: }
   55:
   56: @property (nonatomic, weak) id <HUDDelegate> delegate; /**< colorLabel displays name of color in hash
value */
   57: @property (nonatomic, strong) CCLabelTTF* xyzcoords; /**< xyzcoords is the X,Y,Z coordinates in string
 form for displaying on the HUD in real-time for debugging */
   58:
   59: /**
   60:  ToolMoved updates the last known tracked position of the tool.
   61:  @param point is the coordinate location on the screen in which pointable interesects
   62:  @param toolid is LeapSDK provided tool id of the tool moving
   63:  */
   64: - (void)toolMoved:(CGPoint)point toolID:(NSString*)toolid;
   65: /**
   66:  StartTrackingTool begins the process of tracking a tool starting with a new path
   67:  @param point is the coordinate location on the screen in which pointable interesects
   68:  @param toolid is LeapSDK provided tool id of the tool moving
   69:  */
   70: - (void)startTrackingTool:(CGPoint)point toolID:(NSString*)toolid;
   71: /**
   72:  MoveTrackingTool updates the position and path of a tool.
   73:  @param point is the coordinate location on the screen in which pointable interesects
   74:  @param toolid is LeapSDK provided tool id of the tool moving
   75:  */
   76: - (void)moveTrackingTool:(CGPoint)point toolID:(NSString*)toolid;
```

```
77: /**
78:  EndTracking tool singles the end of the tool being tracked.
79:  The tool may be lost or no longer drawing
80:  */
81: - (void)endTrackingTool;
82:
83:
84: - (void)changeColor:(ccColor3B)color;
85: - (void)changeBrush:(NSString*)brushname;
86: - (void)changeScale:(float)size;
87: - (void)erasingMode:(BOOL)mode;
88:
89: @end
```

```
    1: //
    2: //  HUDLayer.m
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 4/1/13.
    6: //
    7: //
    8:
    9: #import "HUDLayer.h"
   10:
   11: @implementation HUDLayer
   12: @synthesize delegate;
   13: @synthesize xyzcoords;
   14: - (id)init
   15: {
   16:         if ((self = [super init]))
   17:         {
   18:                 // Get window size
   19:                 CGSize size = [[CCDirector sharedDirector] winSize];
   20:
   21:                 // Add a button which takes us back to HelloWorldScene
   22:
   23:                 // Create a label with the text we want on the button
   24:                 CCLabelTTF *label = [CCLabelTTF labelWithString:@"Tap Here" fontName:@"Helvetica" font
Size:32.0];
   25:
   26:                 // Create a button out of the label, and tell it to run the "switchScene" method
   27:                 CCMenuItem *button = [CCMenuItemLabel itemWithLabel:label target:self selector:@select
or(testing:)];
   28:
   29:                 // Add the button to a menu - "nil" terminates the list of items to add
   30:                 CCMenu *menu = [CCMenu menuWithItems:button, nil];
   31:
   32:                 // Place the menu in center of screen
   33:                 [menu setPosition:ccp(size.width / 2, size.height / 2)];
   34:
   35:         lastColor = ccWHITE;
   36:         lastBrush = @"roundbrush.png";
   37:         lastScale = 1.0;
   38:
   39:         eraseMode = false;
   40:
   41:                 // Finally add the menu to the layer
   42:                 //[self addChild:menu];
   43: #ifdef __IPHONE_OS_VERSION_MAX_ALLOWED
   44:                 self.isTouchEnabled = YES;
   45:                 self.isAccelerometerEnabled = YES;
   46: #elif defined(__MAC_OS_X_VERSION_MAX_ALLOWED)
   47:                 self.isMouseEnabled = YES;
   48:         self.isKeyboardEnabled= YES;
   49: #endif
   50:         inputMode = kDepthMode;
   51:
   52:         /*
   53:         self.xyzcoords = [CCLabelTTF labelWithString:@"Coords" fontName:@"Helvetica" fontSize:16.0];
   54:         self.xyzcoords.position = ccp(size.width / 2, 50);
   55:         [self addChild:self.xyzcoords];
   56:
   57:         [[NSNotificationCenter defaultCenter] addObserver:self
   58:                                                  selector:@selector(handleHUDCoordUpdate:)
   59:                                                      name:@"CoordHUDUpdate"
   60:                                                    object:nil];
   61:
   62:          */
   63:
   64:         }
   65:         return self;
   66: }
   67:
   68:
   69: //Add the sprite hud
   70: - (LPTool*)addLPTool:(CGPoint)p objectID:(NSString*)objectID withBrushName:(NSString*)brushname{
   71:
   72:         LPTool *sprite = [LPTool spriteWithFile:brushname];
   73:
   74:     [self addChild:sprite];
   75:
   76:     sprite.updated = TRUE;
   77:     sprite.toolID = objectID;
   78:     [sprite setScale:lastScale];
```

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

    return sprite;
}

/* Tool Moved */
- (void)toolMoved:(CGPoint)point toolID:(NSString*)toolid{

    if (primaryTool == nil){
        [self startTrackingTool:point toolID:toolid];
    }else{
        [self moveTrackingTool:point toolID:toolid];
    }
}

/* Start Tracking Tool */
- (void)startTrackingTool:(CGPoint)point toolID:(NSString*)toolid{
    if (primaryTool == nil){
        primaryTool = [self addLPTool:point objectID:toolid withBrushName:lastBrush];
    }
}

/* Move Tracking Tool*/
- (void)moveTrackingTool:(CGPoint)point toolID:(NSString*)toolid{

    //Create tool if it does not exist
    if (primaryTool == nil){
        primaryTool = [self addLPTool:point objectID:toolid withBrushName:lastBrush];
    }else{
        //Update since it does exist
        primaryTool.position =  point;
        if ([toolid isNotEqualTo:primaryTool.toolID]){
            primaryTool.toolID = toolid;
        }
    }
}

/* End Trackingn Tool */
- (void)endTrackingTool{
    if (primaryTool != nil){
        [self removeChild:primaryTool cleanup:YES];
        primaryTool = nil;
    }
}


//Key up event
-(BOOL) ccKeyUp:(NSEvent*)event{

    unichar ch = [event keyCode];

    if (inputMode == kPressKeyMode){
        if ( ch == 49){
            [self.delegate painting:FALSE];
        }
    }

    if ( ch == 18){
        //change to space bar press mode
        inputMode = kPressKeyMode;
        [self.delegate changeMode:inputMode];
    }else if(ch == 19){
        //Change to depth mode
        inputMode = kDepthMode;
        [self.delegate changeMode:inputMode];
    }

    return YES;
}
//Key down event
-(BOOL) ccKeyDown:(NSEvent*)event{
    unichar ch = [event keyCode];

    if (inputMode == kPressKeyMode){
        if ( ch == 49){
            [self.delegate painting:TRUE];
        }
    }
    return YES;
```

```
159: }
160:
161: - (void)changeColor:(ccColor3B)color{
162:
163:
164:
165:     if(primaryTool != nil){
166:
167:         [primaryTool setColor:color];
168:
169:     }
170:     lastColor = color;
171: }
172:
173: - (void)changeBrush:(NSString*)brushname{
174:
175:     lastBrush = brushname;
176:     if (primaryTool != nil){
177:         //Save important data
178:         CGPoint lastlocation = primaryTool.position;
179:         NSString* toolid = [primaryTool.toolID copy];
180:
181:         //Remove Tool
182:         [self removeChild:primaryTool cleanup:YES];
183:
184:         //Add it back
185:         primaryTool = [self addLPTool:lastlocation objectID:toolid withBrushName:lastBrush];
186:     }
187:
188: }
189:
190:
191: - (void)changeScale:(float)size{
192:
193:     lastScale = size;
194:     if(primaryTool != nil){
195:
196:         [primaryTool setScale:size];
197:     }
198: }
199:
200:
201: - (void)erasingMode:(BOOL)mode{
202:
203:     eraseMode = mode;
204:
205:     //turn Erasing Mode on
206:     if (mode){
207:         previousColor = lastColor;
208:         lastColor = ccRED;
209:         [primaryTool setColor:ccRED];
210:     }else{
211:         //Turn erasing mode off
212:         lastColor = previousColor;
213:         [primaryTool setColor:lastColor];
214:     }
215:
216: }
217:
218: - (void)updateCoordsHUDWithX:(float)x withY:(float)y withZ:(float)z{
219:
220:     self.xyzcoords.string = [NSString stringWithFormat:@"Coords x: %0.0f, y %0.0f, z %0.0f",x,y,z];
221:
222: }
223:
224: - (void)handleHUDCoordUpdate:(id)sender{
225:
226:     NSNotification* note = sender;
227:     //LEDColor* ledColor = note.object;
228:
229:     SimplePoint* point = note.object;
230:
231:     [self updateCoordsHUDWithX:point.x withY:point.y withZ:point.z];
232:
233:
234: }
235:
236:
237: @end
```

```
 1: //
 2: //  LPCCControlButtonVariableSize.h
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 4/9/13.
 6: //
 7: //
 8: #import "cocos2d.h"
 9: #import "CCControlExtension.h"
10: /**
11:  LPCCControlButtonVariableSize Extends CCLayer to have a customizable control button interface
12:  */
13: @interface LPCCControlButtonVariableSize : CCLayer
14:
15: /** Creates and return a button with a default background and title color. */
16: - (CCControlButton *)standardButtonWithTitle:(NSString *)title;
17: @end
```

```
    1: //
    2: //  LPCCControlButtonVariableSize.m
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 4/9/13.
    6: //
    7: //
    8:
    9: #import "LPCCControlButtonVariableSize.h"
   10:
   11: @implementation LPCCControlButtonVariableSize
   12: - (id)init
   13: {
   14:     if ((self = [super init]))
   15:     {
   16:         CGSize screenSize = [[CCDirector sharedDirector] winSize];
   17:
   18:         // Defines an array of title to create buttons dynamically
   19:         NSArray *stringArray = [NSArray arrayWithObjects:@"Hello",@"Variable",@"Size",@"!", nil];
   20:
   21:         CCNode *layer = [CCNode node];
   22:         [self addChild:layer z:1];
   23:
   24:         double total_width = 0, height = 0;
   25:
   26:         // For each title in the array
   27:         for (NSString *title in stringArray)
   28:         {
   29:             // Creates a button with this string as title
   30:             CCControlButton *button = [self standardButtonWithTitle:title];
   31:             [button setPosition:ccp (total_width + button.contentSize.width / 2, button.contentSize.he
ight / 2)];
   32:             [layer addChild:button];
   33:
   34:             // Compute the size of the layer
   35:             height = button.contentSize.height;
   36:             total_width += button.contentSize.width;
   37:         }
   38:
   39:         [layer setAnchorPoint:ccp (0.5, 0.5)];
   40:         [layer setContentSize:CGSizeMake(total_width, height)];
   41:         [layer setPosition:ccp(screenSize.width / 2.0f, screenSize.height / 2.0f)];
   42:
   43:         // Add the black background
   44:         CCScale9Sprite *background = [CCScale9Sprite spriteWithFile:@"buttonBackground.png"];
   45:         [background setContentSize:CGSizeMake(total_width + 14, height + 14)];
   46:         [background setPosition:ccp(screenSize.width / 2.0f, screenSize.height / 2.0f)];
   47:         [self addChild:background];
   48:     }
   49:     return self;
   50: }
   51:
   52: #pragma mark -
   53: #pragma CCControlButtonTest_HelloVariableSize Public Methods
   54: #pragma CCControlButtonTest_HelloVariableSize Private Methods
   55:
   56: - (CCControlButton *)standardButtonWithTitle:(NSString *)title{
   57:     /** Creates and return a button with a default background and title color. */
   58:     CCScale9Sprite *backgroundButton = [CCScale9Sprite spriteWithFile:@"button.png"];
   59:     CCScale9Sprite *backgroundHighlightedButton = [CCScale9Sprite spriteWithFile:@"buttonHighlighted.p
ng"];
   60:     CCLabelTTF *titleButton = [CCLabelTTF labelWithString:title fontName:@"Marker Felt" fontSize:30];
   61:     [titleButton setColor:ccc3(159, 168, 176)];
   62:     CCControlButton *button = [CCControlButton buttonWithLabel:titleButton backgroundSprite:background
Button];
   63:     [button setBackgroundSprite:backgroundHighlightedButton forState:CCControlStateHighlighted];
   64:     [button setTitleColor:ccWHITE forState:CCControlStateHighlighted];
   65:
   66:     return button;
   67: }
   68:
   69:
   70: @end
```

```
    1: //
    2: //  LPLine.h
    3: //  LeapPuzz
    4: //
    5: //  Created by cj on 4/2/13.
    6: //
    7: //
    8:
    9: #import <Foundation/Foundation.h>
   10: /**
   11:  LPLine is tracks the points in one line from beginning to end
   12:  */
   13: @interface LPLine : NSObject {
   14:
   15: }
   16:
   17: @property (nonatomic, strong) NSMutableArray* points;  /**< points is a an array of points for the lin
e */
   18: @property (nonatomic, readwrite) float width;    /**< width is a constant width for the line */
   19:
   20: @end
```

```
 1: //
 2: //  LPLine.m
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 4/2/13.
 6: //
 7: //
 8:
 9: #import "LPLine.h"
10:
11: @implementation LPLine
12:
13: @synthesize points;
14: @synthesize width;
15:
16: - (id)init
17: {
18:         if ((self = [super init]) == nil) {
19:         self.points = [[NSMutableArray alloc] init];
20:                 self.width = 1.0f;
21:         }
22:     return self;
23: }
24:
25: @end
```

```objc
//
//  LPPoint.h
//  LeapPaint
//
//  Created by cj on 5/8/13.
//  Copyright (c) 2013 cjdesch. All rights reserved.
//

#import <Foundation/Foundation.h>



/**
 LPLinePoint is a plotted point for drawing onto the canvas
 */
@interface LPLinePoint : NSObject

@property (nonatomic, readwrite) float x; /**< x coordinate */
@property (nonatomic, readwrite) float y; /**< y coordinate */
@property (nonatomic, readwrite) float width; /**< width of the point */

/**
 * Init constructor with existing point to create with no width
 * @param p an point (x,y)
 * @return object instance
 */
- (id)initWithPosition:(CGPoint)p;
/**
 * Init constructor with x and y values with no width
 * @param xVal coordinate value
 * @param yVal coordinate value
 * @return object instance
 */
- (id)initWithX:(float)xVal withY:(float)yVal;
/**
 * Init constructor with existing point with width
 * @param p a point (x,y)
 * @param wVal width of the point
 * @return object instance
 */
- (id)initWithPosition:(CGPoint)p withWidth:(float)wVal;
/**
 * Init constructor with x and y values with width
 * @param xVal coordinate value
 * @param yVal coordinate value
 * @param wVal width of the point
 * @return object instance
 */
- (id)initWithX:(float)xVal withY:(float)yVal withWidth:(float)wVal;

/**
 * Returns point based on x and y
 * @return CGPoint
 */
- (CGPoint)point;
@end
```

```
 1: //
 2: //  LPPoint.m
 3: //  LeapPaint
 4: //
 5: //  Created by cj on 5/8/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import "LPLinePoint.h"
10:
11: @implementation LPLinePoint
12: @synthesize x;
13: @synthesize y;
14: @synthesize width;
15:
16: /** init 2d point with CGPoint */
17: - (id)initWithPosition:(CGPoint)p{
18:     if (self = [super init]) {
19:
20:         self.x = p.x;
21:         self.y = p.y;
22:         self.width = 0.0f;
23:
24:
25:     }
26:     return self;
27: }
28: /** Init Point with 2 separate values */
29: - (id)initWithX:(float)xVal withY:(float)yVal{
30:     if (self = [super init]) {
31:
32:         self.x = xVal;
33:         self.y = yVal;
34:         self.width = 0.0f;
35:     }
36:     return self;
37: }
38: /** Init  point with CGPoint and width Value */
39: - (id)initWithPosition:(CGPoint)p withWidth:(float)wVal{
40:     if (self = [super init]) {
41:         self.x = p.x;
42:         self.y = p.y;
43:         self.width = wVal;
44:     }
45:     return self;
46: }
47: /** Init Point with x and y values with width*/
48: - (id)initWithX:(float)xVal withY:(float)yVal withWidth:(float)wVal{
49:     if (self = [super init]) {
50:         self.x = xVal;
51:         self.y = yVal;
52:         self.width = wVal;
53:     }
54:     return self;
55: }
56: /** Return the CGPoint type from the object */
57: - (CGPoint)point{
58:     return CGPointMake(self.x, self.y);
59: }
60: @end
```

```
    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:
   12: /**
   13:  Extends CCSprite object with two properties for tracking sprites with pointable objects
   14:  */
   15: @interface LPTool : CCSprite
   16: @property (nonatomic, strong) NSString* toolID; /**< toolID is the ID number assigned by the LeapMotio
n SDK */
   17: @property (nonatomic, readwrite) BOOL updated; /**< updated is if the sprite has been updated in that
frame.*/
   18: @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: @end
```

```
 1: //
 2: //  main.m
 3: //  LeapPaint
 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: //  SimplePoint.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: /**
12:  * 2D or 3D space coordinate for temporarily maniulapting points
13:  *
14:  */
15: @interface SimplePoint : NSObject
16: @property (nonatomic, readwrite) float x; /**< x coordinate */
17: @property (nonatomic, readwrite) float y; /**< y coordinate */
18: @property (nonatomic, readwrite) float z; /**< z coordinate */
19: @property (nonatomic, readwrite) BOOL is3d; /**< is3d is 2d or 3d point type */
20: /**
21:  * Init constructor with existing point to create a 2d Point
22:  * @param p an point (x,y)
23:  * @return object instance
24:  */
25: - (id)initWithPosition:(CGPoint)p;
26: /**
27:  * Init constructor with x and y values to create a 2d point
28:  * @param xVal coordinate value
29:  * @param yVal coordinate value
30:  * @return object instance
31:  */
32: - (id)initWithX:(float)xVal withY:(float)yVal;
33: /**
34:  * Init constructor with existing point to create a 3d Point
35:  * @param p a point (x,y)
36:  * @param zVal coordinateValue
37:  * @return object instance
38:  */
39: - (id)initWithPosition:(CGPoint)p withZ:(float)zVal;
40: /**
41:  * Init constructor with x, y and z values to create 3D point
42:  * @param xVal coordinate value
43:  * @param yVal coordinate value
44:  * @param zval coordinate value
45:  * @return object instance
46:  */
47: - (id)initWithX:(float)xVal withY:(float)yVal withZ:(float)zVal;
48:
49: /**
50:  * Returns point based on x and y
51:  * @return CGPoint
52:  */
53: - (CGPoint)point;
54: @end
```

```
 1: //
 2: //  SimplePoint.m
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 2/19/13.
 6: //
 7: //
 8:
 9: #import "SimplePoint.h"
10:
11: @implementation SimplePoint
12:
13: @synthesize x,y,z;
14: @synthesize is3d;
15:
16:
17: /** init 2d point with CGPoint */
18: - (id)initWithPosition:(CGPoint)p{
19:     if (self = [super init]) {
20:
21:         self.x = p.x;
22:         self.y = p.y;
23:         self.z = 0.0f;
24:         self.is3d = false;
25:
26:     }
27:     return self;
28: }
29:
30: /** Init 2d Point with 2 separate values */
31: - (id)initWithX:(float)xVal withY:(float)yVal{
32:
33:     if (self = [super init]) {
34:
35:         self.x = xVal;
36:         self.y = yVal;
37:         self.z = 0.0f;
38:         self.is3d = false;
39:
40:     }
41:     return self;
42: }
43:
44:
45: /** Init 3d point with CGPoint and z Value */
46: - (id)initWithPosition:(CGPoint)p withZ:(float)zVal{
47:     if (self = [super init]) {
48:
49:         self.x = p.x;
50:         self.y = p.y;
51:         self.z = zVal;
52:         self.is3d = true;
53:
54:     }
55:     return self;
56: }
57:
58:
59: /** Init 3d Point with 3 separate values */
60: - (id)initWithX:(float)xVal withY:(float)yVal withZ:(float)zVal{
61:
62:     if (self = [super init]) {
63:
64:         self.x = xVal;
65:         self.y = yVal;
66:         self.z = zVal;
67:         self.is3d = true;
68:
69:     }
70:     return self;
71: }
72:
73: /** Return the CGPoint type from the object */
74: - (CGPoint)point{
75:     return CGPointMake(self.x, self.y);
76: }
77:
78:
79: @end
```

```
 1: //
 2: //  Point.h
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 2/19/13.
 6: //
 7: //
 8:
 9: #import <Foundation/Foundation.h>
10:
11: /**
12:  * 2D space coordinate for temporarily maniulapting points
13:  *
14:  */
15: @interface SimplePointObject : NSObject
16:
17: @property (nonatomic, readwrite) CGPoint point; /**< point is the X and Y coordinates */
18: /**
19:  * Init constructor with existing point to create a 2d Point
20:  * @param p an point consisting of (x,y)
21:  * @return object instance
22:  */
23: - (id)initWithPosition:(CGPoint)p;
24: /**
25:  * Init constructor with existing point to create a 2d Point
26:  * @param x is x axis coordinate
27:  * @param y is y axis coordinate
28:  * @return object instance
29:  */
30: - (id)initWithX:(float)x withY:(float)y;
31:
32: @end
```

```
 1: //
 2: //  Point.m
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 2/19/13.
 6: //
 7: //
 8:
 9: #import "SimplePointObject.h"
10:
11: @implementation SimplePointObject
12:
13:
14: - (id)initWithPosition:(CGPoint)p{
15:     if (self = [super init]) {
16:
17:         self.point = p;
18:
19:     }
20:     return self;
21: }
22:
23: - (id)initWithX:(float)x withY:(float)y{
24:
25:     if (self = [super init]) {
26:
27:         self.point = CGPointMake(x, y);
28:
29:     }
30:     return self;
31: }
32: @end
```

```
 1: //
 2: //  SketchRenderTextureScene.h
 3: //  Cocos2D-CCRenderTexture-Demo
 4: //
 5: //  Copyright (c) 2011 Steffen Itterheim.
 6: //      Distributed under MIT License.
 7: //
 8:
 9: #import "cocos2d.h"
10: #import "SimplePoint.h"
11: #import "GameSettings.h"
12: @interface SketchRenderTextureScene : CCLayer
13: {
14:         CCSprite* brush;
15:         NSMutableArray* touches;
16:
17:     ccColor3B lastColor;
18:     ccColor3B previousColor;
19:     NSString* lastBrush;
20:     float lastScale;
21:     bool eraseMode;
22: }
23:
24: @property (nonatomic,readwrite) float opacity;
25:
26: - (void)beginDraw:(CGPoint)point withZ:(float)z;
27: - (void)updateDraw:(CGPoint)point withZ:(float)z;
28: - (void)endDraw:(CGPoint)point;
29: - (void)changeColor:(ccColor3B)color;
30: - (void)changeBrush:(NSString*)brushname;
31: - (void)changeScale:(float)size;
32: - (void)changeOpacity:(float)o;
33: - (void)erasingMode:(BOOL)mode;
34: - (void)clearDrawing;
35: @end
```

```
    1: //
    2: //  SketchRenderTextureScene.m
    3: //  Cocos2D-CCRenderTexture-Demo
    4: //
    5: //  Copyright (c) 2011 Steffen Itterheim.
    6: //      Distributed under MIT License.
    7: //
    8:
    9: #import "SketchRenderTextureScene.h"
   10:
   11:
   12: @implementation SketchRenderTextureScene
   13: @synthesize opacity;
   14:
   15: -(id) init
   16: {
   17:         if ((self = [super init]))
   18:         {
   19:                 // create a simple rendertexture node and clear it with the color white
   20:
   21:         //target = [CCRenderTexture renderTextureWithWidth:s.width height: s.height pixelFormat:kCCTex
ture2DPixelFormat_RGBA8888];
   22:                 CGSize s = [CCDirector sharedDirector].winSize;
   23:
   24:         CCDirector* sharedDirector =[CCDirector sharedDirector];
   25:         CGSize frameSize = sharedDirector.view.frame.size;
   26:
   27:
   28:
   29:         float topbottombar = 300;
   30:         float sidebars = 300;
   31:
   32:
   33:
   34:
   35:         //CCSprite* imageBackground = [CCSprite spriteWithFile:@"squarebrush.png"] ;
   36:         //imageBackground set
   37:
   38:
   39:         CCRenderTexture* rtx = [CCRenderTexture renderTextureWithWidth:frameSize.width-sidebars height
: frameSize.height-topbottombar];
   40:                 [rtx clear:1.0f
   41:                                 g:1.0f
   42:                                 b:1.0f
   43:                                 a:1.0f];
   44:
   45:                 rtx.position = CGPointMake(s.width/2, s.height/2);
   46:                 [self addChild:rtx z:0 tag:1];
   47:
   48:
   49:
   50:
   51:                 //CCLabelTTF* label = [CCLabelTTF labelWithString:@"Drawing onto CCRenderTexture witho
ut clear" fontName:@"Arial" fontSize:16];
   52:                 //label.position = CGPointMake(240, 15);
   53:                 //label.color = ccGRAY;
   54:                 //[self addChild:label];
   55:
   56:                 // create and retain the brush sprite, but don't add it as child
   57:
   58:         lastColor = ccWHITE;
   59:         lastBrush = @"roundbrush.png";
   60:         lastScale = 1.0;
   61:
   62:         eraseMode = false;
   63:         self.opacity = 10;
   64:
   65:         [self addBrush:lastBrush];
   66:
   67:
   68:
   69:                 //brush.scale = 0.5f;
   70:
   71:                 // create the array holding the touches
   72:                 touches = [[NSMutableArray alloc] init];
   73:
   74:                 //[CCTouchDispatcher sharedDispatcher] addTargetedDelegate:self priority:0 swallowsTou
ches:NO];
   75:
   76:                 [self scheduleUpdate];
```

```
 77:
 78:         }
 79:         return self;
 80: }
 81:
 82: - (void)addBrush:(NSString*)brushName{
 83:
 84:     brush = [CCSprite spriteWithFile:brushName] ;
 85:     [brush setScale:lastScale];
 86:
 87:
 88:     if(eraseMode){
 89:         //[brush setBlendFunc:(ccBlendFunc) { GL_ZERO,GL_ONE_MINUS_SRC_ALPHA }];
 90:         [brush setBlendFunc:(ccBlendFunc) { GL_ONE,GL_ONE }];
 91:
 92:
 93:
 94:                 [brush setOpacity:80];
 95:     }else{
 96:         brush.color = lastColor;
 97:         brush.opacity = opacity;
 98:     }
 99: }
100:
101: -(void) cleanup
102: {
103:         brush = nil;
104:         touches = nil;
105:
106:         [super cleanup];
107: }
108:
109: - (void)beginDraw:(CGPoint)point withZ:(float)z{
110:     //NSLog(@"Begin Draw");
111:     SimplePoint* simplePoint = [[SimplePoint alloc] initWithPosition:point withZ:z];
112:     [touches addObject:simplePoint];
113:
114: }
115: - (void)updateDraw:(CGPoint)point withZ:(float)z{
116:
117:       //  NSLog(@"update Draw");
118:     SimplePoint* simplePoint = [[SimplePoint alloc] initWithPosition:point withZ:z];
119:     [touches addObject:simplePoint];
120:
121: }
122: - (void)endDraw:(CGPoint)point {
123:         [touches removeAllObjects];
124: }
125:
126:
127: /*
128: -(BOOL) ccTouchBegan:(UITouch *)touch withEvent:(UIEvent *)event
129: {
130:         // add new touches to the array as they come in
131:         [touches addObject:touch];
132:         return YES;
133: }
134:
135: -(void) ccTouchEnded:(UITouch *)touch withEvent:(UIEvent *)event
136: {
137:         // must remove the touches that have ended or where cancelled
138:         [touches removeObject:touch];
139: }
140:
141: -(void) ccTouchCancelled:(UITouch *)touch withEvent:(UIEvent *)event
142: {
143:         [self ccTouchEnded:touch withEvent:event];
144: }
145:
146: */
147: -(void) setBrushColor:(int)color
148: {
149:         switch (color)
150:         {
151:                 default:
152:                 case 0:
153:                         brush.color = ccWHITE;
154:                         break;
155:                 case 1:
156:                         brush.color = ccGREEN;
```

```
  157:                         break;
  158:                 case 2:
  159:                         brush.color = ccRED;
  160:                         break;
  161:                 case 3:
  162:                         brush.color = ccc3(0, 255, 255);
  163:                         break;
  164:                 case 4:
  165:                         brush.color = ccBLUE;
  166:                         break;
  167:         }
  168: }
  169:
  170: -(void) update:(ccTime)delta
  171: {
  172:
  173:         CCRenderTexture* rtx = (CCRenderTexture*)[self getChildByTag:1];
  174:
  175:         // explicitly don't clear the rendertexture
  176:         [rtx begin];
  177:
  178:         //int color = 0;
  179:
  180:         // Since we store all current touches in an array, we can render a sprite at each touch locati
on
  181:         // even if the touch isn't moving. That way a continued press will increase the opacity of the
 sprite
  182:         // simply because the sprite is drawn repeatedly with low opacity at the same location.
  183:     NSArray* tempTouches = [[NSArray alloc] initWithArray:touches];
  184:         for (SimplePoint* touch in tempTouches)
  185:         {
  186:                 //CGPoint touchLocation = [director convertToGL:[touch locationInView:director.openGLV
iew]];
  187:         CGPoint touchLocation = [touch point];
  188:
  189:                 // the location must be converted to the rendertexture sprite's node space
  190:                 touchLocation = [rtx.sprite convertToNodeSpace:touchLocation];
  191:
  192:                 // because the rendertexture sprite is flipped along its Y axis the Y coordinate must
be flipped:
  193:                 touchLocation.y = rtx.sprite.contentSize.height - touchLocation.y;
  194:
  195:                 //CCLOG(@"touch: %.0f, %.0f", touchLocation.x, touchLocation.y);
  196:
  197:                 // set the brush at that location and render it
  198:                 brush.position = touchLocation;
  199:                 //[self setBrushColor:color++];
  200:                 [brush visit];
  201:         }
  202:
  203:
  204:
  205:         [rtx end];
  206:
  207:     [touches removeAllObjects];
  208: }
  209:
  210: - (void)changeColor:(ccColor3B)color{
  211:
  212:
  213:     if(brush != nil){
  214:
  215:         brush.color = color;
  216:
  217:     }
  218:     lastColor = color;
  219:
  220: }
  221: - (void)changeBrush:(NSString*)brushname{
  222:
  223:     lastBrush = brushname;
  224:     if (brush != nil){
  225:         //Save important data
  226:         CGPoint lastlocation = brush.position;
  227:         [self addBrush:lastBrush];
  228:         brush.position = lastlocation;
  229:     }
  230:
  231: }
  232:
```

```
233: - (void)changeScale:(float)size{
234:
235:     lastScale = size;
236:     if(brush != nil){
237:
238:         [brush setScale:size];
239:
240:     }
241: }
242:
243: - (void)changeOpacity:(float)o{
244:
245:     self.opacity = o;
246:     if (brush != nil){
247:
248:         brush.opacity = self.opacity;
249:     }
250:
251: }
252:
253: - (void)clearDrawing{
254:
255:         CCRenderTexture* rtx = (CCRenderTexture*)[self getChildByTag:1];
256:
257:         // explicitly don't clear the rendertexture
258: //      [rtx begin];
259: //    glClearColor(r, g, b, a);
260:   //  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
261:         //get rid of the mask
262:         // glColorMask(TRUE, TRUE, TRUE, FALSE);
263: //    [rtx end];
264:
265:     [rtx clear:1 g:1 b:1 a:0];
266:
267: }
268:
269:
270:
271: - (void)erasingMode:(BOOL)mode{
272:
273:     eraseMode = mode;
274:
275:     //turn Erasing Mode on
276:     if (mode){
277:         previousColor = lastColor;
278:         lastColor = ccRED;
279:
280:         CGPoint lastlocation = brush.position;
281:         [self addBrush:lastBrush];
282:         brush.position = lastlocation;
283:
284:     }else{
285:         //Turn erasing mode off
286:         lastColor = previousColor;
287:         CGPoint lastlocation = brush.position;
288:         [self addBrush:lastBrush];
289:         brush.position = lastlocation;
290:
291:     }
292:
293: }
294:
295:
296:
297:
298:
299: @end
```

```
 1: //
 2: //  Utility.h
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 4/24/13.
 6: //
 7: //
 8:
 9: #import <Foundation/Foundation.h>
10:
11:
12:
13: /**
14:  Utility class provides common usage function throughout the application.
15:  */
16:
17: @interface Utility : NSObject {
18: }
19:
20: /**
21:  Generates a random number between two designated integers
22:  @param from is the bottom of the range
23:  @param to is the top of the range
24:  @return a random number between the from and to parameters
25:  */
26: + (int)getRandomNumberBetween:(int)from to:(int)to;
27: /**
28:  Generates a random number between 0 designated integer
29:  @param to is the top of the range
30:  @return a random number between 0 and to parameters
31:  */
32: + (int)getRandomUniformNumberUnder:(int)to;
33: /**
34:  Generates a random number between 0 designated integer
35:  @param to is the top of the range
36:  @return a random number between 0 and to parameters
37:  */
38: + (int)getRandomNumberUnder:(int)to;
39: //- (void) initRandomSeed(long firstSeed);
40: //float nextRandomFloat();
41: @end
```

```
 1: //
 2: //  Utility.m
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 4/24/13.
 6: //
 7: //
 8:
 9: #import "Utility.h"
10:
11: @implementation Utility
12:
13: /** returns random number within a range with defined upper and lower bounds */
14: + (int)getRandomNumberBetween:(int)from to:(int)to {
15:
16:     //Check that one isn't greater than the other
17:     //if so, flip them
18:
19:     return (int)from + arc4random() % (to-from+1);
20: }
21:
22: /** Returns a random number from 0 to an upper bound */
23: + (int)getRandomNumberUnder:(int)to{
24:     return (arc4random() % to);
25: }
26:
27:
28: /** Returns a Uniform Random Number from 0 to an upper bound */
29: + (int)getRandomUniformNumberUnder:(int)to{
30:     //Check if uniform available
31:     if (arc4random_uniform != NULL)
32:         return  arc4random_uniform (to);
33:     else
34:         return  (arc4random() % to);
35: }
36:
37:
38:
39:
40:
41: /*
42: static unsigned long seed;
43:
44: void initRandomSeed(long firstSeed)
45: {
46:     seed = firstSeed;
47: }
48:
49: float nextRandomFloat()
50: {
51:     return (((seed= 1664525*seed + 1013904223)>>16) / (float)0x10000);
52: }*/
53:
54:
55:
56: @end
```