Design Standard for IdeaStorm

Table of Contents

1 INTRODUCTION 3

2 Drawing Engine Design 3

2.1 Classes 3

2.1.1 AppDelegate 3

2.1.2 Brush 3

2.1.3 DrawingColor 4

2.1.4 DrawingEngine 4

2.1.5 DrawingTool 5

2.1.6 DrawingViewController 5

2.1.7 EraserDrawingTool 6

2.1.8 GLProgram 6

2.1.9 GLView 6

2.1.10 PencilDrawingTool 8

2.1.11 PenDrawingTool 8

2.1.12 Toolbar 8

2.1.13 ToolbarItem 9

2.1.14 ToolSet 9

2.1.15 TutorialOverlay 10

2.1.16 Vertex and Fragment Shaders 10

2.2 Algorithms 10

2.2.1 Point Sprites 10

2.2.2 Curve Interpolation 11

2.2.2.1 Calculating the Control Points 11

2.2.2.2 Estimating the Curve Length 12

2.2.2.3 Calculating Curve Points 12

2.2.3 Changing Orientation 13

2.2.4 Saving Drawing Images 14

3 Gallery Design 15

3.1 Architectural Design 15

3.2 Component Design 15

3.2.1 Gallery View Components 15

3.2.1.1 GalleryView 16

3.2.1.2 GalleryViewDelegate 16

3.2.1.3 GalleryToolbar 17

3.2.1.4 GalleryToolbarDelegate 17

3.2.1.5 PositioningHelper 18

3.2.2 View Controller Components 19

3.2.2.1 GalleryViewController 19

3.2.3 Database Components 19

3.2.3.1 Database 20

3.2.3.2 GalleryItem 21

3.2.3.3 Stack 22

3.2.3.4 Drawing 23

4 TIME AND RESOURCE ESTIMATION 23

5 RISK ANALYSIS 24

5.1 Technical Risks 24

5.2 Schedule Risk 25

6 LIST OF CODE AND DOCUMENTS TO BE ADDED/MODIFIED 25

6.1 Classes 25

6.2 Documentation 25

7 References 26

# INTRODUCTION

This document is based on the Analysis report outlined for GUMP Short created by [Pinto] and has been adapted to be used exclusively as a design document. This document outlines the design for IdeaStorm in the SRS for IdeaStorm document (cole\_IdeaStorm\_srs\_iter03.docx).

This document is not intended to provide the complete design for IdeaStorm. Instead it will provide a brief explanation of the general design. This document will also provide more detailed design descriptions for implementing important or difficult to implement requirements.

This document will also include a description of the estimated time & resources, risks and documents that will be involved. The last section of this document also includes a list of references that were used in this documents creation.

This document will reference requirements in the SRS document by their requirement number. This is typically followed by the title of the requirement in parentheses, for example, EXR001 (Example Requirement).

# Drawing Engine Design

## Classes

This section provides a description of the major classes that will be created to make the IdeaStorm drawing engine. Each subsection provides a brief description of the class. Also, some class descriptions include additional sections that describe the data structures, properties and methods that must be included in the class. However, classes are not limited to using only these data structures, properties or methods and the developer may choose to implement any other part of these classes at their discretion as long as the requirements in the SRS are met.

### AppDelegate

The AppDelegate class will be used to initialize the DrawingViewController and the GalleryViewController. The AppDelegate will set the GalleryViewController as its root view controller.

Properties:

* drawingViewController – This is the DrawingViewController that will be used to display the drawing area and receive touch input from the app user.
* galleryViewController – This is the GalleryViewController that will be used to display all gallery items to the app user.

### Brush

The Brush class is a subclass of ToolbarItem. The Brush class stores the texture filename to be used when drawing.

Properties:

* textureFilename – The filename of the texture to be used for drawing.

Methods:

* - (id)initWithTexture:(NSString \*)textureFilename – Initializes a Brush object and sets the textureFilename property to the NSString provided.

### DrawingColor

The DrawingColor class is a subclass of ToolbarItem. The DrawingColor class stores the color that will be sent to the DrawingTool for Vertex creation.

Properties:

* color – The Color that will be sent to the DrawingTool for Vertex creation.

Methods:

* - (id)initWithColor:(Color)color – Initializes a DrawingColor object and sets the color property to the Color provided.

### DrawingEngine

When the DrawingEngine is initialized, it will also initialize its renderView property with a GLView object. The DrawingEngine will be responsible for creating a default tool set for these properties in case they are not otherwise defined.

After initialization, the DrawingEngine will receive touch data from the DrawingViewController. When the DrawingEngine receives the touch data it will send the position data from the touch data, the drawing color and the point size from its activeToolSet property to the drawing tool in its activeToolSet property to create a Vertex array. The DrawingEngine will then send the Vertex array to its renderView property to render to the screen.

The DrawingEngine will also record the touch positions, colors, brushes and drawing tools to the stroke property of the drawing property.

Properties:

* activeToolSet – This is the tool set that the app user is currently using for drawing.
* reserveToolSet – This is the tool set that the app user can switch to as an alternative tool set for drawing.
* renderView – The GLView object that will render and display the app user’s drawing.
* database – The Database that all Drawings will be saved to.
* drawingTool – The DrawingTool object that contains the Brush and drawing color to use when drawing.
* drawing – The drawing that is currently loaded or the drawing that is in progress.

Methods:

* - (id)init – This will initialize a default DrawingEngine that has a GLView set to the size of the current screen.
* - (id)initWithFrame:(CGRect)frame – This will initialize a DrawingEngine with a GLView set to the size and position of the CGRect provided.
* - (id)initWithFrame:(CGRect)frame andDatabase:(Database \*)database – This will initialize a DrawingEngine as stated for the initWithFrame: method, but will also set the database property to the Database provide.
* - (bool)saveCurrentDrawing – Saves the Drawing stored in the drawing property using the Database.
* - (void)newDrawingForStack:(Stack \*)stack – Creates a new drawing with the parent property set to stack specified.
* - (bool)saveAndAddDrawing – Saves the current drawing using the saveCurrentDrawing method and starts a new drawing to that drawing’s parent using the newDrawingForStack: method. This also clears the contents of the current drawing area.

### DrawingTool

The DrawingTool class will be a protocol that specifies the methods and properties that must be implemented for Vertex creation. All drawing tools that the app user can select for drawing will implement this protocol.

Methods:

* - (Vertex \*)verticesFromPoint:(CGPoint)point andDrawingColor:(Color)color andPointSize:(CGFloat)size isLastPoint:(bool)lastPoint – This method will take position, drawing color and point size data to create a Vertex array that can be utilized by the drawing engine for drawing. Also provided is a Boolean value that indicates the last position value of the current curve. The classes that implement the DrawingTool protocol can choose to manipulate or ignore any values provided to it and can return a Vertex array of any size, including an empty array or null array.

### DrawingViewController

The DrawingViewController is responsible for initializing and linking the major classes of the drawing engine. Within the DrawingViewContoller’s initialization, the following will occur:

1. Set the database property using the Database provided
2. Creating an instance of the DrawingEngine class using the database property
3. Creating an instance of the Toolbar class
4. Setting the Toolbar’s drawingEngine property to the DrawingEngine instance created
5. Adding the Toolbar instance created as a subview of the DrawingEngine’s renderView property
6. Adding the DrawingEngine’s renderView property as a subview of the DrawingViewController’s view property
7. Use the database property to load the any user defaults defined

The order above is a suggested order for initialization. These maybe adjusted to fit the needs of development.

The DrawingViewController will also handle all touch input by the app user. The touch data used for drawing will be passed to its drawingEngine property.

The DrawingViewController will also handle repositioning and adjusting all UI elements in the event that the iPad’s orientation changes. This will be handled using the algorithm described in section 2.2.3.

Properties:

* drawingEngine – This is the DrawingEngine responsible for taking touch data and transforming it into a drawing.
* toolbar – This is the Toolbar that will be displayed to allow the app user to change tools.
* database – This is the Database that will be responsible for saving and loading from the file system and the user defaults.

Methods:

* - (id)initWithNibName:(NSString \*)nibNameOrNil bundle:(NSBundle \*)nibBundleOrNil andDatabase:(Database \*)database – This will initialize the DrawingViewController as stated in the description above.

### EraserDrawingTool

This class is a subclass of Toolbaritem and will also implement the DrawingTool protocol. When the EraserDrawingTool receives the data from the verticesFromPoint:andDrawingColor:andPointSize:isLastPoint: method, it will use the point size provided as is when creating the Vertex data. The EraserDrawingTool will interpolate addition points along a curve using the points provided. This will use the algorithm described in 2.2.2. The position data from the interpolated points, the point size and a white drawing color [1.0, 1.0, 1.0, 1.0] will then be used to create a Vertex array that will be returned by the verticesFromPoint:andDrawingColor:andPointSize:isLastPoint: method.

### GLProgram

GLProgram is a class that was developed by [Lamarche]. This class simplifies loading, compiling and working with shaders in OpenGL ES 2.0. This class was slightly modified to allow for easier debugging. This modification includes adding additional error message outputs to the vertexShaderLog and fragmentShaderLog methods.

### GLView

GLView utilizes an OpenGL ES 2.0 setup for rendering the app user’s drawing to the screen. This allows the iPad to use its graphics hardware in parallel with the CPU in order to meet performance requirements PR001 (Drawing Response) and P002 (Erasing Response). There is a slight modification to the way that shaders are handled. Instead of manually coding the loading, compiling and handling of the shaders, the class GLProgram written by [Lamarche] will be used instead.

To render Vertex data that has been passed to it from the DrawingEngine, the DrawingEngine must call GLView’s render method.

To clear the screen of all rendered content, the clearScreen method must be called from the DrawingEngine.

Data Structures:

* Color – This data structure will be implemented to include the red, green, blue and alpha channels that define a color. Each channel will be defined using values that range from 0.0 to 1.0.

typedef struct {

GLfloat r;

GLfloat g;

GLfloat b;

GLfloat a;

} Color;

* Position – This data structure will be implemented to include the x and y coordinates that will be used in defining the position of a point sprite.

typedef struct {

GLfloat x;

GLfloat y;

} Position;

* Vertex – This data structure will be implemented to include position, drawing color and point size attributes that will be used when rendering point sprites. The value for point size is valid as long as it is greater then 0.0.

typedef struct {

Position position;

Color color;

GLfloat pointSize;

} Vertex;

Properties:

* vertexArray – This array stores all vertices that have not been drawn yet.

Methods:

* - (void)clearScreen – This method clears the drawing area to a blank white color.
* - (void)addVertex:(Vertex)vertex – This method will add a vertex to the vertexArray property to be rendered.
* - (void)setTexture:(NSString \*)textureFilename – This method will set the texture for drawing to the file specified by textureFilename.
* - (void)render – Renders the app user’s drawing on the iPad’s screen using the vertices provided.
* - (UIImage \*)getRenderedImage – Returns a UIImage that contains the drawing within the drawing area. This will utilize the algorithm outlined in section 2.2.4.

### PencilDrawingTool

This class is a subclass of Toolbaritem and will also implement the DrawingTool protocol. When the PencilDrawingTool receives the data from the verticesFromPoint:andDrawingColor:andPointSize:isLastPoint: method, it will use the drawing color as is when creating the Vertex data. The PenDrawingTool will interpolate addition position data along a curve using the position data provided. This will use the algorithm described in 2.2.2. The point size provided will be manipulated to become smaller in size as the distance between points provided become further apart along the curve. This point size will also approach the original size as the distance between points provided become closer together. The position data from the interpolated points, the manipulated point size and drawing color will then be used to create a Vertex array that will be returned by the verticesFromPoint:andDrawingColor:andPointSize:isLastPoint: method.

### PenDrawingTool

This class is a subclass of Toolbaritem and will also implement the DrawingTool protocol. When the PenDrawingTool receives the data from the verticesFromPoint:andDrawingColor:andPointSize:isLastPoint: method, it will use the drawing color and point size provided as is when creating the Vertex data. The PenDrawingTool will interpolate addition points along a curve using the points provided. This will use the algorithm described in 2.2.2. The position data from the interpolated points, the point size and drawing color will then be used to create a Vertex array that will be returned by the verticesFromPoint:andDrawingColor:andPointSize:isLastPoint: method.

### Toolbar

The Toobar will be a subclass of the UIView class. This class will contain the gallery button, quick switch button, drawing tool buttons, brush buttons, drawing color buttons and new drawing button as described in EIR012 (Drawing Toolbar), EIR022 (Quick Switch Button) and EIR011 (New Drawing Button).

The quick switch button and the new drawing button will be created and added as subviews of the Toolbar upon initialization. Drawing tool buttons, brush buttons and drawing color buttons will be added via the addToolbarItem: method.

When a drawing tool, brush, drawing color or quick switch button is select, the Toolbar will modify the activeToolSet property of its drawingEngine property to reflect this new selection fulfilling EIR025 (Changing the Tool Set).

Properties:

* drawingEngine – The DrawingEngine that the Toolbar will send new ToolSet selections to.
* buttons – A NSMutableDictionary containg the UIButtons that the Toolbar currently has added to its subview.
* toolbarItems – A NSMutableDictionary containing the ToolbarItems that Toolbar tracks.

Methods:

* - (NSInteger)addToolbarItem:(ToolbarItem \*)toolbarItem – This method will take a ToolbarItem and create a UIButton for it and add it as a subview of the Toolbar. For instances of DrawingTool, Brush and DrawingColor, it will also automatically add a action to the UIButton that will send a new selection for the ToolSet to the activeToolSet property of the DrawingEngine specified in the drawingEngine Property. This method will also return the NSInteger that is stored as the key for the corresponding ToolbarItem and UIButton in the toolbarItems and buttons properties. This NSInteger will also be set for tag property of the UIButton created.

### ToolbarItem

Other classes that will be placed into the Toolbar will subclass or create an instance of ToolbarItem. ToolbarItem will include an UIImage property that will be set as the image icon for the button that will represent the ToolbarItem. It will also include a title property that uniquely identifies the item being added to it.

Properties:

* icon – The UIImage that will be display in the button that represent the item select.
* title – The NSString title of the tool.

Methods:

* - (bool)setIconWithImageName:(NSString \*)imageFilename – Will load a png image from a filename into the icon property. This method will return YES if the file specified was loaded successfully.

### ToolSet

The ToolSet class contains data items that describe a tool set being used by the app user. This tool set includes the drawing tool, brush, drawing color and point size.

Properties:

* drawingTool – This is the DrawingTool object that is part of the tool set.
* brush – This is the Brush object that is part of the tool set.
* drawingColor – This is the DrawingColor object that is part of the tool set.
* pointSize – This is the point size expressed as a float that is part of the tool set.

Methods:

* - (id)initWithDrawingTool:(DrawingTool \*)drawingTool andBrush:(Brush \*)brush andDrawingColor:(DrawingColor \*)drawingColor andPointSize:(GLfloat)pointSize – Initializes the tool set with the provided drawing tool, brush, drawing color and point size.

### TutorialOverlay

The TutorialOverlay class will subclass the UIView class. This class will be used to display a tutorial image on top of the other content in an UIViewController object. This class will have four properties that define UIImage objects for display in each orientation. The changeToOrientation: method will also select which image is used for display when the TutorialOverlay is displayed.

Properties:

* portraitImage – An UIImage that is displayed in the portrait position.
* portraitUpsideDownImage – An UIImage that is displayed in the portrait upside down position.
* landscapeLeftImage – An UIImage that is displayed in the landscape right position.
* landscapeRightImage – An UIImage that is displayed in the landscape right position.

Methods:

* - (void)changeToOrientation:(UIInterfaceOrientation)orientation – Changes the image displayed based on the orientation provided.

### Vertex and Fragment Shaders

The basic algorithm that will be used for the Vertex and Fragment shaders is described in section 2.2.1. The vertex shader in this algorithm will be modified to take an attribute for the color and pass it to the fragment shader. The fragment shader will then be modified to apply this color to the texture to be rendered.

## Algorithms

### Point Sprites

Drawing in in the drawing engine will be achieved by rendering points on the screen that have the brush’s texture mapped to them. Textured points are commonly referred to as point sprites. This affect will use a custom OpenGL ES 2.0 vertex and fragment shader pair. Basic code that demonstrates how to construct shaders to render points sprites is demonstrated below. This is not the final code that will be used, instead this is an example of how part of the final code will be constructed. The example for this code was provided by [Lazuka].

VERTEX SHADER

attribute vec4 Position;

attribute float PointSize;

void main() {

gl\_PointSize = PointSize;

gl\_Position = Position;

}

FRAGMENT SHADER

uniform sampler2D Texture;

void main() {

gl\_FragColor = texture2D(Texture,gl\_PointCoord);

}

In this example, the position and size of the point to be rendered is passed into the vertex shader from GLView. These attributes are then passed to special OpenGL variables gl\_PointSize and gl\_Position to set the point size and position for the each point being rendered.

The textured to be rendered on every point from the vertex shader is passed from the GLView into the fragment shader. The shader then used the function texture2D() with the special OpenGL variable gl\_PointCoord to map the texture onto the point being rendered.

### Curve Interpolation

As stated in section 2.2.1, point sprites are points that have a texture mapped to them. To make a line out of these points, an algorithm will be used to interpolate points along a curved path, using the touch data that is gathered by the iPad as described in FR002 (Drawing a Line). This algorithm is explained in the proceeding subsections.

#### Calculating the Control Points

The first part of the curve interpolation algorithm is finding the control points needed for the Bezier curve algorithm. Given four consecutive touch points on a user drawn curve called A, B, C and D, the control points: CP1 and CP2 can be calculated for curve segment BC. This is illustrated in the figure below.

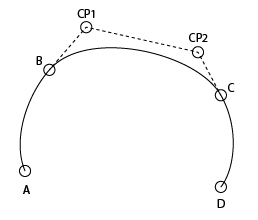


Figure 2‑1 – Points used to calculate control points CP1 and CP2.

Also, utilized in this algorithm is a curve coefficient. As this curve coefficient is increased towards 1, the curve applied using the Bezier curve algorithm becomes more pronounced. As the curve coefficient is decreases towards 0, the curve becomes less pronounced. When using this algorithm in the past, a curve coefficient of 0.15 produced a user perceivable smooth curve.

Below is the pseudocode for the algorithm that calculates the control points using the above elements mentioned. This code was derived from the forum post [Smooth line connecting CGPoints].

CP1.x = B.x + curveCoefficient \* (C.x - A.x);

CP1.y = B.x + curveCoefficient \* (C.y - A.y);

CP2.x = C.x - curveCoefficient \* (D.x - B.x);

CP2.y = C.x - curveCoefficient \* (D.y - B.y);

#### Estimating the Curve Length

The second part of the curve interpolation algorithm involves estimating the length of the curve. An estimation of the curves length is used as calculating the exact length of the curve may impact the processor of the iPad and effect drawing efficiency.

To estimate the length of a curve, the distance between several points along the curve will be calculated. The distance between these points will then be combined for a total estimation of curve length as shown below.

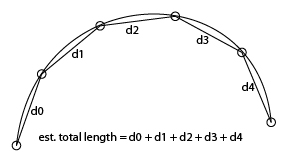


Figure 2‑2 – Estimating curve length using the distance between points on the curve.

The algorithm used for calculating the distance between these points is presented in pseudocode below. The method for developing this algorithm was derived from [Bullock].

distance = squareroot((point2.x – point1.x)2 + (point2.y – point1.y)2);

#### Calculating Curve Points

The third part of this algorithm uses the formula for Bezier curves to do the interpolation of curve points. This formula is demonstrated below in pseudocode. The main part of this algorithm is derived from the forum post [Drawing a curve in OpenGLES - how? - GameDev.net].

float t = 0 ;

CGPoint point;

for (int i=0; i<numSteps;i++) {

point.x = (1 – t)3 \* P0.x + 3 \* (1 – t)2 \* t \* P1.x + 3 \* (1 – t) \* t2 \* P2.x + t3 \* P3.x;

point.y = (1 – t)3 \* P0.y + 3 \* (1 – t)2 \* t \* P1.y + 3 \* (1 – t) \* t2 \* P2.y + t3 \* P3.y;

[pointArray addObject:point];

t += 1 / numSteps;

}

P0 and P3 represent the end points of the curve segment. P1 and P2 are the control points that were calculated in section 2.2.2.1.

The value t in this algorithm is incremented or stepped in fractions between 0 and 1 to generate points along the curve that will be drawn as point sprites. The number of steps to use will be based off the estimated length of the curve based off the algorithm in section 2.2.2.2. This is done so the distance between interpolated points can remain consistent between curve segments of various lengths.

### Changing Orientation

In the drawing engine, orientation changes will not be handled with the standard shouldAutorotateToInterfaceOrientation: method provided in the UIViewController class. Instead, orientation changes will be handled using a custom method. This is being done through a custom method to ensure that all UI elements can be repositioned, resized and rotated, with exception of the GLView. Also, the actions for each swipe gesture direction will be reoriented to the new orientation so it remains correct according to the app user’s orientation.

The DrawingViewController will listen for the orientation change event and call a method each time the orientation changes. This method will then get the device’s orientation and change each UI element to fit within the new orientation with exception of the GLView, which will remain in the same orientation and position. An example of this code is demonstrated below. This is not the final code that will be used; instead it only demonstrates the algorithm that will be used.

//added to the initialization method of the DrawingViewController

[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(didRotate:) name:UIDeviceOrientationDidChangeNotification object:nil];

//method that is part of the DrawingViewController

- (void)didRotate:(NSNotification \*)notification {

UIDeviceOrientation orientation = [[UIDevice currentDevice] orientation];

//change position, rotation and size of all UI elements except GLView

//change actions that will occur for swipe gesture recognizers based on direction

}

### Saving Drawing Images

To obtain a UIImage from the GLView’s drawing area, the code below will be utilized from [iPhone - GLPaint Save Image - Stack Overflow]. This code will be modified slightly including changing the method name and releasing the buffer and buffer2 variables so that memory leaks are not produced.

-(UIImage \*) saveImageFromGLView

{

NSInteger myDataLength = 320 \* 480 \* 4;

// allocate array and read pixels into it.

GLubyte \*buffer = (GLubyte \*) malloc(myDataLength);

glReadPixels(0, 0, 320, 480, GL\_RGBA, GL\_UNSIGNED\_BYTE, buffer);

// gl renders "upside down" so swap top to bottom into new array.

// there's gotta be a better way, but this works.

GLubyte \*buffer2 = (GLubyte \*) malloc(myDataLength);

for(int y = 0; y <480; y++)

{

for(int x = 0; x <320 \* 4; x++)

        {

            buffer2[(479 - y) \* 320 \* 4 + x] = buffer[y \* 4 \* 320 + x];

        }

    }

    // make data provider with data.

    CGDataProviderRef provider =

CGDataProviderCreateWithData(NULL, buffer2, myDataLength, NULL);

    // prep the ingredients

    int bitsPerComponent = 8;

    int bitsPerPixel = 32;

    int bytesPerRow = 4 \* 320;

    CGColorSpaceRef colorSpaceRef = CGColorSpaceCreateDeviceRGB();

    CGBitmapInfo bitmapInfo = kCGBitmapByteOrderDefault;

    CGColorRenderingIntent renderingIntent = kCGRenderingIntentDefault;

    // make the cgimage

    CGImageRef imageRef =

CGImageCreate(320, 480, bitsPerComponent, bitsPerPixel,

bytesPerRow, colorSpaceRef, bitmapInfo,

provider, NULL, NO, renderingIntent);

    // then make the uiimage from that

    UIImage \*myImage = [UIImage imageWithCGImage:imageRef];

    return myImage;

}

# Gallery Design

## Architectural Design

The Jackson Structured Diagram in Figure 3‑1 describes the overall structure of the gallery in IdeaStorm. The gallery will implement the Model-View-Controller design pattern. Within this pattern there will be three major components responsible for the actions of the gallery. These components are the Gallery View (View), the Database (Model) and the View Controller (Controller). These components are described in detail in the sections 3.2, 3.2.3, and 3.2.2.

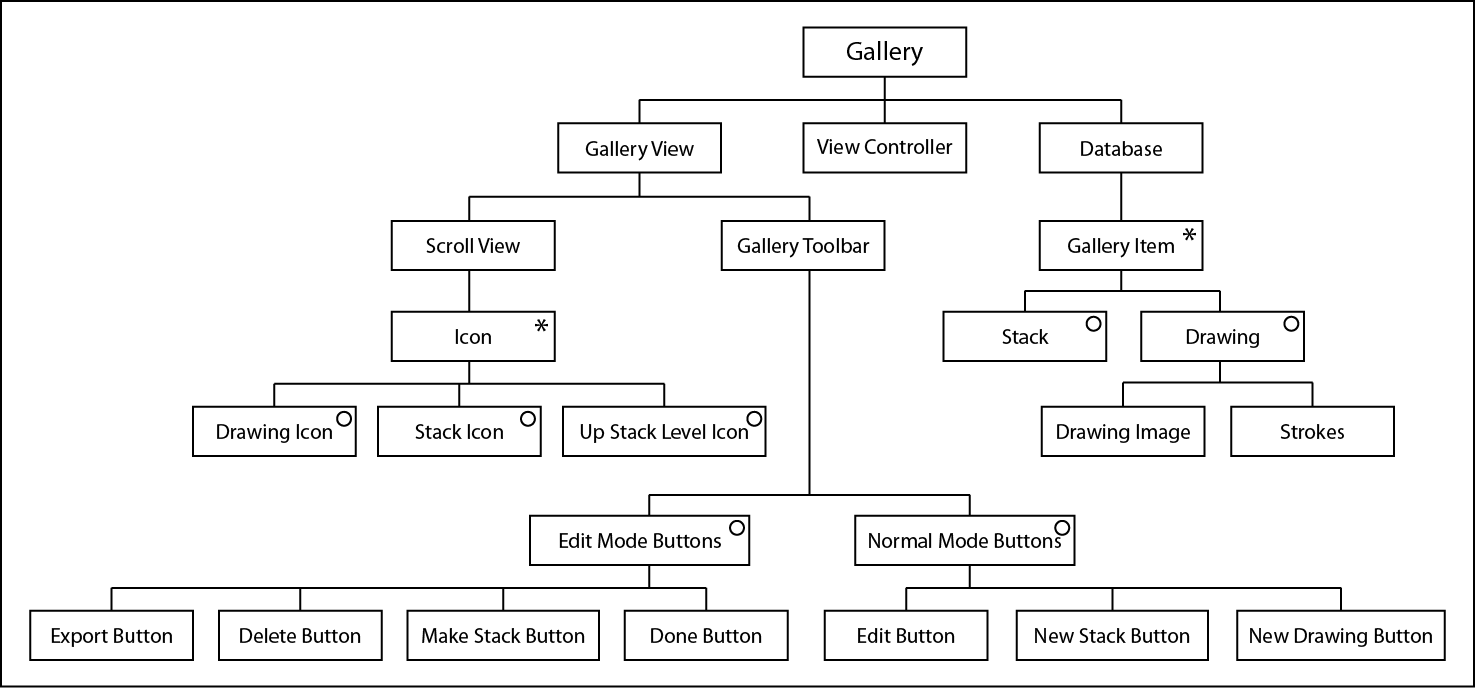


Figure 3‑1 – JSD describing the structure of the gallery.

## Component Design

### Gallery View Components

The gallery view component of the gallery is responsible for all actions defined for the view in the Model-View-Controller design pattern. The gallery view will provide a means for the app user to interact with the gallery and will provide visual feedback for all actions performed. The gallery view’s main class will be the GalleryView class, which is defined in section 3.2.1.1.

The gallery view has two main sub-components, a UIScrollView and the GalleryToolbar. The Scroll View is further described in section 3.2.1.1. The Gallery Toolbar’s main class is described in section 3.2.1.3.

#### GalleryView

The GalleryView class will be responsible for initializing and displaying the UIScrollView and the GalleryToolbar. This class will also conform to the GalleryToolbarDelegate protocol and set itself as the galleryToolbarDelegate for the GalleryToolbar. The UIScrollView will be responsible for displaying the thumbnails for the current stack that the user is viewing, which is represented by the displayedStack property.

The GalleryView will utilize UIButton objects to display the thumbnails of each galleryItem in the displayedStack. Actions will be added to these UIButton objects that open the corresponding objects. In the case where the display stack is not the rootStack, a UIButton will be added that allows the app user to navigate up one stack level.

To position the UIButtons within the UIScrollView, the PositioningHelper class will be utilized.

Properties:

* rootStack – This is the first Stack in a Stack hierarchy.
* displayedStack – This is the Stack that is currently being displayed within the scrollView property.
* scrollView – This is the UIScrollView that will be responsible for displaying the icons that represent the GalleryItems and the icon for navigating up a stack level (when not in the root stack).
* toolbar – This is an instance of the GalleryToolbar that is described in section 3.2.1.3.

Methods:

* - (id)initWithFrame:(CGRect)frame andRootStack:(Stack \*)rootStack – This will initialize the GalleryView to the size specified and will set the rootStack property to the value provided. This will also populate the contents of the scrollView using the rootStack provided by using the openGalleryItem: method.
* - (void)openGalleryItem:(GalleryItem \*)galleryItem – This will open the provided GalleryItem. For a Stack it will display the contents of the Stack within the scrollView. For a Drawing, it will open the Drawing the in the DrawingEngine.

#### GalleryViewDelegate

The GalleryViewDelegate protocol receives instructions from the GalleryView to perform on the Database and DrawingEngine.

Methods:

* - (bool)deleteGalleryItem:(NSObject <GalleryItem>)galleryItem – Deletes the specified GalleryItem including all children.
* - (void)newStackForStack:(Stack \*)stack – Creates a new stack in the stack specified and opens the DrawingEngine so drawings maybe added to it.
* - (void)newDrawingForStack:(Stack \*)stack – Opens the DrawingEngine so new drawings maybe added to the stack specified.
* - (Stack \*)createStackFromDrawing:(Drawing \*)drawing – This creates a new Stack and adds the Drawing provided to the Stack. To accomplish this, the following steps will be done:
  + Create new Stack
  + Get reference to Drawing’s parent
  + Add the Drawing as a child of the new Stack
  + Add the new Stack as a child of the Drawing’s old parent
  + Save the Drawing and the Stack
  + Return the Stack created

#### GalleryToolbar

The GalleryToolbar is a subclass of the UIToolbar class. It is responsible for initializing and displaying all buttons that are used in the gallery. The gallery toolbar will also be responsible for switching between the different button sets depending on the mode that it is currently it. These button sets are described in Figure 3‑1.

Properties:

* galleryToolbarDelegate – This is an NSObject that implements the GalleryToolbarDelegate protocol that will receive commands from the GalleryToolbar when certain buttons are pressed.
* mode – This is read only property that provides the current mode of the GalleryToolbar. This mode an enumeration that will limited to Edit mode and Normal mode.
* editModeButtons – This is an array of UIBarButtonItem object that will be displayed when in Edit mode.
* normalModeButtons – This is an array of UIBarButtonItem objects that will be displayed when in Normal mode.

Methods:

* - (id)initWithFrame:(CGRect)frame – This method will be overridden to initialize the toolbar in Normal mode.
* - (bool)switchToMode:(GalleryToolbarMode)mode – This switch the toolbar to the mode specified which will change the buttons to correspond to the mode.

#### GalleryToolbarDelegate

The GalleryToolbarDelegate is a protocol for receiving commands from an instance of the GalleryToolbar.

Methods:

* exportSelected – This command will be sent when the Export button is pressed.
* deleteSelected – This command will be sent when the Delete button is pressed.
* makeStackFromSelected – This command will be sent when the Make Stack button is pressed.
* createNewStack – This command will be sent when the New Stack button is pressed.
* createNewDrawing – This command will be sent when the New Drawing button is pressed.
* ShowNormalTutorial – This command will be sent when the Help button is pressed in Normal Mode.
* ShowEditTutorial – This command will be sent when the Help button is pressed in Edit Mode.
* modeChange – This command will be sent when the GalleryToolbar changes modes.

#### PositioningHelper

The PositioningHelper class is utilized to calculate and set the x and y coordinates for an array of UIView objects.

Properties:

* viewSpace – This is a CGSize struct that represents the height and width used for spacing the UIView objects provided for positioning. The height represents the value to use as space between centers along the Y-axis while the width represents the value to use as space between the centers along the X-axis.

Methods:

* gridPositionViews:(NSArray \*)arrayOfViews usingWidth:(float)width - This method sets the x and y corrdinates of the provided UIView objects in a grid fashion using the viewSpace height and width and the width provided. The algorithm used for calculating these coordinates is provided in the algorithm section below.

Algorithm:

The algorithm below describes the positioning that takes place in the gridPositionViews:usingWidth: method.

//Gets the number of views that can fit in a single row

viewsPerRow = floor(width / viewSpace.width)

//gets the amount of horizontal space that will not be occupied by views

remainder = width – (viewsPerRow \* viewSpace.width)

//gets the starting point for the first view, will horizontally align the row of views

startPoint = CGPointMake(

( (viewSpace.width / 2) + (remainder / 2) ),

(viewSpace.height / 2) );

point = startPoint

//tracks the number of views positioned in the current row

tally = 0

for (number of views) {

tally++

view.center = point

//if not the last view in the row, shift right, otherwise reset x and shift down

if (tally < viewsPerRow) {

point.x += viewSpace.width

} else {

point.x = startPoint.x

point.y += viewSpace.height

//reset the tally for the row

tally = 0

}

}

### View Controller Components

The view controller component of the gallery is responsible for all actions defined for the controller in the Model-View-Controller design pattern. The view controller will be responsible for passing messages between the gallery view component and the database component. The view controller’s main class for the gallery is the GalleryViewController, which is described in section 3.2.2.1.

#### GalleryViewController

The GalleryView is a subclass of the UIViewController class. The GalleryViewController is responsible for initializing the GalleryView and passing messages between the GalleryView and the Database. The GalleryViewController will also adapt the GalleryViewDelegate protocol.

Properties:

* drawingViewController – This is the DrawingViewController will be displayed when drawings are opened or created.
* galleryView – This is a reference to the GalleryView that will be used in IdeaStorm.
* database – This is a reference to the database that will be used for IdeaStorm.

Methods:

* - (id)initWithNibName:(NSString \*)nibNameOrNil bundle:(NSBundle \*)nibBundleOrNil andDatabase:(Database \*)database – This method would set the database property to the Database specified. This method will also initialize the GalleryView using the root Stack retrieved from the Database. The GalleryView will then be added as a subview of GalleryViewController.

### Database Components

The database component of the gallery is responsible for all actions defined for the model in the Model-View-Controller design pattern. The database will provide a means to save and retrieve the Stacks and Drawings for the gallery. The database will also be responsible for setting any user defaults. The main class for the database will be the Database class described in section 3.2.3.1.

The database components will also implement the Composite design pattern. The GalleryItem protocol will provide a common interface for the Stack and Drawing classes. Implementing this pattern will allow any Stack to contain other Stacks or Drawings as children. All Stacks and Drawings will reside in a Root Stack that is created the first time the app is opened. This structure of the files in the system is illustrated in the figure below.

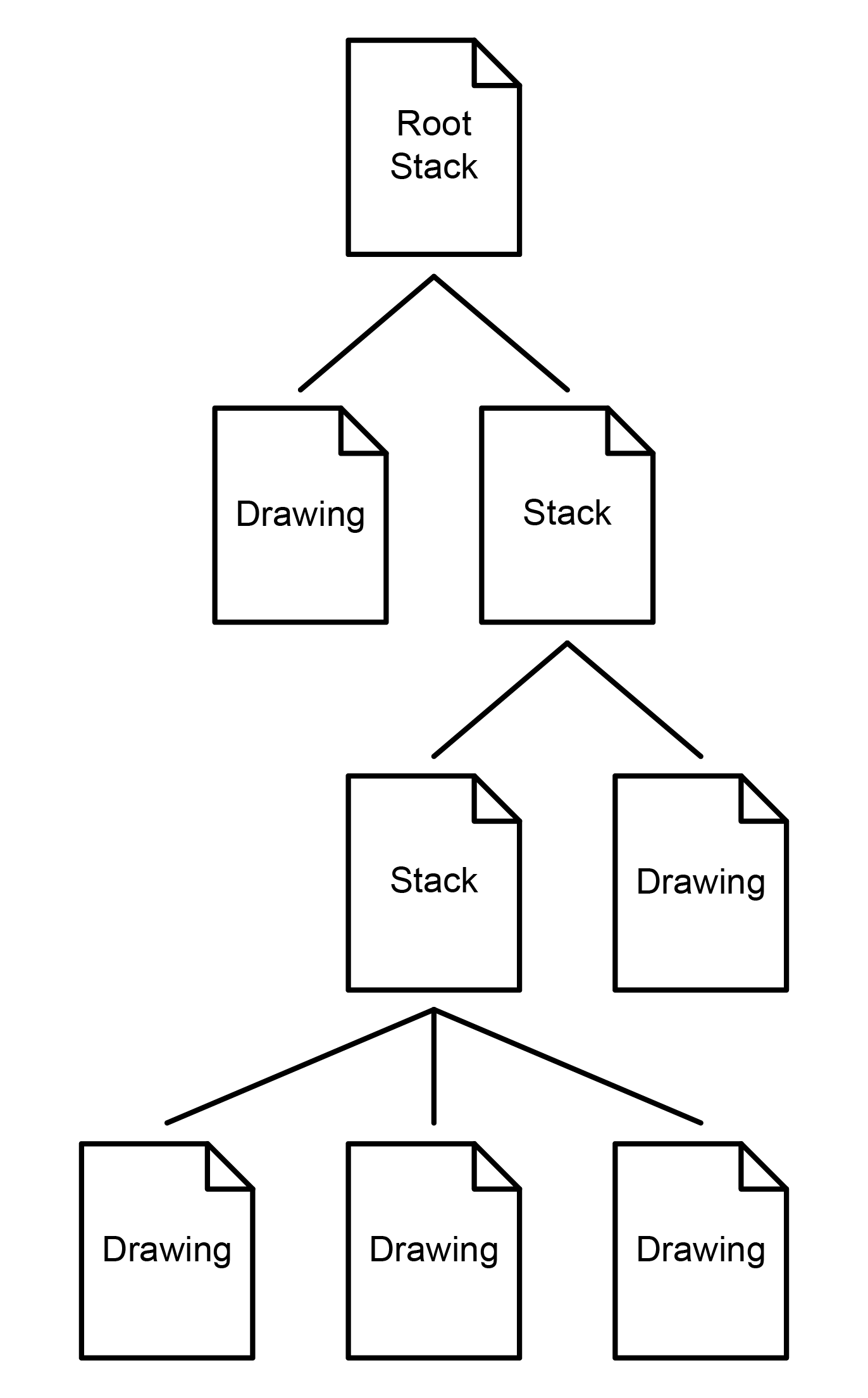


Figure 3‑2 – Example of the file structure for the data in the system.

#### Database

The Database class is responsible for saving, retrieving and modifying GalleryItems. The Database will also provide means of saving and retrieving user defaults and retrieving images from the disk that are used in other parts of the application.

Methods:

* + (UIImage \*)getImageForFilename:(NSString \*)filename – Returns the image with the filename specified. Returns null if the file does not exist.
* - (bool)saveGalleryItem:(NSObject <Galleryitem> \*)galleryItem – Save the provided GalleryItem using the encodeWithCoder: method. The encoded data is saved to a folder that corresponds
* - (NSObject <Galleryitem> \*)getRootGalleryItem – Returns the root Stack that contains all other Stacks and GalleryItems. If there is not root stack currently defined, the gallery will create one.
* - (bool)moveGalleryItem:(NSObject <Galleryitem> \*) child intoGalleryItem:(GalleryItem \*)parent – Reassigns the parent of the current child GalleryItem, copies the child GalleryItem to the new location and deletes the child item from the old location.
* - (bool)deleteGalleryItem:(NSObject <Galleryitem> \*) galleryItem – Deletes the specified GalleryItem. This will also delete of its children recursively.
* + (NSString \*)generateUniqueID – Generates a unique ID.

Algorithms:

To generate the unique ID in the generateUniqueID method, the code below will be utilized. This code was obtained from [McCaughey].

// create a new UUID which you own

CFUUIDRef uuid = CFUUIDCreate(kCFAllocatorDefault);

// create a new CFStringRef (toll-free bridged to NSString)

// that you own

NSString \*uuidString = (NSString \*)CFUUIDCreateString(kCFAllocatorDefault, uuid);

// release the UUID

CFRelease(uuid);

return uuidString;

#### GalleryItem

The GalleryItem is a protocol that that will be adhered to by all items displayed in the GalleryView. This protocol will also incorporate the NSCoding protocol to provide methods for saving and creating a GalleryItem using an NSCoder object.

Properties:

* parent – This is the GalleryItem that contains this GalleryItem. In this implementation this should always be a Stack, however it is typed as a GalleryItem in case other gallery items are added in the future that are able to contain GalleryItems. This will be a nil value for the root stack.
* children – This is a read only property that returns an array of all GalleryItems that are children of this GalleryItem.
* pathID - This is a read only property that contains a unique string ID for the GalleryItem. This will be generated upon initialization using the generateUniqueID: method of the database. This is also used as part of the folder name that the data for the gallery item will be saved in.
* thumbnailImage – This is the thumbnail that represents the GalleryItem. The getter for this property will be overridden to load the property from the disk when the property is currently nil.
* fullImage – This is the full size image for the GalleryItem. The getter for this property will be overridden to load the property from the disk when the property is currently nil.

Methods:

* - (void)encodeWithCoder:(NSCoder \*)encoder – This method will be used to save the properties and state of a GalleryItem using the coder provided.
* - (id)initWithCoder:(NSCoder \*)decoder – This will be used to initialize the GalleryItem using the data stored in the coder provided.
* - (NSString \*)saveThumbnailImage – This is used to save a copy of the thumbnail image for the GalleryItem to storage. This will return the path to the thumbnail’s location.
* - (NSString \*)saveFullImage – This is used to save a copy of the full image for the GalleryItem to storage. This will return the path to the full image’s location.
* - (bool)addChild:(GalleryItem \*)galleryItem – This is used to add a the specified GalleryItem to the children property. This will return true if successful.
* - (bool)deleteChild:(GalleryItem \*)galleryItem – This is used to remove the specified GalleryItem from the children property. This will return true if successful.
* - (NSString \*)getFullPathWithDataFilename:(bool)yesOrNo – This function returns the pathID property prepended with the return value that the parent’s getFullPath function returns. This is a recursive function that will continue these calls until the parent is found to be nil (the root stack). The Boolean value passed to this method determines if the return has the data filename for the object appended.

#### Stack

The Stack class’s main responsibility is to contain other Stacks and Drawings.

The Stack class will implement the GalleryItem protocol. Below outlines the property and methods that will be added or will deviate from the descriptions provided in section 3.2.3.2.

Properties:

* thumbnailImage – This is the thumbnail that represents the Stack. This will be same thumbnail that is used for the first child that the Stack contains, modified to indicate that this is a thumbnail for a stack. An example for this would be placing the child’s thumbnail within a frame to create the Stack’s thumbnail.
* fullImage – This will return a nil value for a Stack.

Methods:

* - (void)encodeWithCoder:(NSCoder \*)encoder – When this method is called, the full path to the data file for each child will be encoded into the encoder.
* - (id)initWithCoder:(NSCoder \*)decoder – Calling this method will initialize the Stack and will in turn initialize the children of the this stack using the file paths encoded with the encodeWithCoder: method.
* - (bool)saveFullImage – This will be an empty implementation that will return a nil value.
* - (bool)saveThumbnailImage – This will be an empty implementation that will return a nil value.

#### Drawing

The Drawing class contains all the data that is related to a drawing created in the DrawingEngine.

The Drawing class will implement the GalleryItem protocol. Below outlines the property and methods that will be added or will deviate from the descriptions provided in section 3.2.3.2.

Properties:

* fullImage – The setter for this property will be overridden to create and set the thumbnailImage property using the image provided. The image will be copied and resized using UIImage categories created by [Harmon]. The method utilized from these categories for resizing is the resizedImageWithContentMode:bounds:interpolationQuality: method. These categories will be modified slightly as needed to work within the system and include the following categories:
  + UIImage+Resize
  + UIImage+RoundedCorner
  + UIImage+Alpha

Methods:

* - (void)encoceWithCoder:(NSCoder \*)encoder – This method will be used to save the properties and state of a GalleryItem using the encoder provided.
* - (id)initWithCoder:(NSCoder \*)decoder – This will be used to initialize the GalleryItem using the data stored in the decoder provided.
* - (bool)addChild:(GalleryItem \*)galleryItem – This will be an empty implementation that will return false.
* - (bool)deleteChild:(GalleryItem \*)galleryItem – This will be an empty implementation that will return false.

Imported Classes:

The following

# TIME AND RESOURCE ESTIMATION

The coding for changes needed should take approximately 29 hours to complete. This estimate is based off the timesheet data for previous iterations that shows coding can take up to 2.43 hours on average per class that is created or modified. Taking this into consideration there are 6 classes that will be created and 4 classes that will be modified in order to implement the gallery for IdeaStorm. Data illustrating this is displayed in the table below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Iteration | Classes Created | Classes Modified | Total Classes | Total Coding Time | Avg Coding Time Per Class |
| 001 | 8 | 1 | 9 | 21.88 hours | 2.43 hours |
| 002 | 9 | 6 | 15 | 33.17 hours | 2.21 hours |
| 003 | 9 | 7 | 16 | 38.88 hours (est) | 2.43 hours (est) |

Testing is estimated to take approximately the same amount of time to complete. Historical data for this is being referenced for the estimate. However this estimate is not based completely on historical data as there are more possible interactions for this iteration as compared to past iterations. This is illustrated by the increase in overall user interface requirements in the SRS. Due to this the estimate for testing has been raised accordingly.

# RISK ANALYSIS

## Technical Risks

The developer is still relatively inexperienced with OpenGL programming. This is being mitigated by his continued research in OpenGL programming techniques.

The drawing engine is increasing in complexity; there is a chance that an issue will be encountered that was not foreseen. Completing this design process to outline a basis for the design and interactions between classes mitigates this risk.

The developer will be working for the first time with the NSCoding protocol and with the NSCoder class. This is being mitigated by research, however because of the relative complexity and possibility that this might fail, other options are being considered for storage of data. The first options that will be considered if using NSCoding and NSCoder fails would be saving the data via the User Defaults as the developer already has experience using this method.

The developer will be working for the first time using UIScrollView and UIToolbar as well. However, these risks are less likely to affect development as the complexity of using these classes are relatively low. Research into the use of these classes is being done to help mitigate this risk.

There is also a high risk that there will be complication implementing the Composite pattern for the gallery items using NSCoding and NSCoder for storage. This is being mitigated by:

* Researching and referencing material that documents the Composite pattern
* The alternative for storage discussed above.

If complications arise that make it unpractical to implement the composite pattern, the storage design of Stacks and Drawings will be changed. This includes two possibilities listed below:

* Storing all Drawings into a single stack.
* Storing only Stacks in the root Stack and storing only Drawings within these Stacks.

## Schedule Risk

The time estimates may be inaccurate as it is based on past historical data. This data was collected on performing tasks that are vastly different then the tasks being performed during this iteration.

There is also a risk that the developer may need to dedicate more time to his other class which would take away available time to work on this project.

If any of these risks occur there is a high risk that the deadline for app store submission could be missed. To mitigate this, the deadlines for each phase of development are being treated as fixed. Instead of adjusting the time schedule for any phase, requirements will be trimmed to meet the deadline.

# LIST OF CODE AND DOCUMENTS TO BE ADDED/MODIFIED

## Classes

The following classes will be added, modified or created during the creation of the IdeaStorm drawing engine.

* AppDelagate.h, AppDelagate.m (Modified)
* Database.h, Database.m (Modified)
* Drawing.h, Drawing.m (Created)
* DrawingEngine.h, DrawingEngine.m (Modified)
* DrawingViewController.h, DrawingViewController.m (Modified)
* GalleryItem.h (Created)
* GalleryToolbar.h, GalleryToolbar.m (Created)
* GalleryToolbarDelegate.h (Created)
* GalleryView.h, GalleryView.m (Created)
* GalleryViewDelegate.h (Created)
* GalleryViewController.h, GalleryViewController.m (Created)
* PositioningHelper.h, PositioningHelper.m (Created)
* Stack.h, Stack.m (Created)
* Toolbar.h, Toolbar.m (Modified)
* UIImage+Resize.h, UIImage+Resize.m (Added, Modified)
* UIImage+RoundedCorner.h, UIImage+roundedCorner.m (Added, Modified)
* UIImage+Alpha.h, UIImage+Alpha.m (Added, Modified)

## Documentation

Other then documentation that will be added to the code as comments, no other documentation will be created as a tutorial for use of the app will be included in the app and installation will eventually be taken care of through the app store.

# References

Bullock, B. (2009, September 30). Approximate length of a cubic bezier curve. *LeMoDa.net*. Retrieved September 13, 2011, from http://www.lemoda.net/maths/bezier-length/index.html

Chung, C., & Bucanek, J. (2011). *Pro Objective-C design patterns for iOS*. New York: Friends of ED/Apress.

Drawing a curve in OpenGLES - how? - GameDev.net. (2009, May 7). *GameDev.net*. Retrieved August 26, 2011, from http://www.gamedev.net/topic/534082-drawing-a-curve-in-opengles---how/

Harmon, T. (2009, October 12). Resize a UIImage the right way ‚Trevor’s Bike Shed. *Trevor's Bike Shed*. Retrieved Febuary 22, 2012, from http://vocaro.com/trevor/blog/2009/10/12/resize-a-uiimage-the-right-way/

iPhone - GLPaint Save Image - Stack Overflow. (2009, December 22). *Stack Overflow*. Retrieved April 22, 2012, from http://stackoverflow.com/questions/946700/glpaint-save-image

Lamarche, J. (2010, November 2). iPhone Development: OpenGL ES 2.0 for iOS, Chapter 4 - Introducing the Programmable Pipeline. *iPhone Development*. Retrieved August 12, 2011, from http://iphonedevelopment.blogspot.com/2010/11/opengl-es-20-for-ios-chapter-4.html

Lazuka, K. (2009, November 19). Life Kid / Point Sprites and OpenGL ES 2.0. *Life Kid*. Retrieved September 13, 2011, from http://klazuka.tumblr.com/post/249698151/point-sprites-and-opengl-es-2-0

McCaughey, D. (2010, September 30). Able Pear Software: Creating a GUID or UUID in Objective-C. *Able Pear Software*. Retrieved March 11, 2012, from http://blog.ablepear.com/2010/09/creating-guid-or-uuid-in-objective-c.html

Pinto, A. (2009, August 1). The Generic UWF Maintenence Process - SHort Version. *Generic Analysis Report Standard*. Retrieved August 28, 2011, from www.cs.uwf.edu/~seadmin/GumpShort/anastan.htm

Smooth line connecting CGPoints. (2010, May 13). Developer Forums. Retrieved May 5, 2011, from https://devforum.apple.com/message/41732?start=0&start=0

Wenderlich, R. (2010, September 7). How To Save Your App Data With NSCoding and NSFileManager | Ray Wenderlich. *Ray Wenderlich | Tutorials for iPhone / iOS Developers and Gamers*. Retrieved March 7, 2012, from http://www.raywenderlich.com/1914/how-to-save-your-app-data-with-nscoding-and-nsfilemanager