**OpenKiln Research**

General Research

* What is a framework? A framework is a means of programming that implements inversion of control. This means that a number of different programs can all run through the same framework but implement different code/functions and thereby achieve different results. When building a framework, the framework should not know the details of the underlying application, but should instead respond to actions by the user or network and call the appropriate subroutines.
* How does this apply to the drawing application? The first step is to envision a variety of different implementations of this framework. Some ideas are:
  + A modelling application that operates from a fixed viewpoint. From here, users can drag and drop their models to create a 3D environment. These models should additionally move and respond to other variables. This application would need:
    - Ability to switch between editor mode (create models: user can draw or import models from online) and environment mode (user can drag-and-drop models into the environment)
    - Toolbar for each (editor: Draw, Add Shape, Copy, Delete, Import, Save) & (environment: Drag & Drop, Copy, Delete, Save)
    - In each mode, and with each option selected on the toolbar, the application should respond to user clicks in various ways
  + An architecture application that operates from a variable viewpoint. Users can rotate the viewing plane as well as place objects, shapes and lines into free space. The elements of this application should be fixed. This application would need:
    - Ability to switch between grid mode (display coordinates around screen for more technical placements) and free mode (view elements cleanly, like you would when exported elsewhere)
    - Toolbar (Rotate Plane, Move Shapes, Add Shape, Copy, Delete, Import, Save)
    - The selected toolbar element would determine whether mouse clicks work to move the screen, objects within the screen, or to perform one of other various functionalities
* Now, what are the overarching themes or related points in the above application that can be extracted to my 3D drawing framework?
  + MODE: Determines which of various modes you are in and passes this information on to other objects or functions so that they can respond appropriately
  + COORDINATE SYSTEM: An origin must be defined, as well as a default camera position relative to that origin. The coordinate system should be able to be swapped between Cartesian/polar/whatever else, and toggles should exist for enabling/disabling gridlines. Additionally, the camera position should be able to be rotated around this coordinate system as well as zoomed in and out, but it should always point directly at the origin. Implementation of these features (like whether they exist or not) can be varied by the internal application.
  + TOOLBAR SELECTION: The various toolbar features should be given icons to place on the screen, and selecting each should modify the program’s behavior. Keyboard shortcuts can also be added to select these options.
    - ROTATE CAMERA: Click-and-drag to shift the camera in a sphere around the origin. Scroll to zoom in or out
    - TOGGLE GRIDLINES: Select between various coordinate planes, and determine whether or not to show the gridlines on the screen.
    - SET BACKGROUND COLOR: The background color can be changed to better display the creation on-screen.
    - PLACE OBJECT: Object can be selected from a selection of defaults (shapes etc.) or imported from user’s computer/phone. Imported objects must be the correct extension type. Selected object can then be placed by user click-and-release. (Object can also be placed by specifying coordinates based on selected system?)
    - MOVE OBJECT: First click-and-release selects and highlights an object. Click-and-dragging this object moves it around the screen.
    - COPY OBJECT: Click-and-release to highlight and copy an object. Switches to ADD OBJECT mode with this object selected for pasting.
    - DELETE OBJECT: Click-and-release to highlight an object. Click-and-release again to remove this object from the screen.
    - DRAW: Allows user to free-draw shapes. Applications can implement their own programs for solidifying user drawings into concrete objects, but the framework should know how to call these. Along with the draw feature should come a number of other features, like pen size, color, density, etc. and an eraser option.
    - CLEAR ALL: Resets the entire canvas to be a blank screen again.
    - SAVE: Saves the creation as one of the available extension types to a location on the user’s computer/phone and stores this creation in the application’s database.
* With this background, let’s begin some research into other frameworks/ implementation methods…
  + Skia 2D Graphics Library (<https://skia.org/>)
    - Skia is a library/API so it may not be the best example for me to follow, but it helps me get an idea of how users might implement my framework. Some general functions/trends I’m seeing in this application are: custom Object types, paths that lines can follow, draw(), a Canvas object, setColor(), setShader(), setEffect(), drawSpecificShape/Object(), rotate(), translate(), etc. Resultingly, my framework should know which commands to call based on the mode and toolbar selection currently made. Not all of these functions need to be recognized by my framework, as my framework can know of a draw() function that in turn makes calls to drawSpecificShape/Object() as an example.
  + Blend2D Graphics Engine (<https://blend2d.com/>)
    - Another API I think, Blend2D reminds me that my framework should know how to draw by following the mouse around the screen as well as by producing vectors. Additionally, it offers a render function, which would be very useful if I want my objects to be constantly moving on the screen. This once again gives insights into the things that my framework will need to know how to support.
  + DirectX (<https://developer.nvidia.com/directx>)
    - DirectX is a 3D graphics API used in virtually all games that can run on Windows. While used for creating video games, it can offer some insights into the 3D aspects of a drawing application that the other APIs were missing. DirectX includes features for Ray Tracing, Mesh Shading, Sampler Feedback, and more. While most of these things go over my head, it is helpful to know about what it would take to create high quality designs that might be needed by programmers. Plenty more information about all of these features can be found at the site above.
  + I cannot find a framework helppp

Use-Case Ideas

Since I don’t know which we’re looking for, I will split this into use-cases for programmers and for end-users.

* Actor: Programmer
* Use-Cases:
  + Create Mode (rendering information, what should be passed to other functions)
  + Customize Toolbar (which abilities should be unlocked for users)
  + Extend Toolbar (create new features not provided by framework?)
  + Set Defaults (default Canvas, coordinate plane, modes, etc.)
* Actor: End-User
* Use-Cases:
  + Select Toolbar
  + Set Draw Mode
  + Set Draw Details
  + Rotate Canvas
  + Select Mode
  + Open Project
  + Close Project
  + Save Project
  + Delete Project

Design Pattern Ideas

* Singleton – probably could be used for the Canvas or Window or Renderer or something
* Façade – definitely could see some uses, especially if breaking the framework down into subsystems like drawing, plane/canvas, toolbar, etc. Façade would bring all of these elements together
* Adapter – Not necessary on its own but programmers may use it to adapt their code to our framework
* Template – All generally-implemented shapes should probably follow a template, custom objects could maybe fit it too
* Factory – Can be used for random generation of a landscape, supply squares of textures and use FM to determine when to place which
* Abstract Factory – Used if creating families of objects, like filled square, bordered square, dashed square, dotted square, etc.
* Builder – Can be used to create more complex objects from simpler ones, like a Pillar using Cylinders and Squares
* Prototype – All objects could implement prototype for easy copying and pasting
* Proxy – Don’t really see a need for Proxy
* Composite – Like Builder, more complex objects can consist of simpler Leaf objects, so Pillar would be a composite element
* Decorator – Like Abstract Factory, ability to append filled or dashed or dotted to a generic square element
* State – Can maintain state of entire application for determining what to render based on the mode and toolbar selection
* Strategy – Would be useful for making a draw() function that varies its algorithm based on the selected Strategy (hold and follow vs. vector)
* Command – Not yet sure how this would be implemented, maybe if various functions (open design, close design, save design, delete design) were treated the same and varied by button selected
* Chain of Responsibility – Don’t see how this one could be used
* Mediator – When objects get convoluted (connection to Copied Objects and Canvas and Physics Engine), Mediator could be an overarching class that handles all the connections between objects
* Observer – Maybe the ability to hard/soft copy objects. Hard copies changes when the any other copies do, soft copies do not. Observer could watch all other copies to see how they change
* Visitor – Could be used when adding new and better functionality, like shaders support, to the existing framework

Questions

* Are the actors in our Use-Case diagrams meant to be the coders interacting with our framework or the users interacting with the built-out programs? Or both?
* I’m trying to look up similar frameworks online but I’m more often finding libraries and APIs. Are these comparable or should a framework be more extracted and customizable than the other two?
* I really haven’t gotten anything out of you quickly scrolling through your example Design Doc, is there a more concrete example that we could follow with less information but to still give us a good idea of how it should be structured? I can’t remember if our doc is supposed to contain sample code or just UMLs and ideas or what