

Milestone 2

Singularity Software

March 22, 2012

Spring 1 Backlog








Backlog									
<input type="button" value="Filter"/>		Show Closed Items:							
<input type="button" value="Quick Close"/>		of T T							
<input type="checkbox"/>	Title	ID	Owner	Status	Estimate	Detail Estimate	Done	Effort	To Do
<input type="checkbox"/>	Emulation: Reload a program	S- 01001	Kurtis	Future	3.00				H Reopen Story
<input type="checkbox"/>	Emulation: Connect UI events to Cube EventHandlers	S- 01002	Ethan	In Progress	10.00	10.00	8.50	1.00	H Quick Close
<input type="checkbox"/>	Emulation: Loading a program	S- 01003	Kurtis	Accepted	5.00	5.00	5.00	0.00	H Reopen Story
<input type="checkbox"/>	UI: Cube drag-and-drop with displacement	S- 01005	Alex		8.00	8.00	5.00	7.00	H Quick Close
<input type="checkbox"/>	UI: Make rotate buttons not rotate with cube	S- 01006	Alex	Accepted	1.00	1.00	1.00	0.00	H Reopen Story
<input type="checkbox"/>	Emulation: Ability to add images to Cubes in programs	S- 01007	Kurtis	Accepted	8.00	8.00	7.50	0.00	H Reopen Story
<input type="checkbox"/>	Emulation: Ability to use Siftee's Data class	S- 01008	Richard	Accepted	8.00				H Reopen Story
<input type="checkbox"/>	Learn MWM	S- 01009	Richard	Accepted	4.00	4.00	4.00	0.00	H Reopen Story
<input type="checkbox"/>	Sprint Write Up	S- 01010	Alex, Kurtis, Ethan, Richard	Accepted	6.00				H Reopen Story

The CubeEventHandlers are being tested for accuracy in the next sprint, and that task will be approved when they are completely tested. The drag-and-drop with replacement feature is in progress but unlikely to be finished for this sprint, so it will carry over to the next sprint.

Sprint 2 Backlog

Backlog Details

Add Story Inline

Filter							
Move to Sprint		O		T		T	
<input type="checkbox"/> Title	ID	Owner	Status	Estimate	Detail Estimate	To Do	
<input type="checkbox"/>  Emulation: Implement public Cube methods (see Sifteo API)	S-01011	Ethan		1.00		Plan Story	
<input type="checkbox"/>  Emulation: Implement public Color methods (see Sifteo API)	S-01012	Ethan		4.00		Plan Story	
<input type="checkbox"/>  Emulation: Implement public CubeSet methods (see Sifteo API)	S-01013	Ethan		4.00		Plan Story	
<input type="checkbox"/>  Emulation: Implement Sprite class	S-01017	Kurtis		20.00		Plan Story	
<input type="checkbox"/>  Emulation: Implement StateMachine class	S-01018	Alex		15.00		Plan Story	
<input type="checkbox"/>  Documentation: Milestone 3	S-01019	Alex, Kurtis, Ethan, Richard		6.00		Plan Story	
<input type="checkbox"/>  Application: Test Cube Actions	S-01020	Richard		8.00		Plan Story	
Move to Sprint		1- of		T		T	

This sprint's work will include finishing out a few classes in the Sifteo namespace and implementing two classes from the Sifteo.Util namespace (Sprite and StateMachine). Additionally, we will be writing an application that will be used to test the cube manipulation event handlers written in the last sprint, and those handlers will be tweaked as necessary.

Class Diagram Before



After

Siftables
Class
↳ Application

Methods

- Application_Exit() : void
- ApplicationStartup() : void
- ApplicationUnhandledException() : void
- ReportErrorToDOM() : void
- Siftables()

AppRunner
Class

Fields

- _appRunner : AppRunner
- _runner : Thread
- _uiDispatcher : Dispatcher
- TimeBetweenTicks : int

Properties

- App : BaseApp
- Cubes : CubeSet
- IsRunning : bool

Methods

- AppRunner()
- FindBaseApp() : BaseApp
- GetInstance() : AppRunner
- LoadApplication() : void
- PauseExecution() : void
- ResetCubes() : void
- RunAppInThread() : void
- StartExecution() : void
- StopExecution() : void

MainWindowView
Class
↳ UserControl

Methods

- MainWindowView()

MainWindowViewModel
Class

Fields

- _imageSources : ImageSources
- _status : string
- NumInitialCubes : int
- ReadyStatus : string

Properties

- AppRunner : AppRunner
- ChangeNumberOfCubesComm and : RelayCommand<EventArgs>
- CubeSet : CubeSet
- CubeViewModels : ObservableCollection<CubeViewModel>
- LoadAFileComm and : RelayCommand
- RefreshNeighborsComm and : RelayCommand
- ReloadAFileComm and : RelayCommand
- SnapToGridComm and : RelayCommand
- Status : string

Methods

- CalculateNeighbors() : void
- MainWindowViewModel()
- NotifyPropertyChanged() : void

Events

- PropertyChanged : PropertyChangedEventHandler

ClipBehavior
Class

Fields

- ToBoundsProperty : DependencyProperty

Methods

- ClipToBounds() : void
- fe_Loaded() : void
- fe_SizeChanged() : void
- GetToBounds() : bool
- OnToBoundsPropertyChanged() : void
- SetToBounds() : void

CubeView
Class
↳ UserControl

Methods

- CubeView()

CubeViewModel
Class

Fields

- _backgroundColor : Brush
- _imageSources : ImageSources
- _pendingBackgroundColor : SolidColorBrush
- _pendingScreenItems : Collection<FrameworkElement>
- _positionX : int
- _positionY : int
- _screenItems : ObservableCollection<FrameworkElement>

Properties

- BackgroundColor : Brush
- ButtonPressComm and : RelayCommand
- ButtonReleaseComm and : RelayCommand
- CubeFlipComm and : RelayCommand
- CubeModel : Cube
- ImageSources : ImageSources
- PositionX : int
- PositionY : int
- RotateCcwComm and : RelayCommand
- RotateCwComm and : RelayCommand
- ScreenItems : ObservableCollection<FrameworkElement>
- TiltDownComm and : RelayCommand
- TiltLeftComm and : RelayCommand
- TiltRightComm and : RelayCommand
- TiltUpComm and : RelayCommand

Methods

- AddPendingImage() : void
- AddPendingRectangle() : void
- CubeViewModel()
- EmptyScreenItems() : void
- NotifyPropertyChanged() : void
- PaintPendingGraphics() : void
- UpdatePendingBackgroundColor() : void

Events

- PropertyChanged : PropertyChangedEventHandler

DragAndDropBehavior
Class
↳ Behavior<UIElement>

Fields

- IsDragging : bool
- mouseClickPosition : Point
- parent : DependencyObject

Methods

- AssociatedObject_MouseLeftButtonDown() : void
- AssociatedObject_MouseLeftButtonUp() : void
- AssociatedObject_MouseMove() : void
- OnAttached() : void
- OnDetaching() : void

CubeSet
Class
↳ Collection<Cube>

Methods

- ClearEvents() : void
- ClearUserData() : void
- CubeById() : Cube
- CubeSet()
- ToArray() : Cube[]

ImageSources
Class

Fields

- ValidImageExtensions : string[]

Properties

- ImageSource : Dictionary<string, BitmapImage>

Methods

- Contains() : bool
- GetBitmapImage() : BitmapImage
- GetImageInfo() : ImageInfo
- GetImageSet() : ImageSet
- ImageSources()
- LoadImages() : void

Cube
Class

Fields

- dimension : int
- SCREEN_HEIGHT : int
- SCREEN_MAX_X : int
- SCREEN_MAX_Y : int
- SCREEN_MIN_X : int
- SCREEN_MIN_Y : int
- SCREEN_WIDTH : int
- userData : object

Properties

- ButtonsPressed : bool
- IsShaking : bool
- Neighbors : Neighbors
- Orientation : Side
- Uniqueld : string

Methods

- FillRect() : void
- FillScreen() : void
- Image() : void
- OnButtonPress() : void
- OnButtonRelease() : void
- OnFlip() : void
- OnMove() : void
- OnRotateCCW() : void
- OnRotateCW() : void
- OnTilt() : void
- Paint() : void

Events

- NotifyBackgroundColorChanged : EventHandler
- NotifyButtonPressed : ButtonEventHandler
- NotifyButtonReleased : ButtonEventHandler
- NotifyCubeFlip : FlipEventHandler
- NotifyCubeMoved : EventHandler
- NotifyCubeTilt : TiltEventHandler
- NotifyNewImage : EventHandler
- NotifyNewRectangle : EventHandler
- NotifyPaint : EventHandler
- NotifyRotateCCW : RotateEventHandler
- NotifyRotateCW : RotateEventHandler
- NotifyScreenItemsEmptied : EventHandler

Nested Types

Neighbors
Class

Fields

- _neighbors : Cube[]
- _numNeighbors : int
- GAP_TOLERANCE : int
- SHARED_EDGE_MINIMUM : i...

Properties

- BOTTOM : Cube
- Count : int
- IsEmpty : bool
- LEFT : Cube
- RIGHT : Cube
- TOP : Cube

Methods

- Contains() : bool
- CubeOnSide() : Cube
- Neighbors() :
- SideOf() : Side
- sideUtil() : void

BaseApp
Class

Properties

- AppID : string
- CubeSet : CubeSet
- DeltaTime : float
- FrameRate : int
- Images : ImageSet
- IsIdle : bool
- ResourcePath : string
- StoredData : Data
- TargetDeltaTime : float
- Time : float

Methods

- AssociateCubes() : void
- BaseApp() :
- Setup() : void
- Tick() : void

RectangleEventArgs
Class
↳ EventArgs

Properties

- Color : Color
- Height : int
- Width : int
- X : int
- Y : int

Methods

- RectangleEventArgs() :

Color
Struct

Fields

- _b : byte
- _g : byte
- _r : byte

Properties

- B : byte
- Black : Color
- G : byte
- R : byte
- White : Color

Methods

- Color() :

ImageSet
Class

Fields

- _imageInfos : Collection<ImageInfo>

Properties

- this : ImageInfo

Methods

- Contains() : bool
- ImageSet() :
- InfoFor() : ImageInfo

BackgroundEventArgs
Class
↳ EventArgs

Properties

- BackgroundColor : Color

Methods

- BackgroundEventArgs() :

Data
Class

Fields

- _data : string

Properties

- IsValid : bool

Methods

- Load() : string
- Store() : void

ImageInfo
Struct

Properties

- height : int
- name : string
- width : int

ImageEventArgs
Class
↳ EventArgs

Properties

- Height : int
- ImageName : string
- Rotation : int
- Scale : int
- SourceX : int
- SourceY : int
- Width : int
- X : int
- Y : int

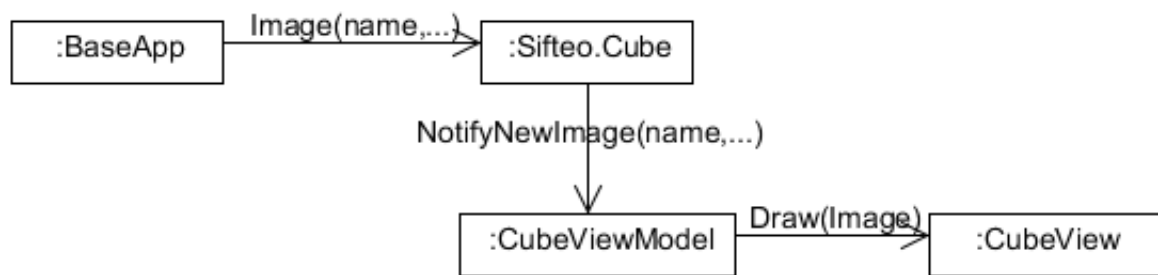
Methods

- ImageEventArgs() :

Struggle with Code Inertia

Implementing the `Cube.Image()` method presented a challenge due to the original structure of our viewmodels.

For an image to be rendered to a cube's screen, and in order to mimic the structure we had in place for background color and rectangles, we needed to carry out the following sequence of events:



`CubeViewModel` is an observer of its `Cube` (Sifteo model domain), which notifies all subscribers when `FillScreen`, `FillRect`, and `Image` are called from an application. The `CubeView` (Silverlight UI) is then bound to the collection of graphical elements which the `CubeViewModel` maintains, so what may be interpreted as method calls above are really notifications (except for the case of `Image(name,...)`).

This worked well for background colors or rectangles, but images introduce the notion of assets which are associated with an application.

When an application is loaded, each of the images in that application's assets directory is loaded in to an `ImageSources` collection, and any image to be rendered on a cube must come from that collection. We have a class `ImageSources` which is responsible for loading and making available the image assets, so each `CubeViewModel` must have a reference to these image sources. Because applications are loaded in `MainWindowViewModel` (through the `AppRunner`), the image assets are also loaded at that time (when the application directory is still available). At this point, each of the `CubeViewModels` needs to have the ability to access those image sources to notify the `CubeViews` which image to render.

This is where our previous structure held us back until some refactoring was done.

First of all, the `MainWindowView` was bound to the `MainWindowViewModel`, as it should be, and each of the `CubeViews` was bound to a `CubeViewModel`, but within the `MainWindowViewModel` instead of initializing a collection of `CubeViewModels`, we had initialized a collection of `CubeViews`. Not only does this violate the principles of MVVM, it does not serve our imaging code either. To access any `CubeViewModel`, we had to go fetch it from the view, using a call along the lines of

```
(CubeViewModel) cubeView.LayoutRoot.DataContext
```

which reeks of “Middle Man,” “Message Chain,” and “Inappropriate Intimacy,” depending on the vantage point. Instead, we decoupled the CubeViews from the MainWindowViewModel (using some form of “Remove Middle Man” and separating the layers of our architecture) so that we never explicitly create a CubeView and instead initialize CubeViewModels and bind the ItemsControl to those viewmodels and use a DataTemplate to indicate to the MainWindowView how we want each viewmodel represented visually.

At this point, we had to refactor those DataContext calls to reference the CubeViewModels we now have direct access to, and after this restructuring giving each CubeViewModel access to the set of image sources is much more natural to implement because it can simply be passed from the MainWindowViewModel, where it is created, to the CubeViewModels directly.

Although this refactoring added time to this particular task, it will clear the path for the features we are adding in future sprints, because it is expected that we will be implementing sprites and sounds in a similar manner. Now the MVVM structure is correctly preserved, and the classes we have are much more cohesive.