EXHIBIT M

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Collections;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Globalization;
using System.Security;

using UMD.HCIL.Piccolo;
using UMD.HCIL.Piccolo.Nodes;
using UMD.HCIL.Piccolo.Event;
using UMD.HCIL.Piccolo.Activities;
using UMD.HCIL.PiccoloX;
using UMD.HCIL.PiccoloX.Events;
using UMD.HCIL.Piccolo.Util;

namespace LaunchPoint
{
	public delegate void NavigationDelegate();

	/// <summary>
	/// Summary description for Form1.
	/// </summary>
	public class ShellForm : PForm
	{
		public static PForm ApplicationForm;
		PCamera landscapeCamera = new LandscapeCamera();
		Landscape landscapeNode;
		//LODNode lodNode;
		Constants.ShellLevel shellLevel;
		LaunchPoint launchPoint;
		//ButtonHarness harness;
		const int HBUTTON_CALENDAR = 1;
		const int HBUTTON_CONTACTS = 2;
		const int HBUTTON_MAIL = 3;
		const int HBUTTON_ITASK = 4;
		const int HBUTTON_SIDE = 5;
		const int HUP = 6;
const int HDOWN = 7;
const int HLEFT = 8;
const int HRIGHT = 9;
const int HCENTER = 10;
private System.Windows.Forms.PageSetupDialog pageSetupDialog1;
static bool PLATFORM_POCKETPC = false;

public ShellForm()
{
        /*
        if (PLATFORM_POCKETPC)
        {
                harness = new ButtonHarness();
                this.Load += new System.EventHandler(this.ShellForm_Load);
                this.Closing += new System.ComponentModel.CancelEventHandler(this.ShellForm_Closing);
        }
        */
        //
        // Required for Windows Form Designer support

        InitializeComponent();

        ReadTransparency();

        Rectangle screenBounds = Screen.PrimaryScreen.Bounds;
        if (screenBounds.Width == 800 && screenBounds.Height == 600)
        {
                System.Windows.Forms.Cursor.Hide();
                FormBorderStyle = FormBorderStyle.None;
        }
        else
        {
                FormBorderStyle = FormBorderStyle.FixedSingle;
        }

        //
        // TODO: Add any constructor code after InitializeComponent call
        //
}
```

```
		public void ReadTransparency() {
			FileStream fs = null;
			try {
				fs = new FileStream("config.txt", FileMode.Open);
			}
			catch (SecurityException e) {
				MessageBox.Show("Unable to load preferences." + "\n\n" + e.Message);
				return;
			}
			catch(FileNotFoundException) {
				return;
			}

			if (fs != null) {
				StreamReader sr = null;
				sr = new StreamReader(fs);
				if (sr != null)
				{
					String line;
					while ((line = sr.ReadLine()) != null)
					{
						String val = sr.ReadLine();
						switch (line)
						{
							case "ALPHA":
								Constants.MAX_TRANSPARENCY = float.Parse(val,
NumberStyles.AllowDecimalPoint);
								break;
							case "NOISE_ALLOWANCE":
								Constants.DRAG_THRESHOLD = int.Parse(val, NumberStyles.None);
								break;
							case "TAP_HOLD_DELAY":
								Constants.TAP_HOLD_DELAY = int.Parse(val, NumberStyles.None);
								break;
							case "MENU_FADEOUT_DELAY":
								Constants.MENU_FADEOUT_DELAY = int.Parse(val, NumberStyles.None);
								break;
						}
					}
					sr.Close();
```

```
                }
                fs.Close();
            }
        }

        /*
        private void ShellForm_Load(object sender, System.EventArgs e)
        {
            //REQUIRED CODE - adding handlers and activating the button harness
            //enable the handler for the four position control by passing it a reference
            //to the form you will be using...
            harness.B_Release += new ButtonReleaseEventHandler(harnessButtonReleased);
            harness.ActivateButtonHarness();
            //END REQUIRED CODE
        }
        */
        /*
        private void ShellForm_Closing(object sender, System.ComponentModel.CancelEventArgs e)
        {
            //REQUIRED CODE - disposing the button harness
            //It is important to call the dispose method - since the button harness *may* not
            //automatically restore access to the keys...
            this.harness.Dispose();
            //END REQUIRED CODE
        }
        */
        protected override void OnLoad(EventArgs e)
        {
            // Need to put this here so caption bar really goes away
            //this.WindowState = FormWindowState.Maximized;
            base.OnLoad (e);

        }
        protected override void OnActivated(EventArgs e)
        {
            Canvas.Focus();
            base.OnActivated (e);
        }
```

```
            public override void Initialize()
            {
                  ApplicationForm = this;

                  // Need to put this here so size is 240x320
                  // OnLoad hasn't happened yet
                  //this.WindowState = FormWindowState.Maximized;
                  Canvas.Size = this.Size;
                  Canvas.ZoomEventHandler = null;
                  Canvas.PanEventHandler = null;
                  Canvas.LostFocus +=new EventHandler(Canvas_LostFocus);

                  PPath shellNode = PPath.CreateRectangle(0,0, ClientRectangle.Width, ClientRectangle.Height);
                  shellNode.Brush = Constants.LIGHT_GRAY_BRUSH;
                  shellNode.Pen = null;
                  Canvas.Layer.AddChild(shellNode);

                  landscapeCamera = this.Canvas.Camera;
                  //landscapeCamera.SetBounds(0, 0, ClientRectangle.Width, ClientRectangle.Height);
                              landscapeCamera.Brush = Constants.MED_GRAY_BRUSH;
                  //          landscapeCamera.AddInputEventListener(new ThumbPanEventHandler());

                  float homeLaunchPointDiameter = (2 * Constants.TITLE_BAR_HEIGHT) + Constants.TILE_VERTICAL_SEP;
                  float appLaunchPointDiameter = 45;

                  landscapeNode = new Landscape(this, landscapeCamera, ClientRectangle.Width,
                        ClientRectangle.Height,
                        Constants.INTER_QUAD_SPACING,
                        new SizeF(homeLaunchPointDiameter,homeLaunchPointDiameter));

                  landscapeNode.X=0;
                  landscapeNode.Y=0;

                  float zoomLaunchPointDiameter = homeLaunchPointDiameter *
      (landscapeCamera.ViewBounds.Width/landscapeNode.Width);

                  //Canvas.Layer.AddInputEventListener(new LaunchPointListener(launchPoint, landscapeNode, landscapeCamera));
                  landscapeCamera.AddInputEventListener(new LaunchPointListener(launchPoint, landscapeNode, landscapeCamera));

                  PLayer landscapeLayer = new PLayer();
```

```
                landscapeLayer.AddChild(landscapeNode);
                landscapeCamera.AddLayer(landscapeLayer);
                //Canvas.Layer.AddChild(landscapeCamera);
                // Who knows about this?!?!
                Canvas.Root.AddChild(landscapeLayer);
                //landscapeCamera.AddInputEventListener(new HardwareEventHandler());
                landscapeNode.HomeQuad(false);

                Canvas.KeyDown += new KeyEventHandler(Canvas_KeyDown);

                base.Initialize ();
            }
/*
            private void harnessButtonReleased(int iEnumCode, int iKeyValue)
            {
                switch (iEnumCode)
                {
                    case HBUTTON_CONTACTS:
                        if (landscapeNode.ActiveApp != null)
                        {
                            landscapeNode.ActiveApp.KeyPressed(System.Windows.Forms.Keys.NumPad1);
                        }
                        break;
                    case HBUTTON_MAIL:
                        if (landscapeNode.ActiveApp != null)
                        {
                            landscapeNode.ActiveApp.KeyPressed(System.Windows.Forms.Keys.NumPad2);
                        }
                        break;
                }
            }
            */
            // All shell level changes go through this, but are stored in the node itself
            // That's why it's safe to change the position of the blue dot here
            public Constants.ShellLevel ShellLevel
            {
                get { return this.shellLevel; } //lodNode.ShellLevel; }
                set
                {
                    if (shellLevel != value) //lodNode.ShellLevel != value)
                {
//                  if (value == Constants.ShellLevel.APPLICATION)
//                      launchPoint.AnimateToBounds(launchPoint.ApplicationBounds.X,
launchPoint.ApplicationBounds.Y,
//                          launchPoint.ApplicationBounds.Width, launchPoint.ApplicationBounds.Height,
Constants.DEFAULT_ANIMATION_TIME);
//                  else if (value == Constants.ShellLevel.HOME)
//                      launchPoint.AnimateToBounds(launchPoint.HomeBounds.X, launchPoint.HomeBounds.Y,
//                          launchPoint.HomeBounds.Width,
launchPoint.HomeBounds.Height,Constants.DEFAULT_ANIMATION_TIME);
                    //      else if (value == Constants.ShellLevel.ZOOM_SPACE)
                    //          launchPoint.AnimateToBounds(launchPoint.ZoomBounds.X,
launchPoint.ZoomBounds.Y,
                    //              launchPoint.ZoomBounds.Width,
launchPoint.ZoomBounds.Height,Constants.DEFAULT_ANIMATION_TIME);
                    shellLevel = value;
                    //lodNode.ShellLevel = value;
                }
            }
        }

        protected void Camera_MouseUp(object sender, PInputEventArgs e)
        {
            if (e.PickedNode is PCamera)
            {
            }
            else
            {
                //                      e.Handled = true;
                landscapeNode.PanToNearest(landscapeCamera);
            }
        }
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            /*
            if (harness != null)
            {
```

```
                        harness.Dispose();
                        harness = null;
                }
                */
                base.Dispose( disposing );
        }
        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
                this.pageSetupDialog1 = new System.Windows.Forms.PageSetupDialog();
                //
                // ShellForm
                //
                this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
                this.ClientSize = new System.Drawing.Size(800, 600);
                this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.None;
                this.Location = new System.Drawing.Point(0, 0);
                this.MaximizeBox = false;
                this.MinimizeBox = false;
                this.Name = "ShellForm";
                this.StartPosition = System.Windows.Forms.FormStartPosition.Manual;
                this.Text = "Xnav2004";

        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>

        static void Main()
        {
                Application.Run(new ShellForm());
        }

        private void Canvas_KeyDown(object sender, KeyEventArgs e)
```

```
		{
			switch (e.KeyCode)
			{
				//case Keys.Enter:
				//	XnavActivated();
				//	break;
				default:
					if (landscapeNode.KeyPressed(e.KeyCode) == false)
					{
						if (e.KeyCode == Keys.Enter || e.KeyCode == Keys.NumPad5 || e.KeyCode == Keys.D5)
						{
							XnavActivated();
						}
					}
					break;
			}
		}

		public void XnavActivated()
		{
			if (landscapeNode.ShellLevel == Constants.ShellLevel.HOME)
			{
				landscapeNode.ZoomSpace();
				landscapeNode.ActiveQuad.Xnav.AnimateToMode(XnavMode.ZoomIn,
Constants.DEFAULT_ANIMATION_TIME);
			}
			else if (landscapeNode.ShellLevel == Constants.ShellLevel.ZOOM_SPACE)
			{
				landscapeNode.HomeQuad();
			}
			//else
			//{
			//	if (landscape.ActiveApp != null && landscape.ActiveApp.blueDot != null)
			//	{
			//		landscape.ActiveApp.blueDot.Clicked();
			//	}
			//}
		}

		private void Canvas_LostFocus(object sender, EventArgs e)
```

```
			{
				if (this.Focused == true)
				{
					Canvas.Focus();
				}
			}

		}


		#region Helper Classes
		class LaunchPoint : PNode
		{
			Bitmap launchImage;
			float offsetX;
			float offsetY;
			float displayWidth;
			float displayHeight;
			RectangleF homeBounds;
			RectangleF applicationBounds;
			RectangleF zoomBounds;
			protected Landscape landscape;

			public LaunchPoint(Landscape scape) : base()
			{
				landscape = scape;
				launchImage = Util.GetImage(this, "LaunchPoint.images.main_blue_dot.png");
			}
			protected override void Paint(PPaintContext paintContext)
			{
				// Use camera scale vs. zoom level because we want to show the dot between zoom levels
				if (landscape.Camera.ViewScale >= landscape.HomeScale &&
					landscape.Camera.ViewScale < landscape.ApplicationScale)
				{
					ImageAttributes attr = new ImageAttributes();
					attr.SetColorKey(launchImage.GetPixel(0,0), launchImage.GetPixel(0,0));

					//	paintContext.Graphics.FillEllipse(Constants.LAUNCH_POINT_BRUSH, Bounds.X, Bounds.Y,
Bounds.Width, Bounds.Height);
					RectangleF destRect = new RectangleF(Bounds.X+offsetX, Bounds.Y + offsetY, displayWidth, displayHeight);
```

```
						PointF[] destPoints = {destRect.Location, new PointF(destRect.Right, destRect.Y), new PointF(destRect.X,
destRect.Bottom)};
						//paintContext.Graphics.DrawImage((Image)launchImage,  destPoints, new RectangleF(0,0,launchImage.Width,
launchImage.Height), GraphicsUnit.Pixel, attr);
				}
				base.Paint (paintContext);
			}
			public Landscape Landscape
			{
				get { return landscape; }
			}
			public RectangleF HomeBounds
			{
				set { homeBounds = value;}
				get { return homeBounds; }
			}
			public RectangleF ApplicationBounds
			{
				set { applicationBounds = value;}
				get { return applicationBounds; }
			}
			public RectangleF ZoomBounds
			{
				set { zoomBounds = value;}
				get { return zoomBounds; }
			}
			protected override void InternalUpdateBounds(float x, float y, float width, float height)
			{
				offsetX = Math.Max(0,(width - launchImage.Width)/2);
				offsetY = Math.Max(0,(height - launchImage.Height)/2);
				if (width > launchImage.Width)
				{
					displayWidth = launchImage.Width;
				}
				else
				{
					displayWidth = width;
				}
				if (height > launchImage.Height)
				{
					displayHeight = launchImage.Height;
				}
				else
				{
					displayHeight = height;
				}
				base.InternalUpdateBounds (x, y, width, height);
			}

			public void Clicked()
			{
				// Not sure it should be responding to events lower down
				// Depending on shell state, do something
				// HOME level = zoom out
				// APP level - who knows
			}
		}

		class CloseNode : UMD.HCIL.Piccolo.Nodes.PImage
		{
			PForm form;
			public CloseNode(PForm form) : base()
			{
				this.form = form;
				Image = Util.GetImage(this, "LaunchPoint.images.close.png");
			}
			public override void OnClick(PInputEventArgs e)
			{
				base.OnClick (e);
				form.Close();
			}

		}
		class StatusNode : PNode
		{
			Bitmap statusImage;
			public StatusNode () : base ()
			{
				statusImage = Util.GetImage(this, "LaunchPoint.images.status.png");
			}
		protected override void Paint(PPaintContext paintContext)
		{
			if (statusImage != null)
			{
				SizeF destSize = new SizeF(0,0);
				SizeF sourceSize = new SizeF(0,0);
				Util.GetBestDestSourceSize(new SizeF(Bounds.Width, Bounds.Height), statusImage.Size, ref destSize, ref
sourceSize);

				paintContext.Graphics.DrawImage((Image) statusImage, new RectangleF(Bounds.X+Bounds.Width-
destSize.Width, Bounds.Y, destSize.Width, destSize.Height), new RectangleF(0, 0, sourceSize.Width, sourceSize.Height), GraphicsUnit.Pixel);
			}
			else
			{
				base.Paint (paintContext);
			}
		}

	}
	class LODNode : PNode
	{
		Constants.ShellLevel shellLevel;
		Bitmap lodStrip;
		ImageNode lodIndicator;

		public LODNode(Constants.ShellLevel startLevel) : base()
		{
			lodStrip = Util.GetImage(this, "LaunchPoint.images.lodstrip.png");
			lodIndicator = new ImageNode(Util.GetImage(this, "LaunchPoint.images.lodindicator.png"));
			lodIndicator.DoScale = true;
			AddChild(lodIndicator);
			lodIndicator.InitialBounds = new RectangleF(Bounds.X, Bounds.Y + (int) startLevel * lodIndicator.Bounds.Height,
lodIndicator.Image.Width, lodIndicator.Image.Height);
		}
		protected override void InternalUpdateBounds(float x, float y, float width, float height)
		{
			float scale = height/lodStrip.Height;
			float newLodHeight = lodIndicator.Bounds.Height * scale;
			lodIndicator.SetBounds(x, y + (int) shellLevel * newLodHeight, lodIndicator.Bounds.Width, newLodHeight);
			base.InternalUpdateBounds (x, y, width, height);
		}

		public Constants.ShellLevel ShellLevel
		{
			get { return shellLevel; }
			set
			{
				shellLevel = value;
				lodIndicator.AnimateToBounds(Bounds.X, Bounds.Y + ((int) value * lodIndicator.Bounds.Height),
lodIndicator.Bounds.Width, lodIndicator.Bounds.Height, Constants.DEFAULT_ANIMATION_TIME);
				/*
				switch (value)
				{
					case Constants.ShellLevel.ZOOM_SPACE:
						lodIndicator.AnimateToBounds(lodIndicator.Bounds.X, lodIndicator.Bounds.Y + (value *
lodIndicator.Image.Height), lodIndicator.Bounds.Width, lodIndicator.Bounds.Height, Constants.DEFAULT_ANIMATION_TIME);
						break;
					case Constants.ShellLevel.HOME:
						currentImage = home;
						break;
					case Constants.ShellLevel.APPLICATION:
						currentImage = application;
						break;
					case Constants.ShellLevel.OBJECT:
						currentImage = obj;
						break;
					case Constants.ShellLevel.CONTEXT:
						currentImage = context;
						break;
					case Constants.ShellLevel.INPUT:
						currentImage = input;
						break;
				}

				this.Repaint();
				*/
			}
		}
		protected override void Paint(PPaintContext paintContext)
		{
					if (lodStrip != null)
					{
						SizeF destSize = new SizeF(0,0);
						SizeF sourceSize = new SizeF(0,0);
						Util.GetBestDestSourceSize(new SizeF(Bounds.Width, Bounds.Height), lodStrip.Size, ref destSize, ref
sourceSize);

						paintContext.Graphics.DrawImage((Image) lodStrip, new RectangleF(Bounds.X, Bounds.Y, destSize.Width,
destSize.Height), new RectangleF(0, 0, sourceSize.Width, sourceSize.Height), GraphicsUnit.Pixel);
					}
					else
					{
						base.Paint(paintContext);
					}
				}
			}

			class LandscapeCamera:PCamera{}
			class LaunchPointListener : PPanEventHandler {
				LaunchPoint launchPoint;
				Landscape landscape;
				//private bool launchPtMouseDown = false;
				//PointF mouseDownPoint = new PointF(-1,-1);
				PCamera camera;
				Constants.NavigationDirection direction = Constants.NavigationDirection.NONE;
				public LaunchPointListener(LaunchPoint launchNode, Landscape scape, PCamera landscapeCamera) : base () {
					launchPoint = launchNode;
					landscape = scape;
					camera = landscapeCamera;
					this.Autopan = false;
				}
				protected override bool ShouldStartDragInteraction(PInputEventArgs e) {
					if (landscape.ShellLevel == Constants.ShellLevel.HOME)
						return PUtil.DistanceBetweenPoints(MousePressedCanvasPoint, e.CanvasPosition)
							>= Constants.CLICK_THRESHOLD;
					else
						return false;
				}
```

```
        protected override void Pan(PInputEventArgs e) {
                //e.TopCamera.Canvas.PaintImmediately();

                if (direction == Constants.NavigationDirection.NONE) {
                        if (Math.Abs(MousePressedCanvasPoint.Y-e.CanvasPosition.Y) > Math.Abs(MousePressedCanvasPoint.X-
    e.CanvasPosition.X)) {
                                direction = Constants.NavigationDirection.VERTICAL;
                        }
                        else if (Math.Abs(MousePressedCanvasPoint.Y-e.CanvasPosition.Y) < Math.Abs(MousePressedCanvasPoint.X-
    e.CanvasPosition.X)) {
                                direction = Constants.NavigationDirection.HORIZONTAL;
                        }
                }
                //else
                if (direction == Constants.NavigationDirection.VERTICAL) {

                        float targetTopY = camera.ViewBounds.Y - (camera.ViewScale * e.Delta.Height *
    Constants.NAVIGATION_MULTIPLIER);
                        float targetBottomY = camera.ViewBounds.Y + camera.ViewBounds.Height - (camera.ViewScale * e.Delta.Height
    * Constants.NAVIGATION_MULTIPLIER);
                        if (0 <= targetTopY && landscape.Height >= targetBottomY) {
                                camera.TranslateViewBy(0, e.Delta.Height * Constants.NAVIGATION_MULTIPLIER);
                        }
                }
                else if (direction == Constants.NavigationDirection.HORIZONTAL) {
                        float targetRightX = camera.ViewBounds.X - (camera.ViewScale * e.Delta.Width *
    Constants.NAVIGATION_MULTIPLIER);
                        float targetLeftX = camera.ViewBounds.X + camera.ViewBounds.Width - (camera.ViewScale *e.Delta.Width *
    Constants.NAVIGATION_MULTIPLIER);

                        if (0 <= targetRightX && landscape.Width >= targetLeftX) {
                                camera.TranslateViewBy(e.Delta.Width * Constants.NAVIGATION_MULTIPLIER, 0);
                        }
                }
        }

        /*
                        public override void OnMouseDown(object sender, PInputEventArgs e)
                        {
                                // this listener only presides over HOME and Zoom Level
                        if (landscape.ShellLevel <= Constants.ShellLevel.HOME)
                        {
                                SanityCheck();

                                // It's any direction's game at this point
                                direction = Constants.NavigationDirection.NONE;

                                if (PUtil.RectangleContainsPoint(launchPoint.Bounds, e.CanvasPosition))
                                {
                                        launchPtMouseDown = true;
                                        // If this is over the launch point, eat the event
                                        //e.Handled = true;
                                }
                                else
                                {
                                        launchPtMouseDown = false;
                                }
                                mouseDownPoint = e.CanvasPosition;
                                // deed this to register drag event
                                base.OnMouseDown (sender, e);
                        }
                }
*/
/*
protected override void OnDrag(object sender, PInputEventArgs e)
{
        // this listener only presides over HOME and Zoom Level
        if (landscape.ShellLevel <= Constants.ShellLevel.HOME)
        {
                float distance = PUtil.DistanceBetweenPoints(mouseDownPoint, e.CanvasPosition);
                // If the movement was in the launchpoint and it is a candidate for a click, make it
                // exceed the click threshold before doing anything
                if (direction == Constants.NavigationDirection.NONE &&
                        //(PUtil.RectangleContainsPoint(launchPoint.Bounds, e.CanvasPosition) ||
                        //PUtil.RectangleContainsPoint(launchPoint.Bounds, e.CanvasPosition) ||
                        //&& launchPtMouseDown &&
                        distance <= Constants.CLICK_THRESHOLD)
                {
                        // do nothing
                }
                else
                {
                    if (direction == Constants.NavigationDirection.NONE)
                    {
                        if (Math.Abs(mouseDownPoint.Y-e.CanvasPosition.Y) > Math.Abs(mouseDownPoint.X-
e.CanvasPosition.X))
                        {
                            direction = Constants.NavigationDirection.VERTICAL;
                        }
                        else if (Math.Abs(mouseDownPoint.Y-e.CanvasPosition.Y) < Math.Abs(mouseDownPoint.X-
e.CanvasPosition.X))
                        {
                            direction = Constants.NavigationDirection.HORIZONTAL;
                        }
                    }
                    else if (direction == Constants.NavigationDirection.VERTICAL)
                    {

                        if (landscape.ShellLevel == Constants.ShellLevel.HOME)
                        {
                            float targetTopY = camera.ViewBounds.Y - (camera.ViewScale * e.Delta.Height *
Constants.NAVIGATION_MULTIPLIER);
                            float targetBottomY = camera.ViewBounds.Y + camera.ViewBounds.Height -
(camera.ViewScale * e.Delta.Height * Constants.NAVIGATION_MULTIPLIER);
                            if (0 <= targetTopY && landscape.Height >= targetBottomY)
                            {
                                camera.TranslateViewBy(0, e.Delta.Height *
Constants.NAVIGATION_MULTIPLIER);
                            }
                        }
                    }
                    else if (direction == Constants.NavigationDirection.HORIZONTAL)
                    {
                        if (landscape.ShellLevel == Constants.ShellLevel.HOME)
                        {
                            float targetRightX = camera.ViewBounds.X - (camera.ViewScale * e.Delta.Width *
Constants.NAVIGATION_MULTIPLIER);
                            float targetLeftX = camera.ViewBounds.X + camera.ViewBounds.Width -
(camera.ViewScale *e.Delta.Width * Constants.NAVIGATION_MULTIPLIER);
```

```
                                                        if (0 <= targetRightX && landscape.Width >= targetLeftX)
                                                        {
                                                                camera.TranslateViewBy(e.Delta.Width *
Constants.NAVIGATION_MULTIPLIER, 0);
                                                        }
                                                }
                                        }
                                }
                                e.Handled = true;
                                //base.OnDrag (sender, e);
                        }
                }
*/
                public override void OnMouseUp(object sender, PInputEventArgs e) {
                        // this listener only presides over HOME and Zoom Level
                        // would have liked to use lanscape.ShellLevel < Constants.ShellLevel.HOME, but set
                        // too quickly
                        if (landscape.Camera.ViewScale <= (int) landscape.HomeScale) {

                                // If we didn't go far enough to drag and the mouse up occured over the launch point,
                                // consider it a launchpoing click
                                if (!Dragging) {
                                        //      if (launchPoint.Bounds.Contains(e.CanvasPosition))
                                        //PUtil.RectangleContainsPoint(launchPoint.Bounds, e.CanvasPosition))
                                        //      {
                                        //              launchPoint.Clicked();
                                        //      }
                                        // otherwise could have been a click at home level or any mouse up at another level
                                        //      else
                                        //      {

                                        PNode picked = e.PickedNode;
                                        if (picked is AppTile) {
                                                ((AppTile)picked).Clicked();
                                        }
                                        else if (picked is QuadTile) {
                                                ((QuadTile)picked).Clicked();
                                        }
                                        else if (picked is NavigationNode) {
                                                ((NavigationNode)picked).Clicked();
                }
                //      }
        }// Else it was a drag and we should snap to grid and it must have
                //occurred at the HOME level
        else {

                //landscape.PanToNearest(this.camera);
                if (direction == Constants.NavigationDirection.HORIZONTAL) {
                        if (Math.Abs(MousePressedCanvasPoint.X-e.CanvasPosition.X) >
(camera.ViewBounds.Width/6)) {
                                if (MousePressedCanvasPoint.X > e.CanvasPosition.X) {
                                        landscape.NavigateRight();
                                }
                                else {
                                        landscape.NavigateLeft();
                                }
                        }
                        else {
                                landscape.AdjustToCurrent(true);
                        }

                }
                else if (direction == Constants.NavigationDirection.VERTICAL) {
                        if (Math.Abs(MousePressedCanvasPoint.Y-e.CanvasPosition.Y) >
(camera.ViewBounds.Height/6)) {
                                if (MousePressedCanvasPoint.Y > e.CanvasPosition.Y) {
                                        landscape.NavigateDown();
                                }
                                else {
                                        landscape.NavigateUp();
                                }
                        }
                        else {
                                landscape.AdjustToCurrent(true);
                        }
                }
                else {
                        landscape.AdjustToCurrent(true);
                }
```

```
                        }

                        direction = Constants.NavigationDirection.NONE;
                        //launchPtMouseDown = false;
                    }
                    base.OnMouseUp(sender, e);
                }
                private void SanityCheck() {
                    if (landscape.Camera.ViewScale == landscape.HomeScale) {
                        landscape.ShellLevel = Constants.ShellLevel.HOME;
                    }
                    else if (landscape.Camera.ViewScale == landscape.ApplicationScale) {
                        landscape.ShellLevel = Constants.ShellLevel.APPLICATION;
                    }
                    else if (landscape.Camera.ViewScale == landscape.ZoomScale) {
                        landscape.ShellLevel = Constants.ShellLevel.ZOOM_SPACE;
                    }
                }
            }

            #endregion
        }
```