Navigation2D: A Directional Navigation Module for XMonad

Norbert Zeh

December 5, 2011

Abstract

Directional navigation feels more natural than flipping through the windows in sequential order, either in the history or in some fixed sequence such as xmonad's window stack. On the other hand, it is hard to get completely right. This document first explores the requirements that useful directional navigation strategies should satisfy and then discusses the two navigation strategies implemented in Navigation2D. These navigation strategies are positioned at the extreme ends of the trade-off between generality of the navigation strategy and how natural it feels. The user can then choose which of these strategies satisfies their needs and even specify different strategies to be used depending on the layout currently in use on a given workspace.

1 Directional Navigation

Directional navigation refers to setting up keybindings that allow the user to navigate to the window to the left, to the right, above or below the current window using just their keyboard, without involvement of the mouse. If there are many windows on the screen (or on multiple screens), this is certainly more convenient than the standard [Alt-Tab] approach in traditional window managers or the focusUp/focusDown method of sequentially traversing the window stack in xmonad.[1] However, it's hard to get right. The problem is that there are different possible definitions of what it means to "go to the left/to the right/up/down". Some of them feel more natural, others are more general. Here are the two requirements directional navigation should meet (in an ideal world) or at least try to meet (and let the user choose in full awareness of the trade-off involved).

Completeness: The above definition intentionally said that directional navigation has to allow the user to switch between windows *without using the mouse*. Moving the mouse pointer to the desired window is always possible as a last resort, but if the user is willing to do this *frequently*, then they are probably not all that keen on a powerful way to navigate between windows using the keyboard in the first place. So we state as the first requirement that it has to be possible to move from any window to any other window using only directional navigation operations.

Naturality: This is a more fuzzy requirement. It simply states that when the user presses the key that means "move left", they certainly do not expect the focus to move to a window that feels more like its above, below or to the right of the current window than it seems to be to the left of the current window. Left should really feel like left. Right should really feel like right. Etc.

The next section discusses the two directional navigation strategies implemented in Navigation2D: *line navigation* and *center navigation*. The section illustrates that, even for tiled window layouts, the

[1] [Alt-Tab] allows going back to windows based on how recently they were used and may be more convenient than directional navigation if these windows are far apart. This should and can be set up using separate keybindings, for example using XMonad.Actions.GroupNavigation.

former is not always complete, while the latter is complete for any layout but may feel unnatural. The section also proves that for most reasonable tiled layouts, line navigation is complete, and it will be obvious that it is a very natural directional navigation strategy.

A third requirement, which the xmonad community has grappled with for a while and which is even more important in the context of directional navigation, is

Support for layers: In a tiling window manager, it is natural to consider windows to live in one of two layers: the tiled layer and the floating layer. Navigation should be confined to the current window's layer.

The reason this is important in the context of directional navigation is that the tiled layout in use on a given workspace may be sufficiently well structured to guarantee that even line navigation is complete. So, for this workspace it is possible to use a directional navigation strategy that is both very natural and complete. On the other hand, one should not assume anything about the relative positions of windows in the floating layer. So the user may wish to use different directional navigation strategies to navigate among floating windows and to navigate among tiled windows. Navigation2D allows the specification of different navigation strategies for the floating layer and the tiled layer. Within the tiled layer, it allows the choice of different strategies depending on the layout currently in use. The justification is that the user should be able to choose whether completeness or a natural feel is more important to them. Maybe they want completeness but, within the constraints this imposes, still want the navigation to feel as natural as possible. Then this requires different directional navigation strategies for different layouts, and different navigation strategies for the tiled and floating layers.

2 The Two Navigation Strategies Implemented in Navigation2D

We discuss the two directional navigation strategies implemented in Navigation2D using navigation to the left from the current window as the running example. The other three directions are handled analogously.

2.1 Line Navigation

The first directional navigation strategy implemented in Navigation2D is called *line navigation* and is illustrated in Figure 1. Using this strategy, we find the next window to the left of the current window as follows. We draw a horizontal line ℓ through the center of the current window. A window is a candidate window to receive the focus only if it intersects this line. Now we sort these candidate windows by the x-coordinates of their right boundaries and choose the rightmost one whose right boundary is to the left of the current window's left boundary. If such a window exists, focus moves to this window. Otherwise focus stays with the current window.

This is a very stringent notion of moving left, right, up or down, and it certainly feels natural. Unfortunately, even when windows do not overlap in a tiling layout, line navigation is not complete: in Figure 2 it is impossible to navigate from any of the peripheral windows to the window in the center. However, we can prove that this navigation strategy is complete if the layout satisfies a natural condition satisfied by the vast majority of standard layouts available in xmonad. We call a layout *tree-like* if it partitions the workspace into non-overlapping rectangles assigned to the windows in the layout and can be represented using a binary tree as follows. The leaves of the tree are the windows in the layout. For every internal node, there exists a horizontal or vertical line that separates the windows in the left subtree from the windows in the right subtree. Note that we do not require the layout to represent the arrangement of windows in this fashion. We require only that the layout *can* be represented this way.

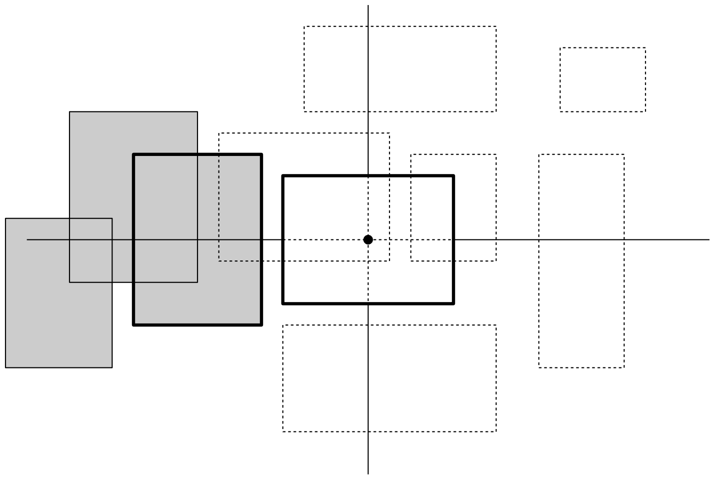

Figure 1: Illustration of line navigation. The current window is shown with a fat border. The windows that are candidates to navigate to are shaded grey. The one whose right boundary is closest to the current window (and the one that receives focus) is shown with a fat border.

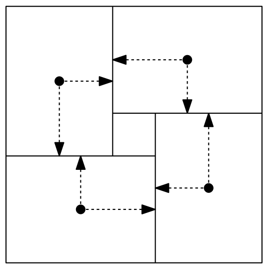

Figure 2: The four peripheral windows can navigate between each other using line navigation, but the window in the middle cannot be reached from any of them.

Lemma 1. *Line navigation is complete for tree-like layouts.*

Proof. Consider a tree representation of the layout. We prove by induction on the size of the subtree that any two windows in the same subtree can reach each other. The base case is when the subtree contains only one leaf. In this case, the claim is true because nothing needs to be done to navigate from a window to itself.

So consider a subtree with root r and two subtrees L and R, and let w_1 and w_2 be any two windows in r's subtree. We prove that we can navigate from w_1 to w_2. If $w_1, w_2 \in L$ or $w_1, w_2 \in R$, this is true by the induction hypothesis. So we can assume without loss of generality that $w_1 \in L$ and $w_2 \in R$. We can further assume that the line ℓ separating L and R is vertical. Let w_1' be a window in L whose right boundary is part of ℓ. Since the layout is tree-like, such a window exists. For the same reason, there exists a window $w_2' \in R$ whose left boundary is part of ℓ and which intersects the horizontal line through the center of w_1'. By the induction hypothesis, we can navigate from w_1 to w_1' and from w_2' to w_2 because they belong to the same subtrees. We can also navigate from w_1' to w_2' because w_2' is the window with the leftmost left boundary among the windows that intersect the horizontal line through the center of w_1' and whose left boundaries are to the right of the right boundary of w_1'. (Since no two windows in the layout overlap, there cannot be any other window that also intersects the horizontal line through the center of w_1' and whose left boundary coincides with the left boundary of w_1'.) This shows that there exists a path to navigate from w_1 to w_2 via w_1' and w_2'. $\square$

2.2 Center Navigation

It is easy to define complete directional navigation strategies, but they all feel fairly unnatural. The author chose to implement the following strategy in Navigation2D, which we call *center navigation*.[2] It appears to be one of the most natural navigation strategies that can easily be shown to be complete for any layout. To navigate to the left, we consider the cone bounded by the two rays emanating from the center of the current window at 45° angles in the north-west and south-west directions. For a window to be a candidate to receive the focus, its center must lie inside this cone. A center on the bottom ray is not considered to be in the cone, while a center on the top ray is considered to be in the cone. A center that coincides with the center of the current window may or may not be in the cone. When considering the left cone, as we do here, or the top cone, a window center that coincides with the center of the current window is in the cone if it precedes the current window in the window stack. When considering the right or bottom cone, such a window is in the cone if it succeeds the current window in the window stack. To continue the definition of navigating to the left: If there is no window in the cone, focus stays with the current window. Otherwise we sort the center points primarily by increasing L_1-distance from the current window's center, secondarily by increasing angles between the right boundary of the cone and the rays from the current window center through these center points and, as a third criterion, by decreasing position in the window stack. Focus moves to the first window in this sorted list. This navigation strategy is illustrated in Figure 3. Center navigation

[2]This is the same strategy as used by openbox, augmented with a tie breaking rule for equally ranked windows that ensures that every window is reachable.

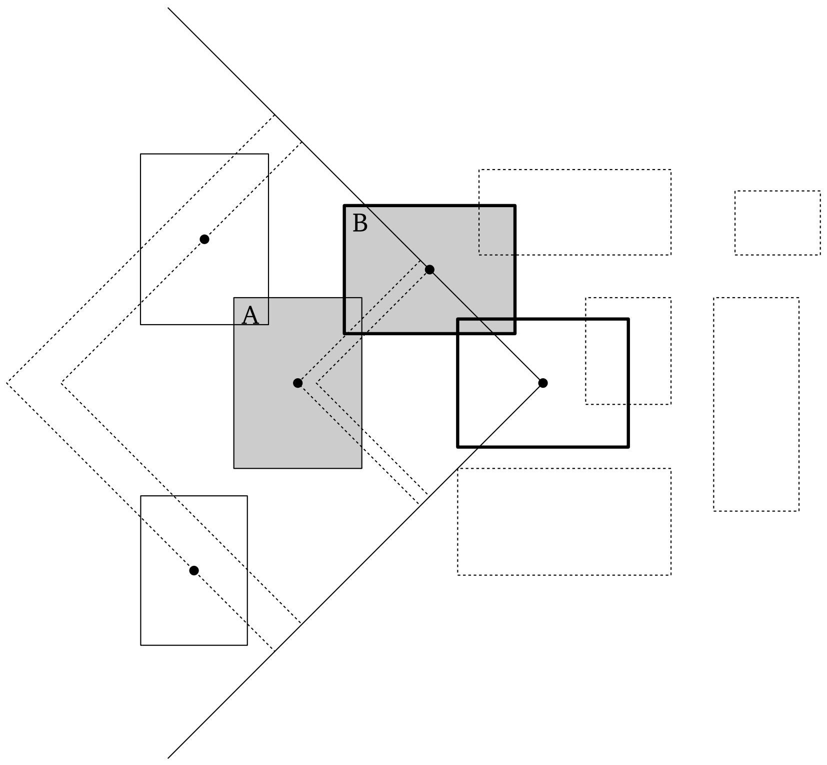

Figure 3: Illustration of center navigation. It may lead to unnatural behaviour as shown here, where one would probably expect window A to receive the focus but window B receives the focus instead. Nevertheless, focus never moves to a window that feels more like it's above or below the current window than it is to its left.

may not feel as natural as line navigation, as illustrated in the figure. However, as already mentioned, it has the following desirable property, which makes it suitable as a general navigation strategy even for the floating layer.

Lemma 2. *Center navigation is complete for any layout, even those that allow windows to overlap.*

Proof. Since the navigation of windows is defined entirely using their centers, we refer to windows by their center points in this proof. We show that using center navigation, we can navigate from any point p to any other point q. To navigate from p to q, we use the following natural strategy. Choose a directional cone of p that contains q. If p and q do not coincide, this cone is unique. Otherwise q is in the top and left cones or in the bottom and right cones; in this case, we choose the cone arbitrarily. Since there is at least one candidate point in the chosen cone, namely q, navigating from p in the corresponding direction leads to a point $p' \neq p$. If $p' = q$, we are done. Otherwise we make p' the current point and iterate this strategy.

Let $p = p_1, p_2, \ldots$ be the sequence of points we traverse using this strategy. We call navigating from p_i to p_{i+1} a *step*. We call a step *stationary* if p_i and p_{i+1} coincide. We prove the lemma by showing that the sequence $p_1, p_2, \ldots$ contains every window center at most once. This implies that the sequence is finite. If q were not part of the sequence, on the other hand, the sequence would have to be infinite because, as we argued above, for every point $p_i \neq q$, there exists a point in the same directional cone of p_i as q, that is, we can choose a successor p_{i+1} for p_i in the sequence. Together, these two observations imply that q must be part of the sequence, that is, we reach q after a finite number of navigation steps.

The proof that every window center appears at most once in the sequence $p_1, p_2, \ldots$ is based on the following three claims.

Claim 1. *For all i such that $p_i \neq q$, $||p_{i+1} - q||_1 \leq ||p_i - q||_1$.*

Proof. Consider the L_1-circle O with radius $r := ||p_i - q||_1$ around q (see Figure 4). This circle is divided into four quarter-circles L, R, T, and B as shown in the figure. We can assume that p_i lies on the periphery of L and, thus, that q and p_{i+1} lie in p_i's right directional cone C. Since $||p_{i+1} - p_i||_1 \leq ||q - p_i||_1 = r$, p_{i+1} lies in the intersection of C and the L_1-circle O' with radius r around p_i. This intersection, however, is completely contained in O, which implies that $||p_{i+1} - q||_1 \leq r = ||p_i - q||_1$. □

Claim 2. *If $||p_i - q||_1 = ||p_{i+1} - q||_1$, then p_i and p_{i+1} coincide or p_{i+1} is located on the right boundary ray of the directional cone of p_i that contains q.*

Proof. We use the same notation as in the proof of Claim 1. Observe that $||p_i - q||_1 = ||p_{i+1} - q||_1$ implies that p_{i+1} lies on the periphery of O. Now we distinguish four cases.

If p_i and p_{i+1} coincide, the claim holds. So assume they do not coincide.

If p_i lies on the north-west to south-east line through q (point p_1 in Figure 4), then the only points other than p_i on the periphery of O and in the intersection of O' and C lie on the line segment s_1 in the figure. Any point on s_1, however, has the same L_1-distance from p_i as q, and q is the point in C with this L_1-distance from p_i that minimizes the angle to the right boundary ray of C. This contradicts the choice of p_{i+1} as the successor of p_i. Thus, this case cannot arise.

If p_i lies on the segment s_2 in the figure and does not coincide with p_1 (e.g., point p_2), then the intersection of C and O' lies *strictly* inside O, that is, $||p_{i+1} - q||_1 < ||p_i - q||_1$ unless p_i and p_{i+1} coincide. Thus, this case cannot arise either.

The only remaining option is that p_i lies on the segment s_3 in the figure (e.g., point p_3). In this case, all points in the intersection of O' and C and on the periphery of O lie on the right boundary ray of C, that is, the claim holds. □

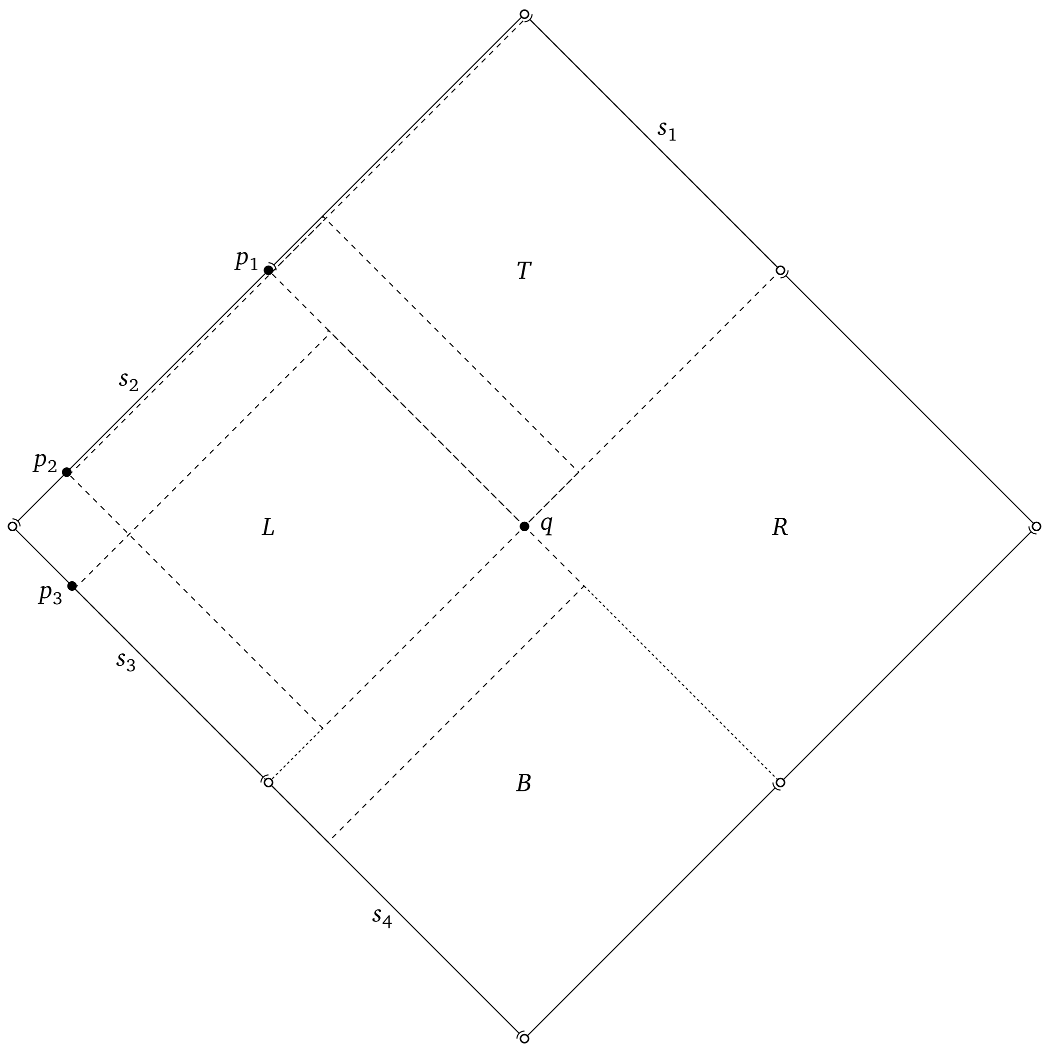

Figure 4: Proof of Lemma 2.

Claim 3. *Every sequence of consecutive stationary steps that visit a sequence of points $p_i, p_{i+1}, \dots, p_j$ traverse the window stack in the same direction.*

Proof. If q does not coincide with p_i, then q is contained in the same unique directional cone of all points $p_i, p_{i+1}, \dots, p_{j-1}$. Assume w.l.o.g. that this is the right cone of p_i. Then, for all $i \leq h < j$, p_{h+1} belongs to this cone only if it succeeds p_h in the window stack. Thus, each of these steps moves forward in the window stack.

If q coincides with p_i, assume w.l.o.g. that q succeeds p_i in the window stack. If q succeeds p_h in the window stack, for all $i \leq h < j$, then again p_{h+1} belongs to the same two cones of p_h as q only if p_{h+1} succeeds p_h in the window stack. Thus, once again each step moves forward in the window stack.

So assume there exists an h such that q precedes p_h in the window stack. If we choose the minimum such h, then $h > i$ and q succeeds p_{h-1} in the window stack. In particular, q appears between p_{h-1} and p_h in the window stack. This, however, contradicts the choice of p_h, as it is the first window center after p_{h-1} in the window stack and which coincides with p_{h-1}. $\square$

To finish the proof of the lemma, we show that these three claims imply that the sequence $p_1, p_2, \dots$

visits every window center at most once. Assume the contrary, and consider two occurences p_i and p_j of the same window center p. By Claim 1, all points $p_i, p_{i+1}, \dots, p_j$ are equidistant from q.

If p and q coincide, then $p_i, p_{i+1}, \dots, p_j$ coincide with q, and the steps traversing points $p_i, p_{i+1}, \dots, p_j$ are stationary. Thus, by Claim 3, we have $p_i \neq p_j$, a contradiction.

So assume p and q do not coincide. Then q is contained in a unique directional cone of p. Assume w.l.o.g. that it is the right cone of p. Then p belongs to the boundary of the left quarter circle L of circle O in Figure 4. Now let us break the sequence of points $p_i, p_{i+1}, \dots, p_j$ into maximal contiguous groups of coinciding points; that is, the steps between the points in each group are stationary, while the steps between points in consecutive groups are non-stationary. Denote these groups by $P_1, P_2, \dots, P_k$. By Claim 3, no group P_h contains any window center more than once. Thus, since $p \in P_1$ and $p \in P_k$, we have $k \geq 2$. To finish the proof, we show that no two points in different groups P_h and $P_{h'}$, $h < h'$, coincide, which contradicts that $p \in P_1$ and $p \in P_k$.

Let $1 \leq k' \leq k$ be maximal such that, for all points in groups $P_1, P_2, \dots, p_k'$, q lies in their right directional cone. For $1 \leq h \leq k'$, let p_{i_h} be the last point in group P_h. By Claim 2, since p_{i_h} and $p_{i_{h+1}}$ do not coincide, for all $1 \leq h \leq k'$, $p_{i_{h+1}}$ lies on the right boundary ray of the right directional cone of p_{i_h}. Furthermore, the proof of Claim 2 shows that, for all $1 \leq h < k'$, p_{i_h} lies on segment s_3 in Figure 4. Together, these two facts show that points $p_{i_1}, p_{i_2}, \dots, p_{i_{k'}}$ appear in order along s_3, that is, two points in different groups among $P_1, P_2, \dots, P_{k'}$ do not coincide.

If $k' = k$, this finishes the proof. If $k' < k$, the first point $p_{j'}$ of $P_{k'+1}$ belongs to segment s_4 in Figure 4 and, thus, does not coincide with any point in groups $P_1, P_2, \dots, P_{k'}$. Furthermore, observe that s_4 plays the same role for the bottom quarter circle B of O as s_2 plays for the left quarter circle L of O. Thus, as shown in the proof of Claim 2, it is impossible to take a step from $p_{j'}$ towards q and which does not decrease the L_1-distance from q. This shows that $j' = j$, a contradiction because $p_{j'}$ and $p_i \in P_1$ do not coincide. □

3 The Complete Set of Operations Supported by Navigation2D

The previous section discussed directional navigation using moving between windows as an example. In a tiling window manager, however, any navigation strategy is of limited use if it cannot also be used to rearrange windows in the layout. Moreover, in multi-monitor setups, moving across multiple monitors using only directional navigation on the windows may be tedious. Thus, the same navigation strategies should be supported to move more quickly between monitors. The following is the list of operations supported by Navigation2D.

switchLayer: Switches focus between the tiling and the floating layer. Focus moves to the window in the other layer whose center is closest to the center of the current window.

windowGo: Moves focus to the window in the given direction. The navigation strategy used depends on the layer in use. If the current window is in the floating layer, then the strategy for the floating layer is used. If the current window is in the tiled layer, then the most general strategy among the strategies defined for all visible workspaces is used.

windowSwap: Choose a window in the same way as windowGo but instead of moving focus to this window, the window is swapped with the current window.

screenGo: Moves focus to the currently focused window of the workspace on the screen in the given direction from the current screen. To find this screen, the navigation strategy defined for screen navigation is used.

screenSwap: Chooses a screen in the same way as screenGo does but swaps the workspace on this screen with the workspace on the current screen instead of moving focus to the chosen screen.

windowToScreen: Chooses a screen in the same way as screenGo does but moves the currently focused window on the current screen to the chosen screen. The window is inserted before the currently focused window on the chosen screen.

4 A Final Question

Why yet another directional navigation module for xmonad? There already exist X.A.WindowNavigation and X.L.WindowNavigation. The answer is essentially the same as for all xmonad modules that see the light of day: the existing modules didn't quite do what I wanted.

X.A.WindowNavigation says in its documentation that it is experimental, and it certainly felt that way when I played with it. Sometimes when navigating between screens, the focus jumped in unexpected ways. I then looked at the code and decided I didn't want to understand the notion of a current point stored in the module's state. This point was guaranteed to be a point inside the current window, and the behaviour of the navigation strategy depended on where this point was. This is a poor choice in my mind because the user has no visual cue of where this magic point is and, thus, cannot predict the result of the next navigation operation.

X.A.WindowNavigation grew out of X.L.WindowNavigation. The difference between the two is that the latter is implemented as a layout modifier and, thus, does not support navigation across screens. A feature X.L.WindowNavigation provides (and X.A.WindowNavigation and X.A.Navigation2D don't) is highlighting of the windows the focus will move to next in each direction. This is useful but I did not care enough about it to implement it in Navigation2D. Moreover, as it is currently implemented in X.L.WindowNavigation, it's broken. There's an easily reproducible bug in recolouring windows that results in the previously focused window and the currently focused window having the border colour of the focused window.

Neither of the two WindowNavigation modules supports separate navigation of the tiled and float layers. In fact, handling of floating windows is highly inconsistent in that it is possible to navigate from floating windows to tiled windows but there is no way to go from a tiled window to a floating window.

Neither of the two WindowNavigation modules supports customization of the navigation strategy.

Finally, neither of the two WindowNavigation modules supports the application of the same navigation strategies to screens instead of windows.