KNOTSWITHSTANDING!

By

Erik S. Nestor

An Honors Project Submitted in Partial Fulfillment

of the Requirements for Honors

in

The Department of Mathematics and Computer Science

Faculty of Arts and Sciences

Rhode Island College

2015

Acknowledgements

Copious thanks to Dr. Robert Ravenscroft for wisdom, support, and patience through many added functionalities and last-minute implementation changes. Thanks to Dr. John Burke for inspiration and feedback.

Thanks also to my Rhode Island College colleagues and professors, especially Doctors Kathryn Sanders and Ed McDowell taught me almost everything I know in addition to sitting on the advisory board for this project.

And finally, thanks to Sebastian Edward Nestor for continually inspiring me to better myself, and to my wife, Toni Fellela, for sustaining my life while I sat in a dark room for three years swearing at a laptop.

Table of Contents

Pictures and Diagrams

1. Introduction and Overview

Knotswithstanding! is an interactive pedagogical tool intended to demonstrate fundamental properties of mathematical knots. It was conceptualized and designed by Erik Nestor, an undergraduate computer science major at Rhode Island College under the mentorship of Dr. Robert Ravenscroft as an honors project for the 2014-15 academic year. The tile set used in Knotswithstanding! was suggested by Dr. John Burke of the Rhode Island College Mathematics Department. Dr. Burke also deserves credit as the primary motivator for this project as it was begun as a response to his expressed need for a system which could demonstrate introductory level knotting concepts in the classroom.

Computational representation of knot theory problems have travelled a slow and tentative road, with knot theory's principal problems remaining unsolved. Significant work in graphical representation has been done by Robert Scharein of the University of British Columbia in his 1998 doctoral thesis *Interactive Topological Drawing*. His accompanying software package KnotPlot is an OpenGL utility which allows the user to render and manipulate 3D representations of mathematical knots. It and selected 3D renderings of particularly eye-catching knots are available for download at knotplot.com. The site also offers a partially crippled version of the software for limited evaluation.

Knotswithstanding!, by contrast, will be hosted online and run in a browser, making it immediately available in any internet-ready classroom. It has a distinct pedagogical focus, and a stylistic slant. Stylistically it is influenced by 'square' knot depictions, in which mathematical knots are represented using only vertical and horizontal strokes. The tile system used is one in which the knots are constructed by arranging tiles representing common knot 'fragments' (lines, corners, crossings, and their respective rotations). This tile system, in addition to looking fabulous, helps to normalize user input so it can be more readily parsed.

Knotswithstanding! allows the user to build knots through either mouse or keyboard entry. Full editing capabilities are available, including block selection, shifting, rotation and deletion. Three different tile size options are available, offering flexibility to the user in regards to display preferences. Functionality is also available to analyze a valid

knot by traversing and delivering the number of crossings, and a Dowker notation uniquely identifying it. Tricolorability can be checked and, if it is determined that the knot is tricolorable, the coloring is performed. If it is not tricolorable, no coloring is performed and a message is displayed indicating the given knot's lack of tricolorability.

Knotswithstanding! contains 500 cached registers which can each contain a unique drawing. Any two registers can be compared to determine if their Dowker sequences are identical. Additionally, playback functionality is available to iterate through populated registers, displaying each and thus "playing" flipbook-style animations which are capable of demonstrating ambient isotopies performed on any representable knot. If desired, tricoloring can be performed during playback animation to demonstrate how a given knot's tricolor schema is affected by Reidemeister moves.

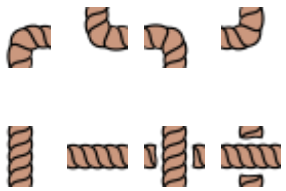
Knotswithstanding! allows a user to save partial or complete knots and animation sequences to their local hard drive, and also to re-load them for display or further editing.

Because Knotswithstanding! is a web-based "rich internet" application, no purchase, license, or installation is required. It can run anywhere a web browser and internet connection are present, eliminating considerations of compatibility and system requirements. However, because Knotswithstanding! utilizes features of HTML5 which are not yet supported by Internet Explorer, Mozilla Firefox and Google Chrome are recommended browsers, as both are known to support client side file downloads and HTML5's canvas element, which are both utilized.

It is hoped that Knotswithstanding! can be used in the classroom to demonstrate introductory knotting concepts by a small experimental user base, so that feedback can be generated which would influence further work on this project. Knot theory is currently taught as part of Geometric Topology, and some discrete math curriculums.

2. The Search for a Suitable User Interface

As previously mentioned, the idea for using this particular tile set to depict mathematical knots came from Dr. John Burke of the Rhode Island College Mathematics Department. In theory, any mathematical knot can be constructed by assembling the following eight tiles:



Essentially they are four rotations of a corner tile, and two rotations each of the line and crossing tiles.

Initially the idea was to have a "board" area with a grid layout, and three "stacks" of square tiles representing lines, corners, and crossings which could be dragged and dropped onto the grid, then rotated appropriately to form knot structures. Upon implementation, it became clear that a more efficient manner of producing long runs, such as a strand created by several contiguous line tiles, was needed. In addition to the time consuming process of dragging individual tiles onto the board, each tile could also potentially require rotation if, for instance, the lines in the tile stack were rotated vertically by default and the user wanted to plot a horizontal run.

An early solution to this problem was to implement a keyboard entry method. The user would position a cursor on the grid with a mouse click. This would initiate an "entry mode" in which the user moved the cursor around the grid using the keyboard's arrow keys to navigate north, east, south, or west while the "rope" of the knot was plotted behind it. The [BACKSPACE] key could be used to backtrack, so it was necessary that the key entry itself be an object which remembered its history of tiles plotted, and directions travelled into and out of each plotted cell. This interface was a huge improvement on the initial drag-and-drop method and had the additional fringe benefit of discouraging the user from deviating from linearity.

However, by the time keyboard entry was implemented, it had become clear that the approach was hampered by its own limitations. In addition to the physically cumbersome process of manipulating both the mouse and keyboard simultaneously, key entry mode placed the program logic in a state where certain mouse actions could interfere with the entry object's move history and stored position pointers. For example, a tile could be dragged in from one of the stacks and placed in the cell occupied by the cursor, where no tile previously existed but where one would be imminently placed, should the cursor navigate out of the cell. Similarly, the user could easily delete a recently plotted tile with the mouse while still in key entry mode, creating a discrepancy between the actual plotted tile configuration and the entry object's move history. It was obvious that the next step was to do all plotting with the mouse and make the entry object more dynamic.

The resulting mouse based interface remains the current input implementation. The user activates the cursor and initiates entry mode by hovering the mouse pointer over any unoccupied square on the grid and pressing down the left mouse button. When the left button is released, entry mode terminates and the cursor vanishes. While entry mode is active, the user guides the cursor with the mouse pointer, effectively "dragging" it. It can be dragged north, east, south, and west through any unoccupied square and across strands, creating crossings. Each time the user initiates entry mode, a new entry object is created. The user can "pick up on" dangling strand ends, causing the entry object to assimilate each pre-existent strand tile into its move history, such that the user can actually backtrack and erase contiguous tiles which weren't plotted in the current entry instance. When an entry instance is terminated, it is discarded and replaced with a new entry instance should the user decide to resume plotting either where they left off, at another dangling end, or in any other unoccupied grid cell.

Once mouse entry had been implemented and polished, and developmental focus shifted to flipbook style animation of knot "pages," it became conceivable that the user might want to create more intricate structures at a higher resolution. Functionality was added to toggle between the default 30 pixel cell size, to smaller 20 and 15 pixel cell sizes. For these smaller sizes, a plot "tolerance" was added to maintain the ease of plotting a straight line without deviating from the row or column of the cursor's trajectory. With the cell width set to 15 pixels, for example, a user would have to deviate

from their current trajectory 8 pixels or more into the neighboring row or column to implement a change of direction.

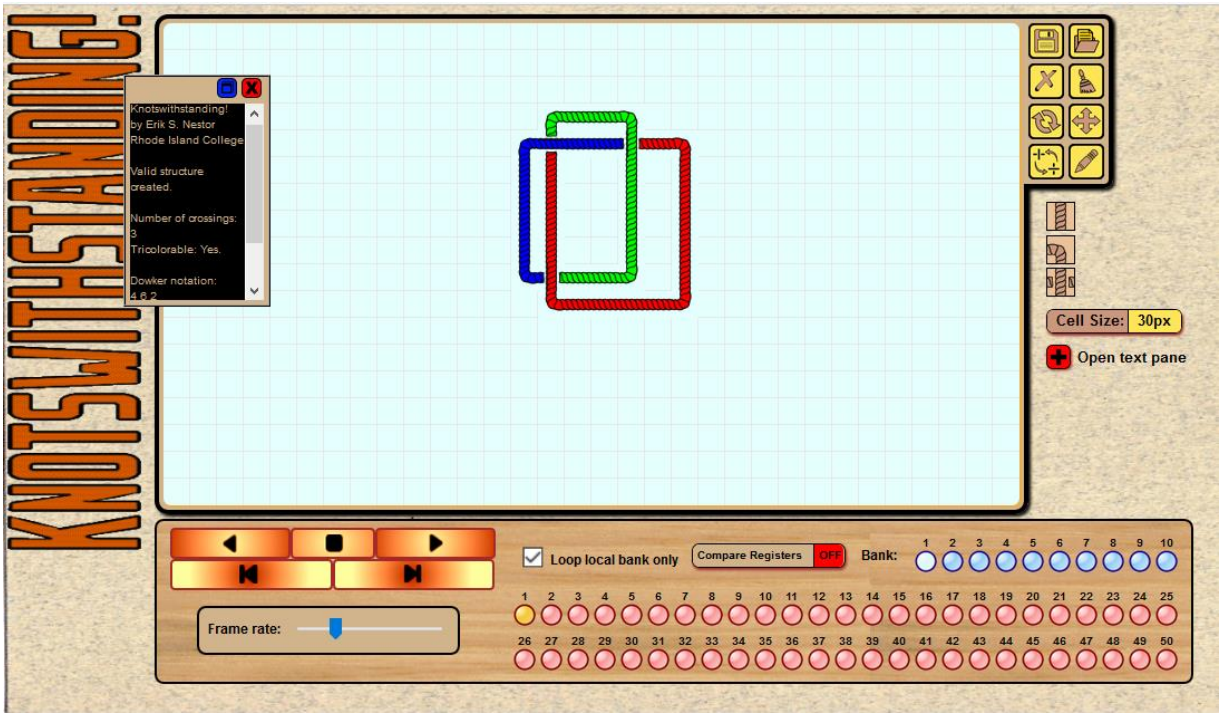


Figure 1 on the following page shows Board and its constituent classes. An instance of Board is created when the program is executed. Board creates and includes instances of RegisterSet, DisplayPane, and Analysis should a user-created structure be analyzed. The Board contains a discrete Cell object for every unoccupied cell on the board. However, only occupied Cells contain a Tile object.

Board creates a new instance of TurboBuild every time the user clicks the left mouse button while the cursor is positioned inside the grid. Each instance replaces the previous instance. TurboBuild manages the automatic generation and placement of tiles during mouse entry, in addition to maintaining a move history. It assimilates contiguous tiles into its move history should the user click next to an occupied cell, “picking up” on a loose end.

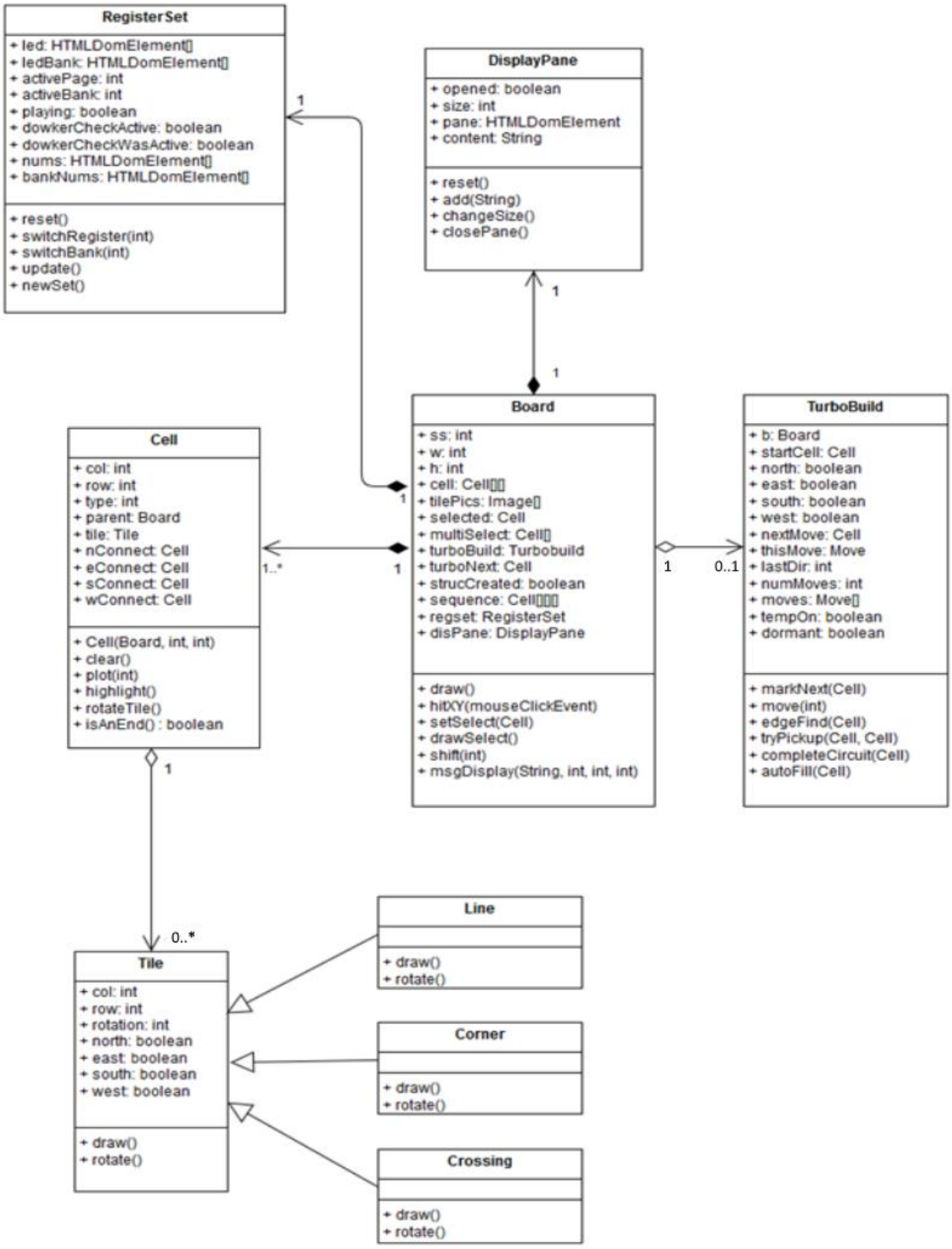


Figure 1 - The Board class and its components

[6]

3. Why Knots? – a *Very* Brief Synopsis

Humans have used knots since prehistoric times to secure and fasten objects. Their symbolic use in Irish and Tibetan religious art to express concepts of balance and interconnectedness dates back several centuries before Christ.

A mathematical knot is a theoretical embedding of a circle in three-dimensional Euclidian space, crossed in such a way that it can't be deformed into a planar circle without intersecting itself at some point during the deformation process. In theory, any mathematical knot can be morphed into an infinite number of unique homeomorphisms by undergoing distinct sequences of spatial deformations termed *ambient isotopies.* These non-traumatic bends and twists are often called *Reidemeister moves*, after German mathematician Kurt Reidemeister who first showed that transformation between any two knot homeomorphisms could be deconstructed into sequencing and repetition of three atomic 'moves'.

Knot theory, the formal study of mathematical knots, is a comparatively recent discipline. The first knot tables were compiled in the late nineteenth century by Scottish physicist Peter Guthrie Tait. Originally used to support Lord Kelvin's vortex theory of the atom, the tables fortunately outlived the ill-fated theory. In the early part of the twentieth century, as the field of topology emerged, knot theory became an intrinsic subdiscipline for the ability of mathematical knots to describe properties of topological surfaces.

Despite its relative youth, knot theory has ramifications in mathematics, biology, physics, and art. The past few decades have seen a tremendous swelling in its applications, and it is increasingly valued for its ability to model knotting phenomena in DNA, as well as its correlations to mathematical methods in quantum field theory. As a result, it is currently working its way into an increasing number of undergraduate curriculums. Increased interest in knot theory has been at least partially catalyzed by computational technology, and development in the field has been rapid since the early 1970s. Despite this, many of the discipline's principal problems remain open.

Regarding single, non-linked, nontrivial knots, there are two principal problems. One involves recognizing if a given homeomorphic *projection* of a knot is in fact the "unknot," meaning a sequence of deformations exists capable of mutating the projection into a planar circle. The other involves recognizing if two knot projections are topologically equivalent to each other. A formal algorithm for the unknotting problem was published in 1961 by German-born algorithmic topologist Wolfgang Haken. This and other unknotting algorithms were shown to be in the NP complexity class in *The Computational Complexity of Knot and Link Problems* (Hass, Lagarias, Pippenger 1999), and in the same paper exponential worst-case runtime bounds were given for the number of moves required to reduce the unknot to a circle. Software implementations of unknotting algorithms have so far been problematic, as the growth exponent is sufficient to rapidly overwhelm most CPUs as the number of crossings increases.

Mathematical knots can be depicted in several ways, but Knotswithstanding! implements what is usually called a *square depiction*, in which a knot is represented exclusively with straight horizontal lines, vertical lines, and corners. Segments and crossings of a knot's square depiction have a one-to-one correspondence with those of a knot's standard depiction. Figures (2a) and (2b) below show both standard and square depictions of the same figure-eight knot, with their corresponding crossings highlighted:

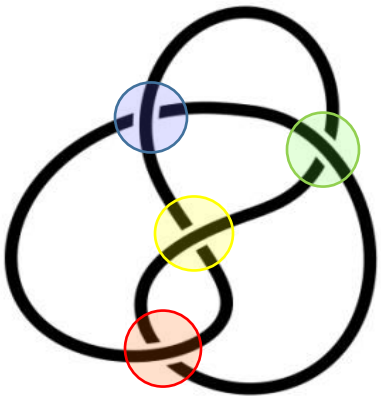


Figure 2a

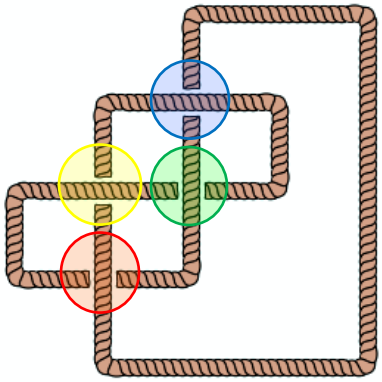


Figure 2b

Because any unique mathematical knot can be represented by an infinite number of distinct projections, invariants are used to prove knot equivalence. An invariant is a property which remains unchanged after certain types of transformations (in knot theory, these would be ambient isotopies, such as the Reidmeister moves described earlier). It can be stated that, if two knot projections are equivalent, any such invariant is present (or absent) in both projections.

Tricolorability is a knot invariant which dictates that exactly three colors can be used to color a knot, and it can be colored in conformance with the following requirements:

1) Each segment (an arc spanning from one undercrossing to the next) is uniquely colored.
2) At each crossing the three segments represented are either
    a. Uniformly colored

        -or-

b. Distinctly colored.

In other words, the number of allowable distinct colors at a given crossing is either 1 or 3. So in the instance that three colors are arbitrarily selected and enumerated 0, 1, and 2, valid tricolorable crossings can exhibit one of the following schemes:

0-1-2
0-0-0
1-1-1
2-2-2

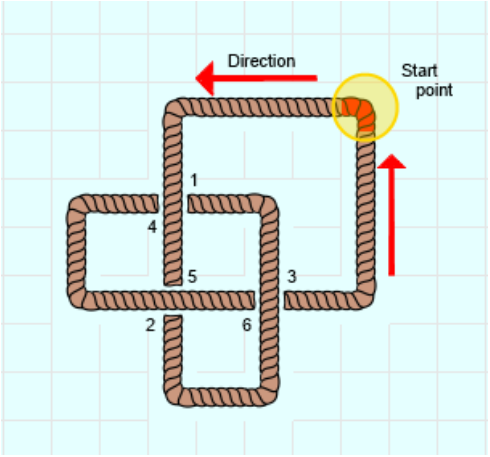
Of course since all three colors *must* be used in any valid tricolor scheme, every tricolorable knot must contain at least one crossing with enumerated scheme 0-1-2, described above.

Tricolorability is important in knot theory because the unknot, having no crossings, is not tricolorable. Therefore, if a knot projection exhibits tricolorability, it is non-trivial and cannot be deformed into the unknot. This, at least, provides a way in

which *some* knots can be proved non-trivial through computation, without directly implementing an unknotting algorithm. And of course, if a given knot projection is tricolorable, all of its homeomorphic projections are as well.

Canadian topologist Clifford Hugh Dowker's notation for knots has been widely adopted by knot theorists, topologists, and students alike. The *Dowker notation* is a sequence generated by traversing the knot in either direction from any chosen point. As the knot is traversed, each of its ***k*** crossings is enumerated with the numbers 1, …, 2***k*** in the order of traversal. When traversal is complete, each crossing will have been visited twice and will be represented by a distinct number pair consisting of both an odd and an even number. In the event that the crossing's even number represents the overcrossing strand, then that even number is represented as a negative. Once all ***k*** crossings are enumerated, the Dowker sequence is the sequence of even numbers paired with odd numbers 1, …, 2***k***-1 in that order.

Here is a trefoil knot with its crossings labeled post-traversal:



As can be seen, each crossing can now be represented by a number pair which indicates the places in which it was encountered during traversal, in this case, (1,4), (5,2), and (3,6). Taking the even numbers which are paired with our ascending odd number sequence (1,3,5) yields (4,6,2) as our Dowker sequence for this trefoil knot projection.

Incidentally, because the “top” numbers in this sequence of pairs are consistently odd and the “bottom” are consistently even (or vice versa), it can be inferred that the trefoil knot is an *alternating* knot, one which alternates between over and undercrossings during ordinary traversal.

4. Analysis of a Valid Structure

Upon building a knot projection, the user may allow the program to traverse the structure, counting the crossings, numbering the segments, and checking for tricolorability. This process can be initiated by the user clicking the 'Tricolor' icon in the editing toolkit (see **A3.8** in User Documentation)

to the right of the Knotswithstanding! grid.

Prior to traversal, the grid is checked for dangling ends which indicate the depicted structure is not circuitous. Once the structure passes that test, the board is scanned at a somewhat lower level. Connections are made between cells as each occupied cell is made aware of the number, locations, and types of its immediate neighbors.

When these rudimentary connections are made, the knot can be traversed. Knotswithstanding! then scans the board, from top to bottom and left to right. The first tile it encounters on any circuitous structure will always be an "upper left" corner:

Assuming this goes as expected, it backtracks in a counter-clockwise direction until it finds the beginning of the current segment, with a segment being defined as an arc between two undercrossings. From here, it switches directions and traverses the entire knot in a clockwise direction, enumerating segments and crossings, and creating superstructures to represent those segments and crossings. These superstructures are data objects which "know" more than the individual cells do. For example, a Segment superobject knows which undercrossings it is bound by, how many and which type of tiles form its interior, its segment number, and ultimately, if colorable, its color. A SuperCrossing knows which segments meet at that location and contains a reference to

each. It also knows which undercrossing it will eventually encounter if it follows any of its segments in any direction. In addition, it knows which actual cell it encapsulates, its position, row and column, and also keeps track of the top and bottom Dowker numbers. When all segments are enumerated and traversed, the structure is then checked for tricolorability, depending on the program's state at the time the Analysis was requested.

Figure 3, on the following page, is a class diagram for Analysis and its constituent classes. When an instance of Analysis is created, its constructor method calls a series of its other methods, specifically isValid(), findFirstCorner(), findCrossings(), connectCells(), and traverse(). Traverse() may or may not call isTricolorable(), again depending on the context from which the Analysis was invoked.

A single instance of the Board may have several instances of Analysis. An instance of Analysis is considered "disposable" and generally any requested instance replaces the previous.

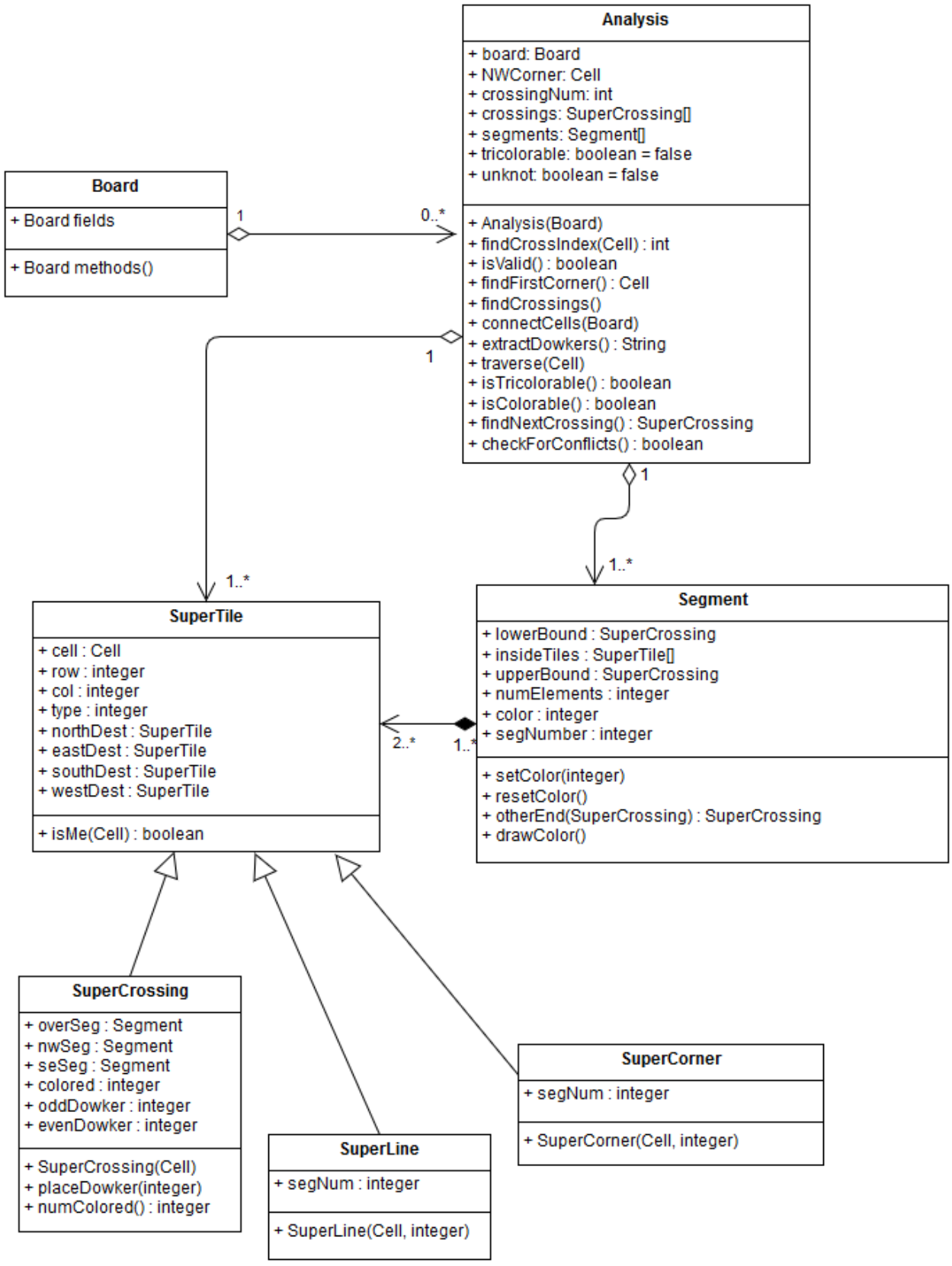
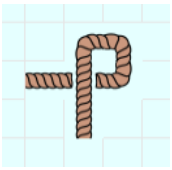


Figure 3 - Analysis Classes

5. Tricolorability Detection

Detection of tricolorability in Knotswithstanding! is currently executed through an exhaustive search in which every possible schema is tried and tested until either a working schema is found, or an insurmountable conflict is encountered. Coloring attempts begin at an arbitrarily selected crossing in which all three joined segments are unique. This is based on the presumptions that (1) if a knot is tricolorable, there must exist at least one crossing at which all three colors meet and (2) at that crossing, any one of the 6 possible tricolor combinations can be used, each with equal success. Since the knot segments are enumerated during initial traversal, it's easy to iterate through each crossing, checking to verify that each segment meeting at that crossing is uniquely enumerated. An example in which the segments of a crossing are not uniquely enumerated would be, for instance, a crossing formed by a Type I Reidemeister twist like the following:



A crossing which satisfies this prerequisite of segment uniqueness is then tricolored arbitrarily and attempts are made to color the entire knot, beginning at that crossing. If a tricolor schema for the knot is not found from that crossing, it can't necessarily be inferred that the knot is not tricolorable because the possibility still exists that three uniquely enumerated segments with the same color value meet at that crossing. If this is the case and the knot is tricolorable, then, just as any of the six possible 3-color configurations were sufficient to find a schema from the starting crossing, likewise either of the three colors of the tricolor scheme can be used with equal success to color all segments of the start crossing uniformly, and seek a schema based upon that. If the knot is tricolorable and we are starting our search at a crossing in which all three segments are the same color, we will have to make continuous prioritized attempts to introduce the remaining two colors into the schema at every crossing, so that all colors are utilized in the schema at some point along the way. The successful implementation of

the remaining two colors is, of course, checked at the conclusion of such searches as a prerequisite of tricolorability.

Attempts to tricolor a knot using any one crossing as a start point are executed through recursive tests on two continually redefined sets of partially-colored crossings. The most exclusive, prioritized set in which two segments are colored and one is not, and a less-exclusive set in which only one segment is colored. The crossings with two segments already colored are, of course, prioritized because there is only one possible color option to fill the remaining segment and if that one color doesn't work, a backtrack is in order.

After each individual segment is colored, the knot is checked for conflicts. If a conflict exists, the last color is reset and the recursion backtracks, returning false. If no conflict exist, the remaining partial crossings are repolled, recategorized, and rechecked. When no partially-colored crossings exist in either set, a tricolor schema has been found. The flowchart in Figure 5 on the following page shows the logic for tricolor attempts from any one given crossing.

An exhaustive search for a working tricolor scheme as described above has an exponential time complexity (EXP, approximately $O(4^n)$ where 'n' is the number of depicted crossings). It is very possible that a more efficient algorithm exists, however the exhaustive search fit well with the scale of this project and, though it's remotely possible to overwhelm most modern CPUs with knots containing 100 crossings or more, conditions have to be fairly precise in order for the CPU to "choke on it." In most common situations, if a tricolor solution exists for a given knot, once the first several segments are colored, the rest tend to "fall in line," as existing segment colors inherently eliminate possibilities as to what the colors can be at their connected partially-colored crossings. Likewise, if no tricolor solution exists, an insurmountable conflict tends to surface fairly quickly. The algorithm only really encounters worst-case behavior in situations where a valid knot has well over 100 crossings, is non-tricolorable, and the relevant conflict occurs on the last crossings checked.

Figure 5 shows a flowchart describing the logic.

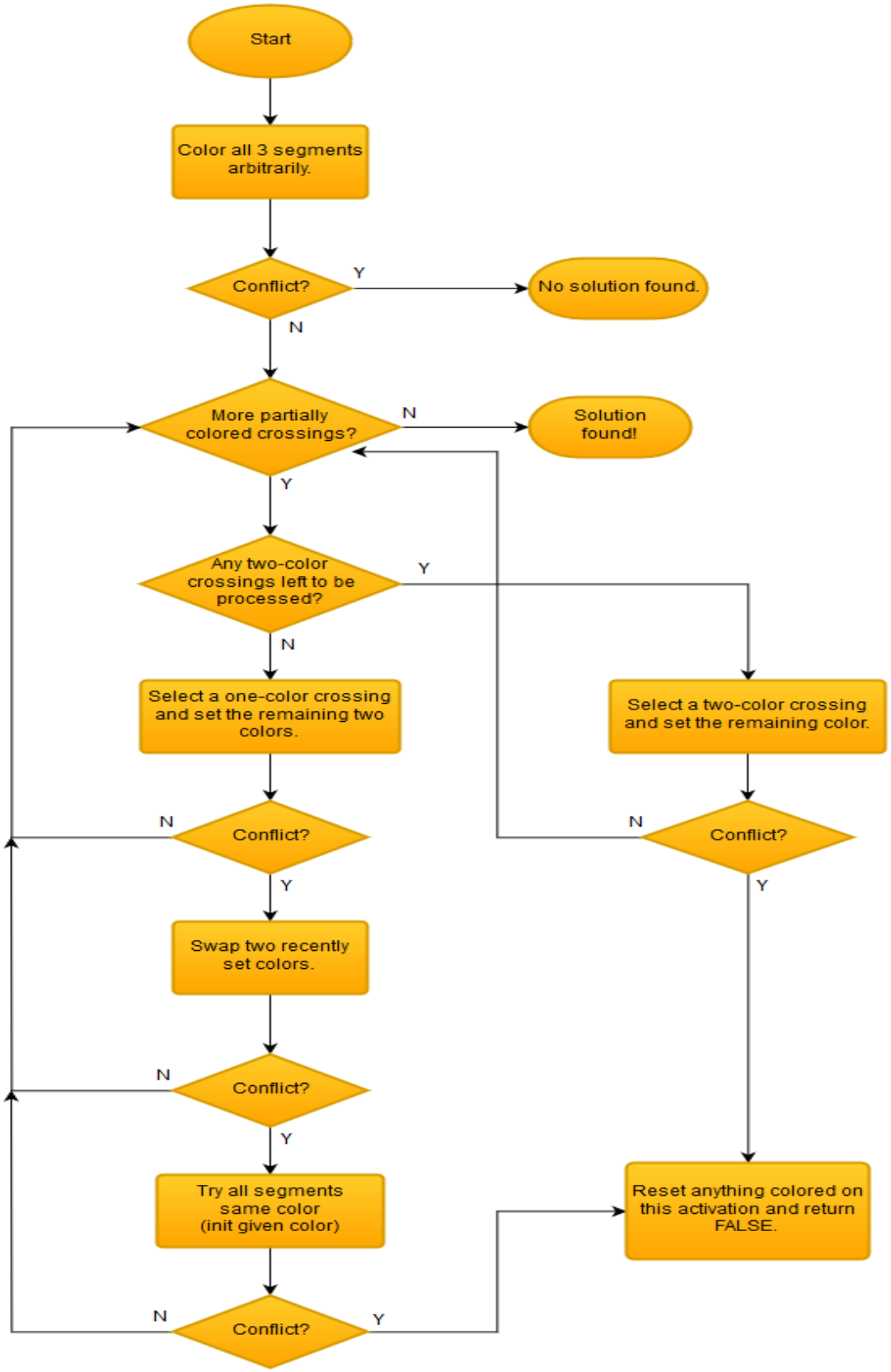


Figure 4 – Tricolor Logic

6. Using Banks and Registers to Animate a Knot

Since a great deal of knot theory is concerned with spatial deformations (the ambient isotopies described in Section 3), it seemed necessary that any sort of pedagogical tool used to demonstrate introductory concepts would have some way to represent these deformations, and also demonstrate the preservation of tricolorability through them.

The easiest way to implement this seemed to be a system in which several separate grids could be stored in memory banks, and configured by the user so that they could be played back, demonstrating whichever deformation the user wanted to portray. By carefully editing each frame, users would have ultimate control over the way in which each successive deformation would take place. Besides the obvious limits of the tile set, the transformations which could be displayed in this manner would only be confined by the limits of the user's imagination.

Toward these ends, a panel was added to the GUI displaying fifty registers which could be dynamically displayed and switched between. A user could simultaneously edit a series of progressive morphisms, then "play" them back in a style akin to flipbook animation. Once this was implemented, a need for more registers to display more advanced animations became apparent. In order to preserve screen real estate, this was done in the form of adding ten "banks," which would each contain a separate and discrete group of fifty registers. Thus the number of registers was increased to 500.

Functionality was added so that these registers could be either iterated through manually, or "played back" in either direction, forward or backward. A frame rate slider was added to manipulate the speed of playback, and a user-controlled switch which restricted playback to the current active bank of 50 registers, so that the user could choose between making a series of unrelated animations and switching manually between them, or stringing the banks together to create one longer, more sophisticated animation. Functionality was then added to tricolor the plotted structures during playback (if possible), so that preservation of tricolorability through ambient isotopies could be demonstrated.

And, only because just about everything needed was already programmed and calculated for other functionalities, the ability to compare two registers in the same bank for Dowker equivalence was also added. This will test true if the respective Dowker sequences for the structures drawn in those registers are identical.

7. What the Future Holds

In Knotswithstanding!'s current state, its most urgent need is for a more efficient tricoloring algorithm.

Also, it is both possible and important to take steps toward unknotting. Certain Reidmeister moves can be indicated by the Dowker sequences which are generated already. For example, any crossing for which the absolute values of the top and bottom Dowker numbers are consecutive integers is a Type I Reidemeister twist. Also, if two crossings are found for which the absolute values of their respective top Dowker numbers are consecutive integers, and the absolute values of their respective bottom Dowker numbers are as well, those two crossings form the periphery of a Type II Reidemeister looping. Unfortunately, not only are these methods insufficient for finding all possible loops and twists, but the task of developing a generalized method for undoing them visually, given the infinitude of ways these twists and loops can be implemented on the grid, is much more daunting.

On a possibly related note, it would be interesting to do more research on newer homologies for discerning knots which are built upon the Alexander and Jones polynomials, and possibly find a way to implement them.

A possibly more manageable short term goal might be to re-implement the canvas container. Currently a bounded grid, implementing it as a scrolling window would allowing for building much larger, theoretically limitless structures. The shift functionality could then be changed to manipulate the actual viewing pane, providing access to the entire construct. This would likely require the grid's representative data structure to be switched to a two-dimensional linked list, rather than a two-dimensional array, as it is now.

Appendix A – User Documentation

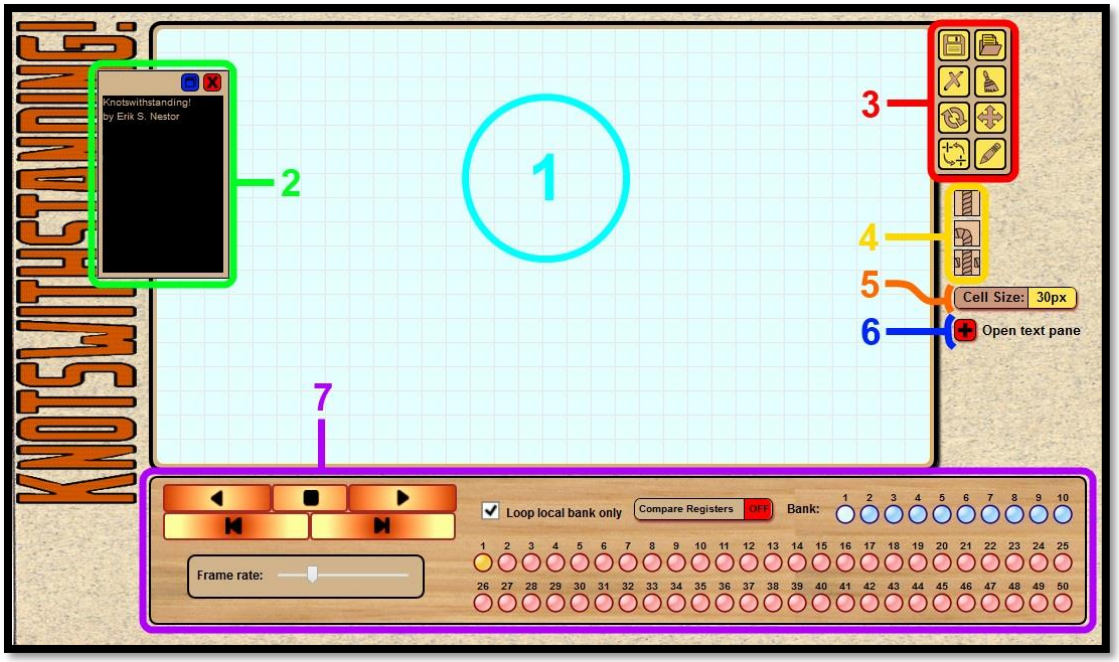


Figure 5 - The *Knotswithstanding!* user interface

1. Grid
2. Text pane
3. Editing toolkit
4. Fragment tiles
5. Cell size toggle
6. Text pane open
7. Storage panel

A1 The Grid

The main grid appears at the center of the Knotswithstanding! user interface. This is the user's primary workspace, and knots can be plotted here using mouse entry, or simply by dragging single tile fragments from one of the three "stacks" to the right of the grid (see **A4**).

When the left mouse button is pressed anywhere on the grid, a cursor with up to four yellow directional arrows appears in the square corresponding with the mouse position. The cursor arrows indicate directions in which the user can legally move. By dragging this cursor along the canvas, a strand of "rope" is generated. The user can then pick up on either end of the rope and drag to elongate it, running the strand along the grid in any indicated legal direction. The user is encouraged to follow a linear path and merging with a corner, or any other move which would create ambiguity, are discouraged during mouse entry. It is possible to construct such ambiguous structures by manually dragging single tiles, but at this point, there is no functionality in place to parse or analyze such a structure.

During the process of mouse entry, the most-recently plotted tile is automatically highlighted, making it easy for the user to rotate or edit the tile after it has been plotted. The appearance of a red cursor arrow indicates it is possible to connect two strands by a single move in that direction. Moving in that direction will prompt Knotswithstanding! to place the proper piece and suspend mouse entry. Should a circuit be completed using this or any other method, the structure is automatically analyzed and the user is given a message in the text pane including the number of crossings, a Dowker sequence identifying the knot, and an indication of the knot's tricolorability.

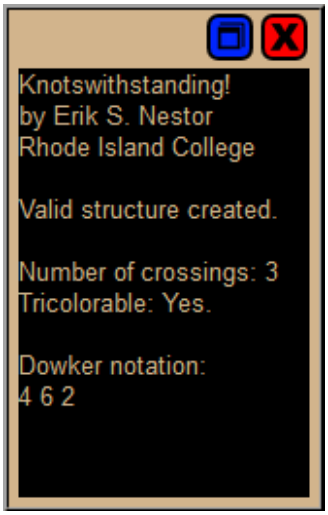
Crossing a strand over itself creates a crossing. The default behavior is for the second strand to cross over the first strand, but it's easy for the user to change overcrossings to undercrossings (and vice versa) by rotating the selected tile with the [SPACE BAR] immediately after plotting, or it can be done later by manually selecting the tile on which the crossing occurs, and using the rotate tool in the editing toolkit, located to the right of the grid (Fig. 8, #3).

Releasing the left mouse button suspends mouse entry, but it can easily be resumed by clicking in any unoccupied square on the grid. The user can “pick up” on the end of any existing strand and continue it in any legal direction.

Pressing the right mouse key in the grid area changes the tooltip to an eraser tip, which will erase any tile it passes over. The user can also erase recently plotted tiles while still in mouse entry by backtracking, which has the effect of undoing recent moves in reverse chronological order. This is done using the [BACKSPACE] key while still in mouse entry with the right button pressed.

Tiles in Knotswithstanding! represent the fragments necessary to build a valid knot projection. The corner, the line, and the crossing are used, with the corner having four distinct rotations, and the line and crossing each having two. The user assembles knot projections by placing the tiles on the board in various configurations, either by mouse or keyboard entry. Placement of tiles is allowed unless the user has entered either shift or multi-select modes. When placement is allowed, the background of the grid will appear light green. When the board is in either shift mode or multi-select mode, the background of the grid will appear light red, indicating that tile placement is not allowed.

A2 The Text Pane



Knotswithstanding! communicates with the user via text alerts displayed in a text window (Fig. 8, #2). The window can be resized, relocated, hidden, and re-displayed. The window is resized using the blue restore button  to toggle between three predefined sizes, and it can be dragged and relocated to any position on screen which suits the user. It can be hidden by clicking the red 'X' tab  , and restored to its last position by clicking the red '+' tab  on the right hand side of the grid (Fig 8, #6).

A3 Editing Toolkit

The editing tool tray appears as a tab located on the right hand margin of the Knotswithstanding! grid. Here's a quick rundown of the tools and what they do:

A3.1 Save Structure

Allows the entire structure to be saved on the client computer's local storage drive. The user will be prompted for a file name and the saved file will include drawings from all occupied banks and frames.

A3.2 Load Structure

Allows the user to load a previously saved structure, with all its accompanying banks and frames. All existing plotted structures in all banks and frames are deleted when the user loads a new file. When the new file is loaded, the first occupied register of the first occupied bank is displayed.

A3.3 Delete Selected Cells

Deletes all selected cells on the current displayed frame. Cells can also be deleted by clicking the right mouse button, which converts the mouse cursor to an erase tool which can be dragged over any region the user wishes to erase.

A3.4 Clear Grid

Deletes all cells on the current displayed frame.

A3.5 Rotate Selected Cells

Rotates all selected cells on the current displayed frame. Pressing the [SPACE BAR] is a hotkey for this action and is functionally equivalent.

A3.6 Shift Selected Cells

Enter shift mode, in which specified cells can be shifted on the grid in one of four possible directions. Clicking on the ‘shift’ icon will cause four large arrows to appear over the grid. Clicking on any of these arrows will shift the entire construct in the indicated direction. Movement in any one direction is limited by the boundaries of the board. If only one cell is selected, or if no cells are, the entire structure will be shifted. If there is a batch, or block selection of more than one cell, only the selected cells will be shifted.

A3.7 Invert Crossings

Inverts (rotates) each plotted crossing in the current frame.

A3.8 Analyze and Color

Analyzes the current structure and displays its status, Dowker notation, and colorability to the display panel provided no animation is playing. The knot is colored as well, in or out of playback mode. During playback, this button is used to toggle tricoloring on or off.

A4 Knot Fragment Tiles

Tile “stacks” representing lines, corners, and crossings appear to the right of the Knotswithstanding! grid. Tiles can be manually dragged onto the grid and placed in any occupied or unoccupied cell. If placed in an occupied cell, the tile will replace whichever tile previously occupied that location. Once placed, the tiles can be rotated appropriately according to necessity.

A5 Cell Size Selector

The cell size selector toggles between three different cell sizes, for board "resolution." The default size for board cells is 30 pixels, but clicking on the yellow half of the tab makes 20 and 15 pixel sizes both available to the user for more detailed or sophisticated drawings. Toggling the cell size will cause all populated banks and registers to be erased to maintain consistency between consecutive registers which might eventually be used for animation playback. The user will be warned and given the chance to cancel the action so that they might save their work if they have not done so already.

A6 Open Text Pane

Located to the right of the Knotswithstanding! grid, the **Open Text Pane** button displays the text pane if it has been previously hidden by the user.

A7 Storage Panel

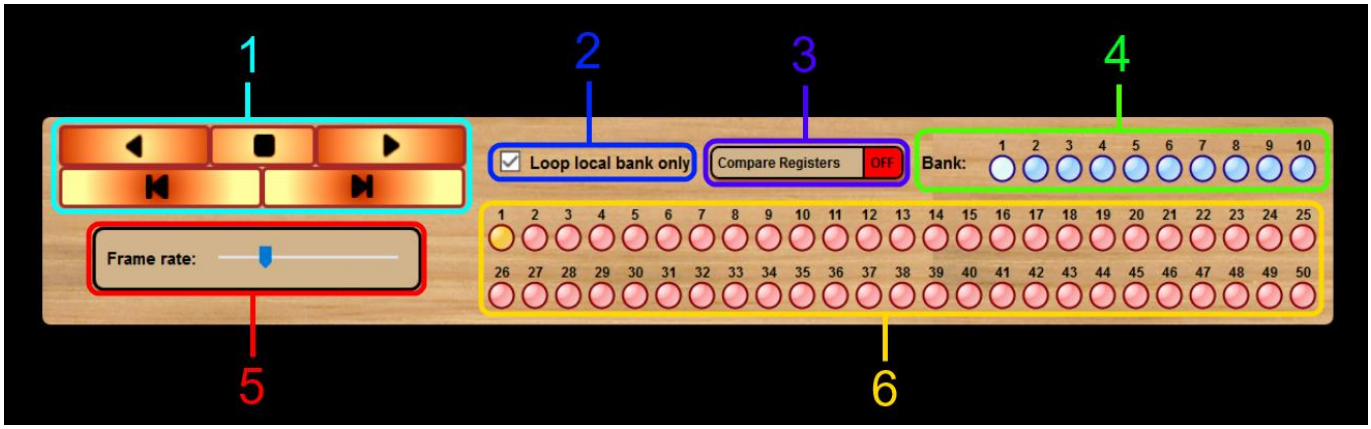
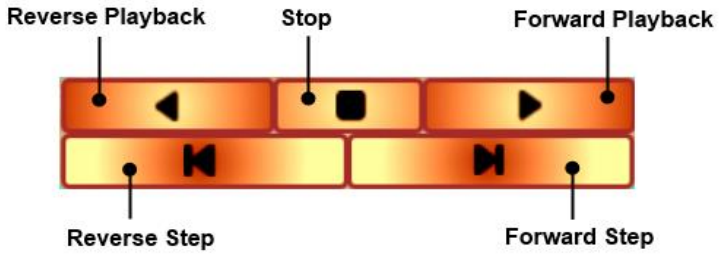


Figure 6 – The *Knotswithstanding!* storage panel

1. Playback controls
2. Loop local bank checkbox
3. Register compare toggle
4. Bank display and select
5. Frame rate slider
6. Register display and select

A7.1 Playback Control Panel



The **Forward Playback** button begins an animation loop. All populated banks and registers will be displayed unless the **Loop local bank only** checkbox is checked, in which case only registers belonging to the currently active bank will be displayed. Playback skips all unpopulated banks or registers. Playback loops indefinitely until **Stop** is clicked.

The **Reverse Playback** button begins an animation loop in reverse. Playback begins with the highest populated register in the highest populated bank and iterates backward through all populated banks and registers. Apart from display sequence, all playback behavior is the same as for **Forward Playback**. Playback loops indefinitely until **Stop** is clicked.

The **Stop** button halts any active playback.

The **Forward Step** button displays the contents of the next populated register if one exists, and **Reverse Step** displays the contents of the previous populated register if one exists.

A7.2 Loop Local Bank Checkbox

When checked, playback is restricted to the registers contained in the currently active bank. This option is checked by default.

A7.3 Register Compare Toggle

Compares the Dowker sequences of two plotted structures. It can only be activated when there is more than one populated register. Clicking the ON/OFF tab toggles this feature on, at which point all populated registers will begin to blink. The user can either select two distinct registers for comparison, or toggle back out of this mode by once more clicking the ON/OFF tab.

A7.4 Bank Display and Select

Displays statuses of the ten enumerated banks.

- indicates the bank is empty and not currently active.
- indicates the bank is empty and currently active.
- indicates the bank is populated and not currently active.
- indicates the bank is populated and currently active.

The user can switch banks at any time by clicking upon the desired bank's "LED."

A7.5 Frame Rate Slider

Adjusts the rate of playback.

A7.6 Register Display and Select

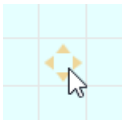
Displays the statuses of the 50 registers contained in the currently active bank.

 indicates the register is empty and contents are not currently displayed.
 indicates the register is empty and contents are currently displayed.
 indicates the register is populated and contents are not currently displayed.
 indicates the register is populated and contents are currently displayed.

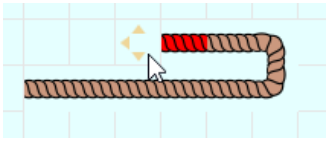
The user can switch registers at any time by clicking upon the desired register's "LED." If the user switches to an unpopulated register, whatever was plotted in the previous register remains displayed. This is for ease of making small changes to progressive frames of an animation. If the user switches to an already populated register, display switches to the contents of that register.

A8 Plotting a Structure

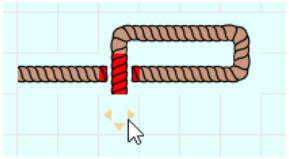
The building of knots in Knotswithstanding! is fairly easy. Rope is laid down by clicking and dragging across any horizontal or vertical region of the board. When the left mouse button is pressed down with the mouse pointer hovered over any unoccupied board cell, a cursor appears.



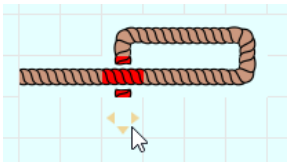
The four yellow arrows which make up the cursor indicate directions in which the user can move legally. The cursor will follow the mouse pointer as it moves along any row or column on the board, laying rope behind it.



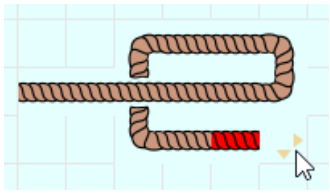
The most recent tile to be plotted is automatically selected and appears red. The user is allowed to cross over previously plotted strands, creating overcrossings.



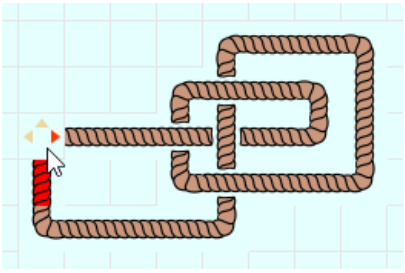
And the [SPACE BAR] is available as a hotkey for rotation, so that overcrossings can be transformed into undercrossings while plotting.



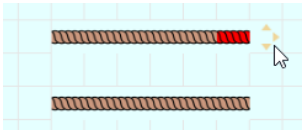
Note that merging with a previously plotted corner is not allowed, as it breaks linearity.

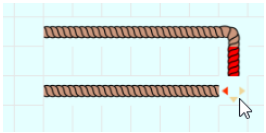


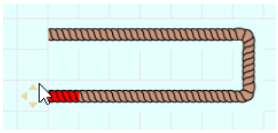
In the event that a plot can be connected to a dangling end, the cursor arrow indicating the appropriate direction will appear red, and a move in that direction will automatically plot the missing piece and cause the cursor to vanish. If a terminating move of this sort forms a circuitous structure, it is automatically evaluated and its status is displayed in the Knotswithstanding! text pane.

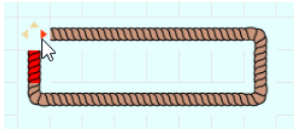


Note that fragments can be plotted separately, then connected by “picking up on” a dangling end of one fragment and connecting it to that of another.









At any time during an active plot, while the right mouse button is pressed, it is possible to backtrack by pressing the [BACKSPACE] key. Backtracking erases the most recently plotted tiles and moves the cursor backward along the rope.

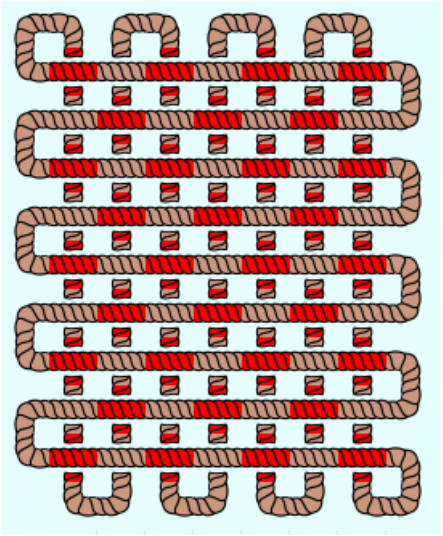
A8.1 Manual Tile Entry

Individual tiles representing knot fragments can be dragged onto the grid from any of the three tile "stacks" (see **A4**) to the right of the Knotswithstanding! grid. These can be placed anywhere. Once on the grid they can be rotated, deleted, and selected in the same manner as the other tiles.

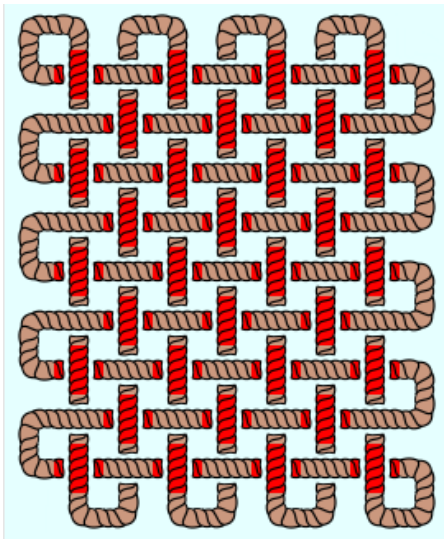
A8.2 Batch Select

During the build, the last tile plotted is automatically selected to facilitate easy rotation and deletion by the user.

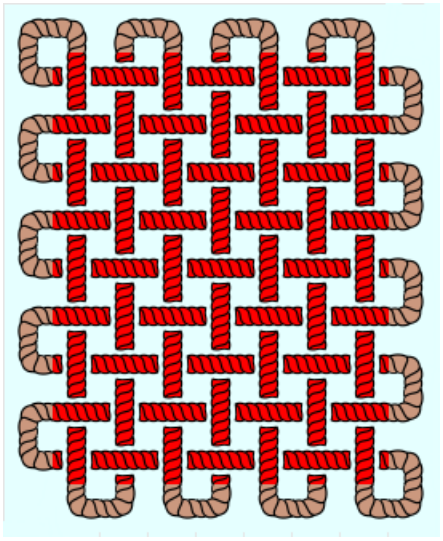
The user can also manually select one or more tiles for rotation or deletion. Single tiles can be selected at any time during knot build by pressing the left mouse button while the mouse cursor is above the target tile. Holding down the [CTRL] key while selecting tiles with the left mouse button allows the user to select multiple tiles.



Selected tiles are eligible for batch processing, such as deletion or rotation.



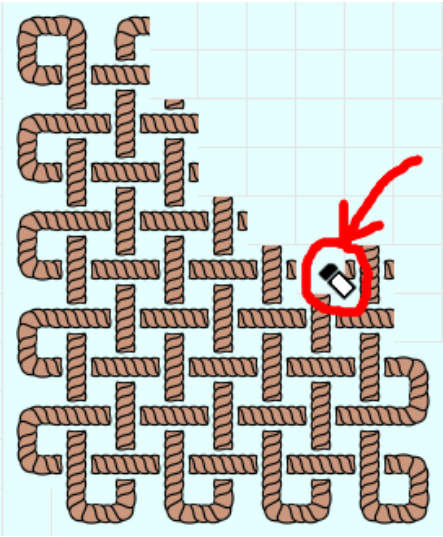
Tiles can also be “block selected” for processing by first selecting one corner, then selecting the opposite corner while holding down the [SHIFT] key.



A8.3 Rotating and Deleting Tiles

Individual tiles can be rotated by selecting them with the left mouse button, then clicking the **Rotate Selected** button in the editing toolkit (see **A3.5**). Similarly, multiple tiles can be batch selected and the **Rotate Selected** button used for simultaneous rotation of multiple tiles. The [SPACE BAR] is a hotkey for tile rotation and can be substituted for the **Rotate Selected** button in any situation that requires it. The [SPACE BAR] can also be used to rotate individual tiles during while the structure is being plotted, in which case only the most recently plotted tile will be affected since it is automatically selected.

Individual tiles can be deleted from the board by right clicking with the mouse cursor positioned over them. In addition, holding down the right mouse button transforms the mouse cursor into an eraser tip which will erase any tiles it is dragged across.



In addition, individual and multiple tiles can be deleted in a similar manner as the rotation process detailed above- by single or batch-selecting one or multiple tiles, then clicking the **Delete Selected** button in the editing toolkit (see **A3.3**).

Clicking on the **Clear Grid** button in the editing toolkit (see **A3.4**) deletes every tile on the grid, selected or not, effectively clearing the board.

Appendix B – The Code

B1 index.html

```
<!DOCTYPE HTML>
<html>

<head>
    <meta charset="UTF-8"/>
    <html lang="en">
    <title>Knotswithstanding!</title>
    <link rel="stylesheet" href="css/styleSheet.css"><!--css/-->
    <script type="text/javascript" src="js/jquery-1.11.1.js"></script>
    <script type="text/javascript" src="js/jquery-ui-1.10.4.custom.js"></script>
    <script type="text/javascript" src="js/AnalysisAndSuperClasses.js"></script>
    <script type="text/javascript" src="js/jcanvas.js"></script>
    <script type="text/javascript" src="js/EventHandling.js"></script>
    <script type="text/javascript" src="js/RegisterSet.js"></script>
    <script type="text/javascript" src="js/DisplayPane.js"></script>
</head>

<body id="bckgrnd" height="600">

    <!--Shift arrows-->
    <div id="shiftButtons">
        <img id="upShift" src="img/trans.png"></img>
        <img id="downShift" src="img/trans.png"></img>
        <img id="rightShift" src="img/trans.png"></img>
        <img id="leftShift" src="img/trans.png"></img>
    </div>

    <!--Tile menu-->
    <div id="tileMenu">
        <img class="draggableTile" id="tileLineVert" border="1" src="img/tileLineVert.png" style="top: 10px; left:
5px;"></img>
        <img class="draggableTile" id="tileCornerTopRight" border="1" src="img/tileCornerTopRight.png"
style="top: 55px; left: 5px;"></img>
        <img class="draggableTile" id="tileCrossOver" border="1" src="img/tileCrossOver.png" style="top: 100px;
left: 5px;"></img>
    </div>

    <div id="text-pane">
        <div id="closePane" class='opened' title='Hide'></div>
```

```
    <div id="sizePane" title = 'Resize'></div>
    <div id="text"></div>
</div>

<div id="resTog">
    <div id="resTogLabel">
        <p class="text">Cell Size:</p>
    </div>
    <div id="resTogBut">
        <p id="txt"  class="text" >30px</p>
    </div>
</div>


<div id='openPane'></div>
<p id="openPaneLabel">Open text pane</p>

<div id="layoutTiles">
    <img id="tileCornerBotRight" src="img/tileCornerBotRight.png"></img>
    <img id="tileCornerBotLeft" src="img/tileCornerBotLeft.png"></img>
    <img id="tileCornerTopLeft" src="img/tileCornerTopLeft.png"></img>
    <img id="tileLineHoz" src="img/tileLineHoz.png"></img>
    <img id="tileCrossUnder" src="img/tileCrossUnder.png"></img>

    <img id="tileCornerTopRightRed" src="img/tileCornerTopRightRed.png"></img>
    <img id="tileCornerBotRightRed" src="img/tileCornerBotRightRed.png"></img>
    <img id="tileCornerBotLeftRed" src="img/tileCornerBotLeftRed.png"></img>
    <img id="tileCornerTopLeftRed" src="img/tileCornerTopLeftRed.png"></img>
    <img id="tileLineHozRed" src="img/tileLineHozRed.png"></img>
    <img id="tileLineVertRed" src="img/tileLineVertRed.png"></img>
    <img id="tileCrossOverRed" src="img/tileCrossOverRed.png"></img>
    <img id="tileCrossUnderRed" src="img/tileCrossUnderRed.png"></img>

    <img id="tileCornerTopRightGreen" src="img/tileCornerTopRightGreen.png"></img>
    <img id="tileCornerBotRightGreen" src="img/tileCornerBotRightGreen.png"></img>
    <img id="tileCornerBotLeftGreen" src="img/tileCornerBotLeftGreen.png"></img>
    <img id="tileCornerTopLeftGreen" src="img/tileCornerTopLeftGreen.png"></img>
    <img id="tileLineHozGreen" src="img/tileLineHozGreen.png"></img>
    <img id="tileLineVertGreen" src="img/tileLineVertGreen.png"></img>
    <img id="tileCrossOverGreen" src="img/tileCrossOverGreen.png"></img>
    <img id="tileCrossUnderGreen" src="img/tileCrossUnderGreen.png"></img>

    <img id="tileCornerTopRightBlue" src="img/tileCornerTopRightBlue.png"></img>
```

```
        <img id="tileCornerBotRightBlue" src="img/tileCornerBotRightBlue.png"></img>
        <img id="tileCornerBotLeftBlue" src="img/tileCornerBotLeftBlue.png"></img>
        <img id="tileCornerTopLeftBlue" src="img/tileCornerTopLeftBlue.png"></img>
        <img id="tileLineHozBlue" src="img/tileLineHozBlue.png"></img>
        <img id="tileLineVertBlue" src="img/tileLineVertBlue.png"></img>
        <img id="tileCrossOverBlue" src="img/tileCrossOverBlue.png"></img>
        <img id="tileCrossUnderBlue" src="img/tileCrossUnderBlue.png"></img>
    </div>

        <div id="canvasDIV">
            <canvas id="canvas" height="540" width="960">Your current browser does not support Canvas! Use
Firefox or Google Chrome.</canvas>
        </div>

        <div id='buttonTray'>
            <input type='button' class='iconBut' id='clearBut' title='Clear Grid'>
            <input type='button' class='iconBut' id='invBut' title='Invert Crossings'>
            <input type='button' class='iconBut' id='eraseBut' title='Erase Selected'>
            <input type='button' class='iconBut' id='rotateBut' title='Rotate Selected'>
            <input type='button' class='iconBut' id='shiftBut' title='Shift'>
            <input type='button' class='iconBut' id='saveBut' title='Save'>
            <input type='button' class='iconBut' id='loadBut' title='Load'>
            <input type='button' class='iconBut' id='colorBut' title='Tricolor'>
        </div>

        <div id='sequencingPanel'>
            <p id = 'bankLbl'>Bank:</p>
            <div id = 'bankButtons'></div>

            <div id = 'frameButtons'></div>
            <div id = 'playPauseButs'>
                <div id = 'playBut' class = 'seqContButs'></div>
                <div id = 'rvsPlayBut' class = 'seqContButs'></div>
                <div id = 'stopBut' class = 'seqContButs'></div>
                <div id = 'prevBut' class = 'seqContButs'></div>
                <div id = 'nextBut' class = 'seqContButs'></div>
            </div>
            <div id = 'frBox'>
                    <p id = 'txtLbl'>Frame rate:</p>
                    <input type="range" id = 'slider' min="1" max="100" value="25"></input>
            </div>
            <label id = 'localBoxLbl'><input type="checkbox" id = 'localBox' name="local" checked>  Loop local
bank only</label>
```

```
            <div id="compBut">
                <div id="compLabel">
                    <p class="text2">Compare Registers</p>
                </div>
                <div id="dCompare">
                    <p id="txt2"  class="text2" >OFF</p>
                </div>
            </div>

        </div>

    <input type='file'  id='fileOpen' name='file'/>

    <div id='banner'></div>

    <div id="modalMsg" class='boxStyle'>
        <p id = "mmtext"></p>
         <input type='button' id='Confirm' class='boxStyle' value='OK'></input>
         <input type='button' id='Cancel' class='boxStyle' value='Cancel'></input>
    </div>

    <div id="modalMsg2" class='boxStyle'>
        <p id = "mmtext2"></p>
         <input type='button' id='Confirm2' class='boxStyle' value='OK'></input>
         <input type='button' id='Cancel2' class='boxStyle' value='Cancel'></input>
    </div>



    <div id="modalMsg3" class='boxStyle'>
        <p>Enter filename:</p>
        <input id='tb' type='text' name='str'></input>
         <input type='button' id='Confirm3' class='boxStyle' value='Save'></input>
         <input type='button' id='Cancel3' class='boxStyle' value='Cancel'></input>
    </div>

    <script type="text/javascript" src="js/Board.js"></script>

</body>
</html>
```

B2 styleSheet.css

```
html{height: 100%;}

body {
    background: url('../img/csbg2.png');
    background-size: 100% 100%;
    height: 700px;
    overflow: auto;
}

#buttonTray{
    position: absolute;
    top: 10px;
    left: 1145px;
    height: 185px;
    width: 95px;
}

#saveBut{
    position: absolute;
    top: 5px;
    left: 5px;
    height: 40px;
    width: 40px;
    background: url(../img/saveBut.png) no-repeat;
    background-size: 100% 100%;
  cursor: pointer;
  border: none;
}

#loadBut{
    position: absolute;
    top: 5px;
    left: 50px;
    height: 40px;
    width: 40px;
    background: url(../img/loadBut.png) no-repeat;
    background-size: 100% 100%;
  cursor: pointer;
  border: none;
```

```
}


#eraseBut{
    position: absolute;
    top: 50px;
    left: 5px;
    height: 40px;
    width: 40px;
    background: url(../img/delBut.png) no-repeat;
    background-size: 100% 100%;
  cursor: pointer;
  border: none;
}

#clearBut{
    position: absolute;
    top: 50px;
    left: 50px;
    height: 40px;
    width: 40px;
    background: url(../img/clearBut.png) no-repeat;
    background-size: 100% 100%;
  cursor: pointer;
  border: none;
}


#rotateBut{
    position: absolute;
    top: 95px;
    left: 5px;
    height: 40px;
    width: 40px;
    background: url(../img/rotateBut.png) no-repeat;
    background-size: 100% 100%;
  cursor: pointer;
  border: none;
}


#shiftBut{
    position: absolute;
```

```
    top: 95px;
    left: 50px;
    height: 40px;
    width: 40px;
    background: url(../img/arrowsBut.png) no-repeat;
    background-size: 100% 100%;
  cursor: pointer;
  border: none;
}

#invBut{
    position: absolute;
    top: 140px;
    left: 5px;
    height: 40px;
    width: 40px;
    background: url(../img/invBut.png) no-repeat;
    background-size: 100% 100%;
  cursor: pointer;
  border: none;
}


#colorBut{
    position: absolute;
    top: 140px;
    left: 50px;
    height: 40px;
    width: 40px;
    background: url(../img/colorBut.png) no-repeat;
    background-size: 100% 100%;
    border: none;
    cursor: pointer;
}

.cbSelected{
    background-image: url('../img/colorButActive2.png') !important;
    background-size: 100% 100%;
}


.iconBut:hover{
  outline-style: solid;
```

```
    outline-color: orange;
   outline-width: 1px;
    box-shadow: 3px 3px 5px orange;
}

#fileOpen{
    position: fixed;
    top: 250px;
    left: 50%;
   margin-left: -100px;
    height: 30px;
    width: 80px;
    z-index: -5;
}



/*-------------------------------------------------------------------------------*/


#sequencingPanel {
    position: absolute;
    background: url('../img/wood1.png');
    top: 575px;
    left: 170px;
    height: 180px;
    width: 1160px;
    border-radius: 10px;
    border: solid black 3px;
    z-index: 80;
}

#bankButtons{
    position: absolute;
    top: 5px;
    left: 850px;
    height: 100px;
    width: 320px;
}

#frameButtons {
    position: absolute;
    top: 65px;
```

```
    left: 400px;
    height: 100px;
    width: 770px;
}

#frameButtons p {
    font: bold 12px arial, sans-serif;
    text-align: center;
    height: 12px;
    width: 25px;
}

#bankButtons p {
    font: bold 12px arial, sans-serif;
    text-align: center;
    height: 12px;
    width: 25px;
}

#playPauseButs {
    position: absolute;
    background: none;
    top: 5px;
    left: 10px;
    height: 75px;
    width: 370px;
}

#bankLbl {
    position: absolute;
    font: bold 15px arial, sans-serif;
    top: 15px;
    left: 790px;
    height: 15px;
    width: 100px;
}

#frBox {
    position: absolute;
    top: 95px;
    left: 45px;
    height: 50px;
    width: 290px;
```

```
    border: 3px solid black;
    border-radius: 10px;
    background: tan;
}

#txtLbl {
    position: absolute;
    font: bold 15px arial, sans-serif;
    top: 0px;
    left: 10px;
    height: 15px;
    width: 100px;
}

#slider {
    position: absolute;
    top: 13px;
    left: 100px;
    cursor: pointer;
}

#localBoxLbl {
    position: absolute;
    font: bold 15px arial, sans-serif;
    top: 29px;
    left: 410px;
}

#localBox {
  -ms-transform: scale(1.4); /* IE */
  -moz-transform: scale(1.4); /* FF */
  -webkit-transform: scale(1.4); /* Safari and Chrome */
  -o-transform: scale(1.4); /* Opera */
  padding: 10px;
  cursor: pointer;
}


/*-------------------------------------------------------------------------------------------*/


#compBut {
    position: absolute;
```

```
    border: 2px solid black;
    border-radius: 8px;
    box-shadow: 1px 2px 2px brown;
    top: 25px;
    left: 600px;
    height: 25px;
    width: 170px;
}

#compLabel {
    position: absolute;
    background: tan;
    top: 0px;
    left: 0px;
    height: 25px;
    width: 135px;
    border-radius: 7px 0px 0px 7px;
}

#dCompare {
    position: absolute;
    background: red;
    top: 0px;
    left: 135px;
    height: 25px;
    width: 33px;
    border-left: 2px solid black;
    border-radius: 0px 6px 6px 0px;
    cursor: pointer;
}

#dCompare:hover {
    background: orange !important;
}




/*-------------------------------------------------------------------------------*/

.seqContButs {
    position: absolute;
    background: brown;
```

```
    border: 3px solid brown;
    border-radius: 5px;
    cursor: pointer;
}


#stopBut{
    top: 2px;
    left: 140px;
    height: 30px;
    width: 90px;
    background: url('../img/stopB.png') no-repeat;
}

#playBut{
    top: 2px;
    left: 235px;
    height: 30px;
    width: 130px;
    margin: auto;
    background: url('../img/playB.png') no-repeat;
}

.playActive{
    border-radius: 5px;
    box-shadow: 2px 2px 2px brown;
    background-image: url('../img/playActive.png') !important;
    background-size: 100% 100%;
}

#rvsPlayBut{
    top: 2px;
    left: 5px;
    height: 30px;
    width: 130px;
    background: url('../img/revPlayB.png') no-repeat;
}

.revPlayActive{
    border-radius: 5px;
    box-shadow: 2px 2px 2px brown;
    background-image: url('../img/revPlayActive.png') !important;
    background-size: 100% 100%;
```

```
}


#prevBut{
    top: 38px;
    left: 5px;
    height: 30px;
    width: 177px;
    margin: auto;
    background: url('../img/prevB.png') no-repeat;
}

#nextBut{
    top: 38px;
    left: 187px;
    height: 30px;
    width: 177px;
    margin: auto;
    background: url('../img/nextB.png') no-repeat;
}

.pHover {
    border-radius: 5px;
    box-shadow: 2px 2px 2px brown;
}



/*-------------------------------------------------------------------------------*/

.boxStyle{
    background: tan;
    color: brown;
    border: 3px solid black;
    border-radius : 7px;
    z-index: 100;
}

.msgBox {
    position: fixed;
    background: tan;
    color: brown;
   top: 50%;
```

```
   left: 50%;
   width: 300px;
   height: 30px;
    border: 4px solid black;
    border-radius : 10px;
    text-align: center;
    /*vertical-align: middle;*/
    font: bold 18px arial, sans-serif;
   margin-top: -15px;
   margin-left: -150px;
}

#modalMsg {
    position: fixed;
    visibility: hidden;
    /*text-align: center;*/
    font: bold 18px arial, sans-serif;
   top: 200px;
   left: 50%;
   /*margin-top: -15px;*/
   margin-left: -100px;
    box-shadow: 1px 1px 2px black;
   width: 200px;
   height: 100px;
    z-index: 150;
}

#mmtext {
    text-align: center;
    margin: 10px;
   top: 0px;
   left: 50%;
}

#modalMsg2 {
    position: fixed;
    visibility: hidden;
    text-align: center;
    font: bold 18px arial, sans-serif;
   top: 200px;
   left: 50%;
   margin-top: -15px;
   margin-left: -100px;
```

```
    box-shadow: 1px 1px 2px black;
   width: 300px;
   height: 150px;
    z-index: 150;
}

#modalMsg3 {
    position: fixed;
    visibility: hidden;
    text-align: center;
    font: bold 18px arial, sans-serif;
   top: 200px;
   left: 50%;
   margin-top: -15px;
   margin-left: -100px;
    box-shadow: 1px 1px 2px black;
   width: 250px;
   height: 140px;
    z-index: 150;
}

#tb {
    width: 200px;
}

#Confirm, #Confirm2, #Confirm3 {
    position: absolute;
    background: #A2FFA2;
    color: black;
    font: bold 14px arial, sans-serif;
    top: 65px;
    left: 100px;
    height: 30px;
    width:  90px;
}

#Confirm2 {
    top: 100px;
    left: 190px;
}

#Confirm3 {
    top: 100px;
```

```
    left: 150px;
}

#Confirm:hover, #Confirm2:hover, #Confirm3:hover{
    background: #00FF00;
    box-shadow: 1px 1px 2px black;
}


#Cancel, #Cancel2, #Cancel3 {
    position: absolute;
    background: #FF8585;
    color: black;
    font: bold 14px arial, sans-serif;
    top: 65px;
    left: 10px;
    height: 30px;
    width: 90px;
}

#Cancel2, #Cancel3 {
    top: 100px;
    left: 20px;
}

#Cancel3 {
    left: 10px;
}

#Cancel:hover, #Cancel2:hover, #Cancel3:hover{
    background: #FF0000;
    box-shadow: 1px 1px 2px black;
}


canvas {
    position: absolute;
    top: 5px;
    left: 5px;
    background: #E5FFFF;
    border: 2px;
    border-radius: 8px;
    box-shadow: 1px 1px 2px orange;
```

```
}

.eraserTip {
    cursor: url('../img/eraser.png'), auto;
}


#layoutTiles{
    display: none;
}

#tileMenu{
    position: fixed;
    top: 220px;
    left: 1170px;
    height: 200px;
    width: 35px;
}

.draggableTile {
    height: 30px;
    width: 30px;
    border: 1px solid black;
    background: #EBC299;
    cursor: pointer;
}

.draggableTile20 {
    height: 20px;
    width: 20px;
    border: 1px solid black;
    background: #EBC299;
    cursor: pointer;
}

.draggableTile:hover{
    outline-style: solid;
    outline-color: black;
   outline-width: 1px;
}


#resTog {
```

```
    position: fixed;
    border: 2px solid black;
    border-radius: 8px;
    box-shadow: 1px 2px 2px brown;
    top: 340px;
    left: 1170px;
    height: 25px;
    width: 150px;
}

#resTogLabel {
    position: absolute;
    background: #CB987E;
    top: 0px;
    left: 0px;
    height: 25px;
    width: 90px;
    border-radius: 6px 0px 0px 6px;
}

#resTogBut {
    position: absolute;
    background: #FFE659;
    top: 0px;
    left: 90px;
    height: 25px;
    width: 59px;
    border-left: 2px solid black;
    border-radius: 0px 6px 6px 0px;
    cursor: pointer;
}

#resTogBut:hover {
    background: orange;
}


.text {
    position: absolute;
    font: bold 16px arial, sans-serif;
    top: 2px;
    left: 10px;
    margin: auto;
```

```
}

.text2 {
    position: absolute;
    font: bold 12px arial, sans-serif;
    top: 5px;
    left: 5px;
    margin: auto;
}

#text-pane {
    position: fixed;
    top: 70px;
    left: 105px;
    width: 160px;
    float: left;
    height: 255px;
    font-family: arial, Helvetica, sans-serif;
    font-size: 13px;
    background-color: tan;
    border: 3px ridge black;
    z-index: 100;
}

#text {
    position: absolute;
    top: 30px;
    left: 5px;
    width: 150px;
    height: 220px;
    overflow: auto;
    font-family: arial, Helvetica, sans-serif;
    font-size: 13px;
    color: tan;
    background-color: black;
    z-index: 100;
}



#openPane {
```

```
    position: fixed;
    left: 1170px;
    top: 380px;
    height: 25px;
    width: 25px;
    background: url('../img/redPlus.png') no-repeat;
    background-size: 100% 100%;
    border: 2px solid black;
    border-radius: 7px;
    box-shadow: 1px 1px 2px brown;
    cursor: pointer;
}

#openPane:hover {
    background: url('../img/redPlusHov.png') no-repeat;
    background-size: 100% 100%;
}

#openPaneLabel {
    position: fixed;
    font: bold 16px arial, sans-serif;
    top: 385px;
    left: 1205px;
    margin: auto;
}

#closePane {
    position: absolute;
    top: 2px;
    left: 130px;
    height: 20px;
    width: 20px;
    background: url('../img/redEx.png') no-repeat;
    background-size: 100% 100%;
    border: 2px solid black;
    border-radius: 6px;
    cursor: pointer;
}

#closePane:hover {
    box-shadow: 1px 1px 0px;
}
```

```
#sizePane {
    position: absolute;
    top: 2px;
    left: 102px;
    height: 20px;
    width: 20px;
    background: url('../img/blueRestore.png') no-repeat;
    background-size: 100% 100%;
    border: 2px solid black;
    border-radius: 6px;
    cursor: pointer;
}

#sizePane:hover {
    box-shadow: 1px 1px 0px;
}

.closed {
        background: url('../img/greenPlus.png') no-repeat 100% 100%;
}


#resTogButton:hover{
    border: 3px solid brown;
}


.hidden {
    visibility: hidden;
}

#shiftButtons {
    display: none;
    z-index: 2;
   /*opacity: 0.9;*/
}

#upShift {
    position: absolute;
    overflow: hidden;
    height: 100px;
    width: 500px;
    left: 410px;
```

```
    top: 30px;
    background: url('../img/newArrowSheet.png') 0 0;
    z-index: 2;
   opacity: 0.2;
}
#upShift:hover{
    background: url('../img/newArrowSheet.png') 0 -100px;
   opacity: 0.4;
}

#downShift {
    position: absolute;
    overflow: hidden;
    height: 100px;
    width: 500px;
    left: 410px;
    top: 450px;
    background: url('../img/newArrowSheet.png') 0px -200px;
    z-index: 2;
   opacity: 0.2;
}

#downShift:hover{
    background: url('../img/newArrowSheet.png') 0px -300px;
   opacity: 0.4;
}

#rightShift {
    position: absolute;
    overflow: hidden;
    height: 500px;
    width: 100px;
    left: 1030px;
    top: 40px;
    background: url('../img/newArrowSheet.png') -700px 0px;
    z-index: 2;
   opacity: 0.2;
}

#rightShift:hover{
    background: url('../img/newArrowSheet.png') -800px 0px;
   opacity: 0.4;
}
```

```
#leftShift {
    position: absolute;
    overflow: hidden;
    height: 500px;
    width: 100px;
    left: 190px;
    top: 40px;
    background: url('../img/newArrowSheet.png') -500px 0px;
    z-index: 2;
   opacity: 0.2;
}
#leftShift:hover{
    background: url('../img/newArrowSheet.png') -600px 0px;
   opacity: 0.4;
}

#banner {
    position: fixed;
    top: 10px;
    left: 5px;
    height: 600px;
    width: 150px;
    background: url('../img/banner2.png');
    background-size: 100% 100%;
}


#canvasDIV, #buttonTray{
    border: 5px solid #000;
}

#canvasDIV {
    position: absolute;
    top: 10px;
    left: 170px;
    height: 550px;
    width: 970px;
    border-radius: 15px 0px 15px 15px;
    background: tan;
    box-shadow: 2px 2px 3px black;
}
```

```
#buttonTray{
    position: fixed;
    background: tan;
    border-left: none;
    border-radius: 2px 10px 10px 2px;
    box-shadow: 2px 2px 3px black;
}

.on {
    background: url('../img/on2.png');
    height: 25px;
    width: 25px;
    background-size: 100% 100%;
}

.off {
    background: url('../img/off2.png');
    height: 25px;
    width: 25px;
    background-size: 100% 100%;
}

.active {
    background: url('../img/active2.png');
    height: 25px;
    width: 25px;
    background-size: 100% 100%;
}

.activeFull {
    background: url('../img/activeFull.png');
    height: 25px;
    width: 25px;
    background-size: 100% 100%;
}




.bOn {
    background: url('../img/groupOn.png');
    height: 25px;
    width: 25px;
```

```
    background-size: 100% 100%;
}

.bOff {
    background: url('../img/groupOff.png');
    height: 25px;
    width: 25px;
    background-size: 100% 100%;
}

.bActive {
    background: url('../img/groupActive.png');
    height: 25px;
    width: 25px;
    background-size: 100% 100%;
}

.bActiveFull {
    background: url('../img/groupOnActive.png');
    height: 25px;
    width: 25px;
    background-size: 100% 100%;
}


.frameSel, .bankSel {
    cursor: pointer;
}

.frameSel:hover {
    border: 2px solid red;
    border-radius: 15px;
}

.bankSel:hover {
    border: 2px solid blue;
    border-radius: 15px;
}

.disabledElement{
   pointer-events: none;
   opacity: 0.5;
}
```

B3 Board.js

```
var elem = document.getElementById('canvas');
var cv = elem.getContext('2d');
var i, j;
var singCol, colsAdded;
var timer;

//-----------------------------------------------
//
//   Board class
//
//-----------------------------------------------

function Board(ss) {

    this.ss = ss;
    this.w = canvas.width/this.ss;
    this.h = canvas.height/this.ss;
    this.cell = new Array(this.h);
    this.tilePics = new Array();
    this.selected = null;
    this.multiSelect = null;
    this.turboBuild = null;
    this.turboNext = null;
    this.strucCreated = false;
    this.sequence = new Array();

    for (i=1; i < 10; i++) this.sequence[i] = null;

    this.regset = new RegisterSet();
    this.disPane = new DisplayPane();

    // Store tile .png filenames in array
    var names = [" ", "tileCornerTopRight", "tileCornerBotRight",
            "tileCornerBotLeft", "tileCornerTopLeft", "tileLineVert",
            "tileLineHoz", "tileCrossOver", "tileCrossUnder"];

    // Load tile images
    for (var indv = 1; indv < 33; indv++) {
        this.tilePics[indv] = new Image();
```

```
        if (indv > 24) {
            this.tilePics[indv] = document.getElementById(names[indv-24] + "Blue");
        } else if (indv > 16) {
            this.tilePics[indv]    = document.getElementById(names[indv-16] + "Green");
        } else if (indv > 8) {
            this.tilePics[indv] = document.getElementById(names[indv-8] + "Red");
        } else {
            this.tilePics[indv] = document.getElementById(names[indv]);
        }
    }

    // Init board cells
    for (i=0; i < this.h; i++) {
        this.cell[i] = new Array(this.w);
        for (j=0; j < this.w; j++) {
            this.cell[i][j] = new Cell(this, i, j);
        }
    }

    // Draw everything
    this.draw();

    // Print startup blurb to text pane
    this.disPane.add("Knotswithstanding!</br>");
    this.disPane.add("Erik Nestor</br>");
    this.disPane.add("Rhode Island College</br></br>");
}



Board.prototype.draw = function () {
    var i, j;

    cv.strokeStyle = "#E6E6E6";
    cv.lineWidth = 1.0;
    cv.beginPath();
    for (i = 1; i < this.w; i++) {
        cv.moveTo(((i*this.ss)-0.5), 0);
        cv.lineTo(((i*this.ss)-0.5), canvas.height);
        cv.closePath();
    }

    for (i = 1; i < this.h; i++) {
```

```
        cv.moveTo(0, ((i*this.ss)-0.5));
        cv.lineTo(canvas.width, ((i*this.ss)-0.5));
        cv.closePath();
    }
    cv.stroke();
    cv.lineWidth = 1.0;
    cv.strokeStyle = "#000000";

    var vert, hoz;
    for (vert = 0; vert < this.h; vert++){
        for (hoz = 0; hoz < this.w; hoz++) {
            if (this.cell[vert][hoz].type) {          // if square is occupied
                this.cell[vert][hoz].clear();         // clear it
                this.cell[vert][hoz].tile.draw(this.cell[vert][hoz]);   // and redraw
            }
        }
    }

    // If we are in build mode, mark leading tile
    if (this.turboBuild && (this.turboBuild.dormant == false)) {
        this.turboBuild.markNext(this.turboNext);
    }
};

// Return clicked cell
Board.prototype.hitXY = function(e) {
    var xPos = Math.ceil((e.pageX - $('canvas').offset().left)/this.ss);
    var yPos = Math.ceil((e.pageY - $('canvas').offset().top)/this.ss);
    var returnCell = this.cell[yPos-1][xPos-1];
    return returnCell;
}

Board.prototype.setSelect = function(c) {
    if (c.type) {
        if (this.multiSelect) {
            if (c.type && !this.multiSelect.isSelected(c)) {
                this.multiSelect.selectList[this.multiSelect.numSelected++] = c;
            }
        } else {
            this.selected = c;
        }
    }
}
```

```
Board.prototype.drawSelect = function() {
    this.draw();
    var l, t, img;

    if (this.multiSelect) {  // Multi-select mode
        var index;

        for (index = 0; index < this.multiSelect.numSelected; index++) {
            if (this.multiSelect.selectList[index].type) {
                l = ((this.multiSelect.selectList[index].col - 1) * this.ss);
                t = ((this.multiSelect.selectList[index].row - 1) * this.ss);
                this.multiSelect.selectList[index].clear();
                img = this.tilePics[this.multiSelect.selectList[index].type + 8];
                cv.drawImage(img, l, t, this.ss, this.ss);
            } else {
                this.multiSelect.selectList[index] = null;
            }
        }

        if (this.multiSelect.selectList.length == 0) {
                                this.multiSelect = null;
                    }
    } else if (this.selected) {    // Single select mode
        if (this.selected.type) {
            l = ((this.selected.col - 1) * this.ss);
            t = ((this.selected.row - 1) * this.ss);
            this.selected.clear();
            cv.drawImage(this.tilePics[this.selected.type + 8], l, t, this.ss, this.ss);
        } else {
                                this.selected = null;
        }
    }
}


Board.prototype.shift = function(dir) {

    var valid = true;
    var changed = false;
    var sel = null;
    var i, j;
```

```
// Not allow movement in any direction where
// knot is already against the grid edge
switch (dir) {
                case "upShift":
        for (i = 0; i < this.w; i++) {
            if (this.cell[0][i].type) {
                                                valid = false;
            }
        }
        break;
    case "rightShift":
        for (i = 0; i < this.h; i++) {
            if (this.cell[i][this.w-1].type) {
                                                valid = false;
            }
        }
        break;
    case "downShift":
        for (i = 0; i < this.w; i++) {
            if (this.cell[this.h-1][i].type) {
                                                valid = false;
            }
        }
        break;
    case "leftShift":
        for (i = 0; i < this.h; i++) {
            if (this.cell[i][0].type) {
                                                valid = false;
            }
        }
}


if (valid) {
    switch (dir) {
        case "upShift":
            for (i = 0; i < this.h-1; i++) {
                for (j = 0; j < this.w; j++) {
                    this.cell[i][j].plot(this.cell[i+1][j].type);
                }
            }

            if (brd.selected) {
```

```
            sel = this.cell[brd.selected.row-2][brd.selected.col-1];
        }
        for (i = 0; i < this.w; i++) {
            if (this.cell[this.h-1][i].type) this.cell[this.h-1][i].plot(0);
        }
        break;
    case "rightShift":
        for (i = (this.w-1); i > 0; i--) {
            for (j = 0; j < this.h; j++) {
                this.cell[j][i].plot(this.cell[j][i-1].type);
            }
        }
        if (brd.selected) {
            sel = this.cell[brd.selected.row-1][brd.selected.col];
        }
        for (i = 0; i < this.h; i++) {
            if (this.cell[i][0].type != 0) this.cell[i][0].plot(0);
        }
        break;
    case "downShift":
        for (i = (this.h-1); i > 0; i--) {
            for (var j = 0; j < this.w; j++) {
                this.cell[i][j].plot(this.cell[i-1][j].type);
            }
        }
        if (brd.selected) {
            sel = this.cell[brd.selected.row][brd.selected.col-1];
        }
        for (i = 0; i < this.w; i++) {
            if (this.cell[0][i].type) this.cell[0][i].plot(0);
        }
        break;
    case "leftShift":
        for (i = 0; i < (this.w-1); i++) {
            for (j = 0; j < this.h; j++) {
                this.cell[j][i].plot(this.cell[j][i+1].type);
            }
        }
        if (brd.selected) {
            sel = this.cell[brd.selected.row-1][brd.selected.col-2];
        }
        for (i = 0; i < this.h; i++) {
            if (this.cell[i][this.w-1].type) this.cell[i][this.w-1].plot(0);
```

```
                }
            }
        }
        this.draw();
        brd.setSelect(sel);
        brd.drawSelect();
}


// Display self-terminating modal message
Board.prototype.msgDisplay = function(msg, t, h, w) {
    var box = document.createElement('div');
    box.classList.add('msgBox');
    box.id  = "mb";
    document.body.appendChild(box);
    var m = document.createElement('p');
    m.id = "mg";

    $('#mb').css({'height' : h+'px', 'width' : w + 'px'});

    document.getElementById('mb').appendChild(m);

    m.innerHTML = msg;

    $('#mg').css('margin', '3px');

    timer = setTimeout(function() {
        document.body.removeChild(document.getElementById('mb'));
    }, t);
}



//-------------------------------------------------
//
//  Cell class
//
//-------------------------------------------------

function Cell(p, r, c) {
    this.col = (c+1);
    this.row = (r+1);
    this.type = 0;
```

```
    this.parent = p;
    this.tile = new Empty();
    this.nConnect =
    this.eConnect =
    this.wConnect =
    this.sConnect = null;
}

Cell.prototype.clear = function(){
    var l = (this.col - 1) * brd.ss;
    var t = (this.row - 1) * brd.ss;
    cv.clearRect(l, t, brd.ss, brd.ss);
}

Cell.prototype.plot = function(t){
    this.type = t;
    if (t == 0) {
        this.tile = new Empty(this.row, this.col);
        this.nConnect =
        this.eConnect =
        this.wConnect =
        this.sConnect = null;
    } else if (t < 5) {
        this.tile = new Corner(this.row, this.col);
        for (i=0; i < (t-1); i++) this.tile.rotate();

    } else if (t < 7) {
        this.tile = new Line(this.row, this.col);
        for (i=0; i < (t-5); i++) this.tile.rotate();

    } else if (t < 9) {
        this.tile = new Crossing(this.row, this.col);
        for (i=0; i < (t-7); i++) this.tile.rotate();
    }
    this.tile.draw();
}

Cell.prototype.highlight = function(){
        cv.fillStyle = "#FFFF00";     // Color yellow
        cv.strokeStyle = "#FFFF00";
        var l = (this.col - 1) * brd.ss;
        var r = (this.row - 1) * brd.ss;
        cv.fillRect(l, r, brd.ss, brd.ss);
```

```
        cv.fillStyle = "#000000";    // Color black again
        cv.strokeStyle = "#000000";
}

Cell.prototype.rotateTile = function(){
    this.tile.rotate();
    if ((this.type > 0) && (this.type < 9)) {
        if (this.type == 4) this.type = 1;
        else if (this.type == 6) this.type = 5;
        else if (this.type == 8) this.type = 7;
        else this.type++;
    }
}

// Return true if tile has a "dangling" end,
// not connected to an adjacent tile
Cell.prototype.isAnEnd = function() {
    return ((this.tile.north && (this.row > 1) && (!brd.cell[this.row - 2][this.col-1].type))
        || (this.tile.south && (this.row < (brd.h - 1)) && (!brd.cell[this.row][this.col-1].type))
        || (this.tile.east && (this.col < (brd.w - 1)) && (!brd.cell[this.row-1][this.col].type))
        || (this.tile.west && (this.col > 1) && (!brd.cell[this.row -1][this.col-2].type)));
}


//-------------------------------------------------
//
//  Empty class
//
//-------------------------------------------------

function Empty(r, c) {
    this.col = c;
    this.row = r;
    this.rotation = 0;
    this.north = 0;
    this.east = 0;
    this.south = 0;
    this.west = 0;
}
```

```
Empty.prototype.draw = function(){
    var l = (this.col - 1) * brd.ss;
    var t = (this.row - 1) * brd.ss;
    cv.clearRect(x1, y1, (brd.ss - 1), (brd.ss - 1));
    brd.cell[this.row-1][this.col-1].clear();
}

Empty.prototype.rotate = function(){}

//-----------------------------------------------
//
//  Corner class
//
//-----------------------------------------------

function Corner(r, c) {
    this.col = c;
    this.row = r;
    this.rotation = 0;
    this.north = 0;
    this.east = 0;
    this.south = 1;
    this.west = 1;
}

// Draw corner to cell
Corner.prototype.draw = function(){
    var l = ((this.col - 1) * brd.ss);
    var t = ((this.row - 1) * brd.ss);
    cv.clearRect(l, t, (brd.ss-1), (brd.ss-1));
    var index = this.rotation + 1;
    cv.drawImage(brd.tilePics[index], l, t, brd.ss, brd.ss);
}

// Rotate corner
Corner.prototype.rotate = function(){
    var temp;
    this.rotation = (this.rotation+1)%4;
    temp = this.north;
    this.north = this.west;
    this.west = this.south;
    this.south = this.east;
    this.east = temp;
```

```
    this.draw();
}


//-----------------------------------------------
//
//  Line class
//
//-----------------------------------------------

function Line(r, c) {
    this.col = c;
    this.row = r;
    this.rotation = 0;
    this.north = 1;
    this.east = 0;
    this.south = 1;
    this.west = 0;
}

Line.prototype.draw = function() {
    var l = ((this.col - 1) * brd.ss);
    var t = ((this.row - 1) * brd.ss);
    cv.clearRect(l, t, (brd.ss-1), (brd.ss-1));
    var index = this.rotation + 5;
    cv.drawImage(brd.tilePics[index], l, t, brd.ss, brd.ss);
}

Line.prototype.rotate = function() {
    this.north = !this.north;
    this.east = !this.east;
    this.south = !this.south;
    this.west = !this.west;
    if (this.rotation) this.rotation = 0;
    else this.rotation = 1;
    this.draw();
}

//-----------------------------------------------
//
//  Crossing Class
//
//-----------------------------------------------
```

```
function Crossing(r, c) {
    this.col = c;
    this.row = r;
    this.rotation = 0;
    this.north = 1;
    this.east = 1;
    this.south = 1;
    this.west = 1;
}

Crossing.prototype.draw = function() {
    var l = ((this.col - 1) * brd.ss);
    var t = ((this.row - 1) * brd.ss);

    cv.clearRect(l, t, (brd.ss-1), (brd.ss-1));
    cv.drawImage(brd.tilePics[7+this.rotation],l,t, brd.ss, brd.ss);
}


Crossing.prototype.rotate = function() {
    if (this.rotation == 1) this.rotation = 0;
    else this.rotation = 1;
}

//-----------------------------------------------
//
//  TurboBuild class
//
//-----------------------------------------------

function TurboBuild(b, c) {
    this.b = brd;
    this.startCell = c;         // reference to the first cell
    this.north = 0;             //
    this.east = 0;              // bool indicating legality of a move in
    this.south = 0;             // each direction from current position
    this.west = 0;              //
    this.nextMove = c;          // Type Cell, square in which target appears, not plotted yet
    this.thisMove = null;       // Type Move, last square actually plotted w/ environment
    this.lastDir = 0;           // Type int, direction travelled to enter current cell
    this.numMoves = 0;          // # moves in history
    this.moves = new Array();    // MOVE HISTORY - Array stack, all elements type Move
```

```
    this.tempTile = null;
    this.tempOn = false;
    this.dormant = false;
    this.markNext(c);               // {place arrow marks on start square}
    brd.turboNext = this.nextMove;
}


// TurboBuild.markNext() takes a reference to a Cell
// and draws arrow marks indicating available (valid) moves
TurboBuild.prototype.markNext = function(c) {

    if (c == null) return;      // if argument null get out

    if (!c.type || c.type == 5 || c.type == 6) {
        var t = (c.row-1)*brd.ss;
        var l = (c.col-1)*brd.ss;

        cv.strokeStyle = '#FF9900';       // set color yellow
        cv.fillStyle = '#FF9900';
        cv.globalAlpha = 0.4;

        // if north move is ok- north cell [exists] and [is empty or hoz line]
        if (c.row > 1
            && (brd.cell[c.row-2][c.col-1].type == 0
                || brd.cell[c.row-2][c.col-1].type == 6) ) {
            var canCross = false;
            for (i = (c.row - 2); i > -1; i--) {
                if (brd.cell[i][c.col-1].type == 0) canCross = true;
                if ((brd.cell[i][c.col-1].type < 5) && (!canCross)) i = -1;
            }
            if (canCross) {
                cv.beginPath();
                cv.moveTo(l+(brd.ss/2), t+(brd.ss*.1));
                cv.lineTo(l+(brd.ss*.35), t+(brd.ss*.3));
                cv.lineTo(l+(brd.ss*.65), t+(brd.ss*.3));
                cv.closePath();
                cv.fill();
                this.north = 1;
            }
        } else this.north = 0;
```

```
// if north cell is start cell
if (c.row > 1) {
    if (this.numMoves > 1 && (brd.cell[c.row-2][c.col-1] != this.thisMove.cell)
        && (brd.cell[c.row-2][c.col-1].tile.south) && (!c.type)) {
        cv.strokeStyle = '#FF3300';       // set color red
        cv.fillStyle = '#FF3300';
        cv.globalAlpha = 1;
        cv.beginPath();
        cv.moveTo(l+(brd.ss/2), t+(brd.ss*.1));
        cv.lineTo(l+(brd.ss*.35), t+(brd.ss*.3));
        cv.lineTo(l+(brd.ss*.65), t+(brd.ss*.3));
        cv.closePath();
        cv.fill();
        this.north = 1;
        cv.strokeStyle = '#FF9900';       // set color yellow
        cv.fillStyle = '#FF9900';
        cv.globalAlpha = 0.4;
    }
}
// if east move is ok- east cell exists and is empty or vert line
if (c.col < brd.w && (brd.cell[c.row-1][c.col].type == 0
   || brd.cell[c.row-1][c.col].type == 5) ) {
    var canCross = false;
    for (i = (c.col); i < brd.w; i++) {
        if (brd.cell[c.row-1][i].type == 0) canCross = true;
        if ((brd.cell[c.row-1][i].type < 5) && (!canCross)) i = brd.w;
    }
    if (canCross) {
        cv.beginPath();
        cv.moveTo(l+(brd.ss*.9), t+(brd.ss/2));
        cv.lineTo(l+(brd.ss*.7), t+(brd.ss*.65));
        cv.lineTo(l+(brd.ss*.7), t+(brd.ss*.35));
        cv.closePath();
        cv.fill();
        this.east = 1;
    }
} else this.east = 0;
// if east cell is start cell
if (c.col < brd.w) {
    if ((this.numMoves > 1) && (brd.cell[c.row-1][c.col] != this.thisMove.cell)
        && (brd.cell[c.row-1][c.col].tile.west) && (!c.type)) {
        cv.strokeStyle = '#FF3300';       // set color red
        cv.fillStyle = '#FF3300';
```

```
            cv.globalAlpha = 1;
            cv.beginPath();
            cv.moveTo(l+(brd.ss*.9), t+(brd.ss/2));
            cv.lineTo(l+(brd.ss*.7), t+(brd.ss*.65));
            cv.lineTo(l+(brd.ss*.7), t+(brd.ss*.35));
            cv.closePath();
            cv.fill();
            this.east = 1;
            cv.strokeStyle = '#FF9900';       // set color yellow
            cv.fillStyle = '#FF9900';
            cv.globalAlpha = 0.4;
        }
    }

    // if south move is ok- south cell [exists] and [is empty or hoz line]
    if (c.row < brd.h && (brd.cell[c.row][c.col-1].type == 0
        || this.b.cell[c.row][c.col-1].type == 6) ) {
        var canCross = false;
        for (i = (c.row); i < brd.h; i++) {
            if (brd.cell[i][c.col-1].type == 0) canCross = true;
            if ((brd.cell[i][c.col-1].type < 5) && (!canCross)) i = brd.h;
        }
        if (canCross) {
            cv.beginPath();
            cv.moveTo(l+(brd.ss/2), t+(brd.ss*.9));
            cv.lineTo(l+(brd.ss*.65), t+(brd.ss*.7));
            cv.lineTo(l+(brd.ss*.35), t+(brd.ss*.7));
            cv.closePath();
            cv.fill();
            this.south = 1;
        }
    } else {
                        this.south = 0;
    }

    // if south cell is start cell
    if (c.row < brd.h) {
        if (this.numMoves > 1 && brd.cell[c.row][c.col-1] != this.thisMove.cell
            && brd.cell[c.row][c.col-1].tile.north && !c.type) {
            cv.strokeStyle = '#FF3300';       // set color red
            cv.fillStyle = '#FF3300';
            cv.globalAlpha = 1;
            cv.beginPath();
```

```
            cv.moveTo(l+(brd.ss/2), t+(brd.ss*.9));
            cv.lineTo(l+(brd.ss*.65), t+(brd.ss*.7));
            cv.lineTo(l+(brd.ss*.35), t+(brd.ss*.7));
            cv.closePath();
            cv.fill();
            this.south = 1;
            cv.strokeStyle = '#FF9900';       // set color yellow
            cv.fillStyle = '#FF9900';
            cv.globalAlpha = 0.4;
        }
    }
    // if west move is ok- west cell [exists] and [is empty or vert line]
    if (c.col > 1 && (this.b.cell[c.row-1][c.col-2].type == 0
        || this.b.cell[c.row-1][c.col-2].type == 5) ) {
        var canCross = false;
        for (i = (c.col-2); i > -1; i--) {
            if (brd.cell[c.row-1][i].type == 0) canCross = true;
            if ((brd.cell[c.row-1][i].type < 5) && (!canCross)) i = -1;
        }
        if (canCross) {
            cv.beginPath();
            cv.moveTo(l+(brd.ss*.1), t+(brd.ss/2));
            cv.lineTo(l+(brd.ss*.3), t+(brd.ss*.35));
            cv.lineTo(l+(brd.ss*.3), t+(brd.ss*.65));
            cv.closePath();
            cv.fill();
            this.west = 1;
        }
    } else {
                            this.west = 0;
    }

    // if west cell is start cell
    if (c.col > 1) {
        if (this.numMoves > 1 && brd.cell[c.row-1][c.col-2] != this.thisMove.cell
            && brd.cell[c.row-1][c.col-2].tile.east && !c.type) {
            cv.strokeStyle = '#FF3300';       // set color red
            cv.fillStyle = '#FF3300';
            cv.globalAlpha = 1;
            cv.beginPath();
            cv.moveTo(l+(brd.ss*.1), t+(brd.ss/2));
            cv.lineTo(l+(brd.ss*.3), t+(brd.ss*.35));
            cv.lineTo(l+(brd.ss*.3), t+(brd.ss*.65));
```

```
                cv.closePath();
                cv.fill();
                this.west = 1;
                cv.strokeStyle = '#FF9900';       // set color yellow
                cv.fillStyle = '#FF9900';
                cv.globalAlpha = 0.4;
            }
        }
        cv.strokeStyle = '#000000';
        cv.fillStyle = '#000000';
        cv.globalAlpha = 1;

        // TRAPPED! No possible moves, must backtrack
        if (!(this.north + this.east + this.south + this.west)) {
            var delay;
            cv.strokeStyle = '#FF3300';       // set color red
            cv.fillStyle = '#FF3300';
            cv.globalAlpha = .5;
            cv.fillRect(l+1, t+1, (brd.ss-2), (brd.ss-2));   // flash red TRAPPED! cell
            cv.fillStyle = "#000000";
            cv.strokeStyle = "#000000";
        }
    }
}

// TurboBuild.move(direction) takes a numeric direction argument
// (1 - east, 2 - south, 3 - west, 4 - north, 5 - backspace) and executes a move
// in that direction, including placement of the appropriate tile, updating
// the TurboBuild.moves[] move history stack, and calling TurboBuild.markNext()
// to reposition TurboBuild cursor.
TurboBuild.prototype.move = function(dir) {
    var tp;                 // holds tile-piece type
    var complete = false;
    var nm = this.nextMove;    // this is the Cell that will be "placed"
    var valid = false;

    if (dir < 5) {                      // IF MOVE IS A FORWARD MOVE
        if (!this.lastDir) {
            this.lastDir = dir;         // if this is the first move, set TurboBuild.lastDir
        }
        this.thisMove = new Move(this.nextMove);     // create new Move object
    }
```

```
// Manage individual direction cases
switch (dir) {
    case 0: break;
    // case east
    case 1:
        if (this.east) {        // check if valid
            valid = true;
            // an east move will be either a hoz line,
            // swCorner, or nwCorner depending on last dir
            if (this.lastDir == 1) tp = 6;    // set hoz line
            if (this.lastDir == 2) tp = 3;    // set swCorner
            if (this.lastDir == 4) tp = 4;    // set nwCorner
            // set TurboBuild.nextMove for one cell to the east
            this.nextMove = brd.cell[this.thisMove.cell.row-1][this.thisMove.cell.col];
            // if there was already a tile here it must be a vert line,
            // so place undercrossing
            if (this.thisMove.cell.type) tp = 8;
            // recognize circuit complete
            if ( (this.nextMove.tile.west) ) complete = true;
        }
        break;
    // case south
    case 2:
        if (this.south) {
            valid = true;
            // a south east move will be either a vert line,
            // neCorner, or nwCorner depending on last dir
            if (this.lastDir == 2) tp = 5;    // set vert line
            if (this.lastDir == 1) tp = 1;    // set neCorner
            if (this.lastDir == 3) tp = 4;    // set nwCorner
            // set TurboBuild.nextMove for one cell to the south
            this.nextMove = this.b.cell[this.thisMove.cell.row][this.thisMove.cell.col-1];
            // if there was already a tile here, it must be a hoz line, so place overcrossing
            if (this.thisMove.cell.type) tp = 7;
            // recognize circuit complete
            if ((this.nextMove.tile.north)) complete = true;
        }
        break;
    // case west
    case 3:
        if (this.west) {
            valid = true;
            // a west move will be either a hoz line,
```

```
            // seCorner, or neCorner depending on last dir
            if (this.lastDir == 3) tp = 6;    // set hoz line
            if (this.lastDir == 2) tp = 2;    // set seCorner
            if (this.lastDir == 4) tp = 1;    // set neCorner
            // set TurboBuild.nextMove for one cell to the west
            this.nextMove = this.b.cell[this.thisMove.cell.row-1][this.thisMove.cell.col-2];
            // if there was already a tile here, it must be a vert line, so place undercrossing
            if (this.thisMove.cell.type) tp = 8;
            // recognize circuit complete
            if ((this.nextMove.tile.east)) complete = true;
        }
        break;
    // case north
    case 4:
        if (this.north) {
            valid = true;
            // a north move will be either a vert line,
            // seCorner, or swCorner depending on last dir
            if (this.lastDir == 4) tp = 5;    // set vert line
            if (this.lastDir == 1) tp = 2;    // set seCorner
            if (this.lastDir == 3) tp = 3;    // set swCorner
            // set TurboBuild.nextMove for one cell to the north
            this.nextMove = this.b.cell[this.thisMove.cell.row-2][this.thisMove.cell.col-1];
            // if there was already a tile here, it must be a hoz line, so place overcrossing
            if (this.thisMove.cell.type) tp = 7;
            // recognize circuit complete
            if ((this.nextMove.tile.south)) complete = true;
        }
        break;
    // case backspace
    case 5:
        // IF there are moves in history to be undone
        if (this.numMoves > 0) {
            // erase square containing previous cursor (if otherwise unoccupied)
            if (!this.nextMove.type) this.nextMove.clear();
            var pop = this.moves.pop();          // pop last move from stack
            this.nextMove = pop.cell;    //.nextMove is a Cell object
            this.numMoves--;       // decrement moves counter
            this.north = this.thisMove.north;
            this.east = this.thisMove.east;
            this.south = this.thisMove.south;
            this.west = this.thisMove.west;
```

```
            if (this.numMoves > 0) {
                this.thisMove = this.moves[this.numMoves-1];
                this.lastDir = pop.lastDir;
            }

            if (this.nextMove.type > 6) {
                if ((this.lastDir == 1) || (this.lastDir == 3)) this.nextMove.plot(5);
                else if ((this.lastDir == 2) || (this.lastDir == 4)) this.nextMove.plot(6);
            } else {
                                        this.nextMove.plot(0);
            }

            brd.turboNext = this.nextMove;

            if (this.numMoves == 0) {
                this.lastDir = 0;
                this.thisMove = null;
                brd.draw();
            } else {
                brd.setSelect(this.thisMove.cell);
                brd.drawSelect();
            }
        }
    }
}

if (valid) {
    this.numMoves++;
    this.thisMove.cell.plot(tp);
    this.thisMove.north = this.north;
    this.thisMove.east = this.east;
    this.thisMove.south = this.south;
    this.thisMove.west = this.west;
    this.thisMove.nextMove = this.nextMove;
    this.thisMove.thisMove = this.thisMove;
    this.thisMove.lastDir = this.lastDir;
    this.moves.push(this.thisMove);
    this.lastDir = dir;
    if (complete) this.completeCircuit(this.thisMove.cell);
    else {
        brd.setSelect(this.thisMove.cell);
        brd.drawSelect();
        this.markNext(this.nextMove);
```

```
            brd.turboNext = this.nextMove;
        }
    }
}

TurboBuild.prototype.edgeFind = function(c) {
    var count = 0;
    var tiles = new Array();
    var tileNum = 0;
    var temp, index;
    var dir = 0;
    var ce;

    // Count adjoining occupied squares
    // 'ce' will be set to the last one counted
    if ((c.row > 1) && brd.cell[c.row-2][c.col-1].type) {
        ce = brd.cell[c.row-2][c.col-1];
        count++;
    }
    if ((c.row < (brd.h-1)) && brd.cell[c.row][c.col-1].type) {
        ce = brd.cell[c.row][c.col-1];
        count++;
    }
    if ((c.col > 1) && brd.cell[c.row-1][c.col-2].type) {
        ce = brd.cell[c.row-1][c.col-2];
        count++;
    }
    if ((c.col < (brd.w-1)) && brd.cell[c.row-1][c.col].type) {
        ce = brd.cell[c.row-1][c.col];
        count++;
    }
    if ((count < 2) && ce) {
                    this.tryPickup(c, ce);   // no ambiguity, we'll try to pick up on it
    }

    count = 0;
    if ((c.row > 1)  && brd.cell[c.row-2][c.col-1]) {
        ce = brd.cell[c.row-2][c.col-1];
        if ((ce.type)  && (ce.tile.south)) {
            temp = brd.cell[c.row-2][c.col-1];
            count++;
            dir = 4;
        }
```

```
}
if ((c.row < (brd.h-1)) && brd.cell[c.row][c.col-1]) {
    ce = brd.cell[c.row][c.col-1];
    if ((ce.type) && (ce.tile.north)) {
        temp = brd.cell[c.row][c.col-1];
        count++;
        dir = 2;
    }
}
if ((c.col > 1) && brd.cell[c.row-1][c.col-2]) {
    ce = brd.cell[c.row-1][c.col-2];
    if ((ce.type) && (ce.tile.east)) {
        temp = brd.cell[c.row-1][c.col-2];
        count++;
        dir = 3;
    }
}
if ((c.col < (brd.w-1)) && brd.cell[c.row-1][c.col]) {
    ce = brd.cell[c.row-1][c.col];
    if ((ce.type) && (ce.tile.west)) {
        temp = brd.cell[c.row-1][c.col];
        count++;
        dir = 1;
    }
}

if (count == 1) {
    while (temp && temp.type) {
        tiles[tileNum] = new Move(temp);
        tiles[tileNum].lastDir = dir;
        tiles[tileNum].thisMove =  temp;

        switch (temp.type) {

            case 1 :
            if (dir == 1) {
                if (tiles[tileNum].cell.row < (canvas.height/brd.ss)) {
                    if (brd.cell[tiles[tileNum].cell.row][tiles[tileNum].cell.col-1].type) {
                        temp = brd.cell[tiles[tileNum].cell.row][tiles[tileNum].cell.col-1];
                        dir = 2;
                    } else {
                        temp = null;
                    }
```

```
    } else {
            temp = null;
    }
} else if (dir == 4) {
    if (tiles[tileNum].cell.col > 1) {
        if (brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col-2].type) {
                temp = brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col-2];
                dir = 3;
        } else {
            temp = null;
        }
    } else {
        temp = null;
    }
 } else {
    temp = null;
}
 break;

case 2 :
if (dir == 1) {
    if (tiles[tileNum].cell.row > 1) {
        if (brd.cell[tiles[tileNum].cell.row-2][tiles[tileNum].cell.col-1].type) {
            temp = brd.cell[tiles[tileNum].cell.row-2][tiles[tileNum].cell.col-1];
            dir = 4;
        } else {
            temp = null;
        }
    } else temp = null;
} else if (dir == 2) {
    if (tiles[tileNum].cell.col > 1) {
        if (brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col-2].type) {
                temp = brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col-2];
                dir = 3;
        } else {
            temp = null;
        }
    } else {
        temp = null;
    }
 } else {
    temp = null;
}
```

```
 break;


case 3 :
if (dir == 2) {
    if (tiles[tileNum].cell.col < (canvas.width/brd.ss)) {
        if (brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col].type) {
                temp = brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col];
                dir = 1;
        } else {
            temp = null;
        }
    } else {
        temp = null;
    }
} else if (dir == 3) {
    if (tiles[tileNum].cell.row > 1) {
        if (brd.cell[tiles[tileNum].cell.row-2][tiles[tileNum].cell.col-1].type) {
                temp = brd.cell[tiles[tileNum].cell.row-2][tiles[tileNum].cell.col-1];
                dir = 4;
        } else {
            temp = null;
        }
    } else {
        temp = null;
    }
 } else {
    temp = null;
}
 break;

case 4 :
if (dir == 3) {
    if (tiles[tileNum].cell.row < (canvas.height/brd.ss)) {
        if (brd.cell[tiles[tileNum].cell.row][tiles[tileNum].cell.col-1].type) {
            temp = brd.cell[tiles[tileNum].cell.row][tiles[tileNum].cell.col-1];
            dir = 2;
        } else {
            temp = null;
        }
    } else {
        temp = null;
    }
```

```
} else if (dir == 4) {
    if (tiles[tileNum].cell.col < (canvas.width/brd.ss)) {
        if (brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col].type) {
            temp = brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col];
            dir = 1;
        } else {
            temp = null;
        }
    }  else {
        temp = null;
    }
 } else temp = null;
 break;

case 5 :
if (dir == 2) {
    if (tiles[tileNum].cell.row < (canvas.height/brd.ss)) {
        if (brd.cell[tiles[tileNum].cell.row][tiles[tileNum].cell.col-1].type) {
            temp = brd.cell[tiles[tileNum].cell.row][tiles[tileNum].cell.col-1];
        } else {
            temp = null;
        }
    } else temp = null;
} else if (dir == 4) {
    if (tiles[tileNum].cell.row > 1) {
        if (brd.cell[tiles[tileNum].cell.row-2][tiles[tileNum].cell.col-1].type) {
            temp = brd.cell[tiles[tileNum].cell.row-2][tiles[tileNum].cell.col-1];
        } else {
            temp = null;
        }
    } else temp = null;
 } else temp = null;
 break;

case 6 :
if (dir == 1) {
    if (tiles[tileNum].cell.col < (canvas.width/brd.ss)) {
        if (brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col].type) {
                temp = brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col];
        } else temp = null;
    } else temp = null;
} else if (dir == 3) {
    if (tiles[tileNum].cell.col > 1) {
```

```
                if (brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col-2].type) {
                        temp = brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col-2];
                } else temp = null;
            } else temp = null;
        } else temp = null;
        break;

        case 7 :
        case 8 :
        if (dir == 1) {
            if (tiles[tileNum].cell.col < (canvas.width/brd.ss)) {
                if (brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col].type) {
                        temp = brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col];
                } else temp = null;
            } else temp = null;
        }

        if (dir == 2) {
            if (tiles[tileNum].cell.row < (canvas.height/brd.ss)) {
                if (brd.cell[tiles[tileNum].cell.row][tiles[tileNum].cell.col-1].type) {
                        temp = brd.cell[tiles[tileNum].cell.row][tiles[tileNum].cell.col-1];
                } else temp = null;
            } else temp = null;
        }
        if (dir == 3) {
            if (tiles[tileNum].cell.col > 1) {
                if (brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col-2].type) {
                        temp = brd.cell[tiles[tileNum].cell.row-1][tiles[tileNum].cell.col-2];
                } else temp = null;
            } else temp = null;
         }
        if (dir == 4) {
            if (tiles[tileNum].cell.row > 1) {
                if (brd.cell[tiles[tileNum].cell.row-2][tiles[tileNum].cell.col-1].type) {
                        temp = brd.cell[tiles[tileNum].cell.row-2][tiles[tileNum].cell.col-1];
                } else temp = null;
            } else temp = null;
         }
         break;
    }
    tiles[tileNum].nextMove = temp;
    tileNum++;
}
```

```
        tiles.reverse();              // reverse array
        this.moves = tiles;
        this.numMoves = this.moves.length;
        this.startCell = this.moves[0].cell;    // set startCell to first element
        this.nextMove = c;                      // set nextMove to current cell
        this.moveAdded = false;
        brd.turboNext = this.nextMove;          // set next move on board object
        this.thisMove = this.moves[this.numMoves - 1];   // Last Move plotted
        for (index = 0; index < this.numMoves; index++) {
            this.moves[index].lastDir = ((this.moves[index].lastDir + 2) % 4);
            if(!this.moves[index].lastDir) this.moves[index].lastDir = 4;
            if (index < (this.numMoves - 1)) {
                this.moves[index].nextMove = this.moves[index+1].cell;
            }
        }
        this.lastDir =  this.moves[this.numMoves-1].lastDir;
        brd.setSelect(this.thisMove.cell);
        brd.drawSelect();
        this.markNext(c);
        this.dormant = false;
    } else if ((count == 2) || (count == 4)) {
        brd.cell[c.row-1][c.col-1].plot(this.autoFill(c));
        var a = new Analysis(brd);
        if (a.isValid()) {
            if (brd.strucCreated) {
                var str = "No.";
                if (a.tricolorable) str = "Yes.";
                brd.disPane.add("Valid structure created.</br></br>");
                brd.disPane.add("Number of crossings: " + a.crossingNum + "</br>");
                brd.disPane.add("Tricolorable: " + str + "</br></br>");
                brd.disPane.add("Dowker notation:</br>" + a.extractDowkers() + "</br></br>");
            } else {
                                    brd.disPane.add("Unknot created.</br></br>");
            }
        }
    } else if (count == 3) {
                brd.cell[c.row-1][c.col-1].plot(this.autoFill(c));
    }
}

TurboBuild.prototype.tryPickup = function(c, ce) {
    // c = where we are, ce = cell we are trying to grab
```

```
var ret = ce.type;
var index;
var dist = c.row - ce.row;

// WE ARE IN THE SAME COLUMN ABOVE OR BELOW AN END SQUARE
if ((Math.abs(dist) == 1) && (c.col == ce.col)) {

    if (dist == -1) {
        if (ce.col > 1 && ce.col < brd.w-1
            && brd.cell[ce.row-1][ce.col-2].tile.east
            && !brd.cell[ce.row-1][ce.col].type) {      // we are above
            ret = 2;
        } else if (ce.col > 1 && ce.col < brd.w-1
             && brd.cell[ce.row-1][ce.col].tile.west
             && !brd.cell[ce.row-1][ce.col-2].type) {
            ret = 3;
        }  else if (ce.row > 1 && ce.row < brd.h-1
             && brd.cell[ce.row][ce.col-1].tile.north
             && brd.cell[ce.row-2][ce.col-1] == c) {
            ret = 5;
        }
        this.lastDir = 4;
    } else {                                    // we are below
        if (ce.col > 1 && ce.col < brd.w-1
            && brd.cell[ce.row-1][ce.col-2].tile.east
            && !brd.cell[ce.row-1][ce.col].type) {
            ret = 1;
        } else if (ce.col > 1 && ce.col < brd.w-1
             && brd.cell[ce.row-1][ce.col].tile.west
             && !brd.cell[ce.row-1][ce.col-2].type) {
            ret = 4;
        }  else if (ce.row > 1 && ce.row < brd.h-1
             && brd.cell[ce.row-2][ce.col-1].tile.south
             && brd.cell[ce.row][ce.col-1] == c) {
            ret = 5;
        }
        this.lastDir = 2;
    }

} else {

    dist = (c.col - ce.col);
    // IF WE ARE IN THE SAME ROW TO THE LEFT OR RIGHT OF AN END SQUARE
```

```
        if (Math.abs(dist) == 1 && c.row == ce.row) {
            if (dist == -1) {                       // we are to the left
                if (ce.row > 1 && ce.row < brd.h-1
                    && brd.cell[ce.row-2][ce.col-1].tile.south
                    && !brd.cell[ce.row][ce.col-1].type) {    // last dir = 2
                    ret = 2;
                } else if (ce.row > 1 && ce.row < brd.h-1
                            && brd.cell[ce.row][ce.col-1].tile.north
                     && !brd.cell[ce.row-2][ce.col-1].type) {   // last dir = 4
                    ret = 1;
                } else if (ce.col > 1 && ce.col < brd.w-1
                     && brd.cell[ce.row-1][ce.col].tile.west
                     && brd.cell[ce.row-1][ce.col-2] == c) {    // last dir = 3
                    ret = 6;
                }
                this.lastDir = 3;
            } else {                            // we are to the right
                if (ce.row > 1 && ce.row < brd.h-1
                    && brd.cell[ce.row][ce.col-1].tile.north
                    && !brd.cell[ce.row-2][ce.col-1].type) {             // last dir = 4
                    ret  = 4;
                } else if (ce.row > 1 && ce.row < brd.h-1
                     && brd.cell[ce.row-2][ce.col-1].tile.south
                     && !brd.cell[ce.row][ce.col-1].type)  {      // last dir = 2
                    ret = 3;
                } else if (ce.col > 1 && ce.col < brd.w-1
                     && brd.cell[ce.row-1][ce.col-2].tile.east
                     && brd.cell[ce.row-1][ce.col] == c) {          // last dir = 1
                    ret = 6;
                }
                this.lastDir = 1;
            }
        }
    }

    if (ce.type < 7) ce.plot(ret);
    brd.draw();
    return ret;
}

TurboBuild.prototype.completeCircuit = function (c){

    brd.turboNext = null;
```

```
    brd.setSelect(c);
    brd.drawSelect();
    this.dormant = true;

    var a = new Analysis(brd);

    if (a.isValid()) {
        if (brd.strucCreated) {
            var str = "No.";
            if (a.tricolorable) str = "Yes.";
            brd.disPane.add("Valid structure created.</br></br>");
            brd.disPane.add("Number of crossings: " + a.crossingNum + "</br>");
            brd.disPane.add("Tricolorable: " + str + "</br></br>");
            brd.disPane.add("Dowker notation:</br>" + a.extractDowkers() + "</br></br>");
        } else {
            brd.disPane.add("Unknot created.</br></br>");
        }
    }
}

TurboBuild.prototype.autoFill = function (c) {

    var n,s,e,w;
    n = s = e = w = false;
    var count = 0;
    var type;

    if (!c.type) {
        if (c.row > 1
           && brd.cell[c.row-2][c.col-1].type
            && brd.cell[c.row-2][c.col-1].tile.south) {
            n = true;
            count++;
        }
        if(c.row < ((canvas.height/brd.ss)-1)
            && brd.cell[c.row][c.col-1].type
            && brd.cell[c.row][c.col-1].tile.north) {
            s = true;
            count++;
        }
        if (c.col > 1
            && brd.cell[c.row-1][c.col-2].type
            && brd.cell[c.row-1][c.col-2].tile.east) {
```

```
            w = true;
            count++;
        }
        if(c.col < ((canvas.width/brd.ss) - 1)
            && brd.cell[c.row-1][c.col].type
            && brd.cell[c.row-1][c.col].tile.west) {
            e = true;
            count++;
        }

        if (n && e && s && w) type = 7;
        else if (count == 3) {
            if (n && s) type = 5;
            else if (e && w) type = 6;
        } else {
            if (n && e) type = 3;
            if (n && s) type = 5;
            if (n && w) type = 2;
            if (e && s) type = 4;
            if (e && w) type = 6;
            if (s && w) type = 1;
        }

        return type;
    }
}

//-----------------------------------------------
//
//       Move class
//     Models cell environment during TurboBuild
//
//-----------------------------------------------

function Move (cell) {

    this.cell = cell;
    this.north = 0;      //
    this.east = 0;       //    Legality (bool) of a move in each direction
    this.south = 0;      //
    this.west = 0;       //
    this.nextMove = null;
    this.thisMove = null;
```

```
    this.lastDir = 0;    //     Direction traveled to enter current cell

}

function BoardCapture(b) {
    this.numCells = 0;
    this.cells = new Array(b.w * b.h);

    var x, y;
    for (x = 0; x < b.h; x++) {
        for (y = 0; y < b.w; y++) {
            if (b.cell[x][y].type) {
                this.cells[this.numCells] = new SquareSave((y+1),(x+1),b.cell[x][y].type);
                this.numCells++;
            }
        }
    }
}

function SquareSave(r, c, t) {
    this.row = r;
    this.col = c;
    this.type = t;
}

//---------------------------------------------

function SelectStruc() {
    this.selectList = new Array();
    this.numSelected = 0;
}

SelectStruc.prototype.isSelected = function(cel){
    var tf = false;
    var index;
    for (index = 0; (index < this.numSelected) && (!tf); index++) {
        if (this.selectList[index] == cel) tf = true;
    }
    return tf;
}

SelectStruc.prototype.remove = function(cel){
    var newList = new Array();
```

```
    var newIndex = 0;
    var index;
    for (index = 0; index < this.numSelected; index++) {
        if (!(this.selectList[index] == cel)) {
            newList[newIndex++] = this.selectList[index];
        } else {
            var subIndex;
            for (subIndex = (index + 1); subIndex < this.numSelected; subIndex++) {
                newList[newIndex++] = this.selectList[subIndex];
            }
            index = this.numSelected--;
            this.selectList = newList;
        }
    }
}

function XOR(a,b) {
  return ( a || b ) && !( a && b );
}

var brd = new Board(30);
```

B4 DisplayPane.js

```
function DisplayPane() {
    this.opened = 1;
    this.size = 1;
    this.pane = document.getElementById('text-pane');
    this.content = "";
}

DisplayPane.prototype.reset = function() {
    $('#text-pane').empty();
}

DisplayPane.prototype.add = function(st) {
    document.getElementById('text').innerHTML += st;
    //$('#text').text(st);
}

DisplayPane.prototype.changeSize = function() {
    this.size = this.size%3 + 1;

    switch (this.size) {
        case 1:
            $('#text-pane').css('height',  '255px');
            $('#text-pane').css('width', '160px');
            $('#closePane').css('left', '130px');
            $('#sizePane').css('left', '102px');
            $('#text').css('height', '220px');
            $('#text').css('width', '150px');
            break;
        case 2:
            $('#text-pane').css('height',  '600px');
            $('#text-pane').css('width', '350px');
            $('#closePane').css('left', '320px');
            $('#sizePane').css('left', '292px');
            $('#text').css('height', '565px');
            $('#text').css('width', '340px');
            break;
        case 3:
            $('#text-pane').css('height',  '200px');
            $('#text-pane').css('width', '950px');
            $('#closePane').css('left', '920px');
```

```
            $('#sizePane').css('left', '892px');
            $('#text').css('height', '165px');
            $('#text').css('width', '940px');
    }
}

$('#closePane').click(function(e){
    if (brd.disPane.opened) {
        $('#text-pane').css('visibility', 'hidden');
        $('#closePane').removeClass('opened');
        $('#closePane').addClass('closed');
        brd.disPane.opened = 0;
    } else {
        $('#text-pane').css('visibility', 'visible');
        $('#closePane').addClass('opened');
        $('#closePane').removeClass('closed');
        brd.disPane.opened = 1;
    }
});
```

B5 RegisterSet.js

```
function RegisterSet() {

    this.led = new Array();
    this.ledBank = new Array();
    this.activePage = 0;
    this.activeBank = 0;
    this.playing = 0;
    this.dowkerCheckActive = 0;
    this.dowkerCheckWasActive = 0;
    this.nums = new Array();
    this.bankNums = new Array();

    var framButs = document.getElementById('frameButtons');
    var bankButs = document.getElementById('bankButtons');

    var t, l, tn, ln;

    for (i=0; i < 50; i++) {
        this.led[i] = document.createElement('div');
        this.nums[i] = document.createElement('p');
        this.nums[i].innerHTML += (i+1);
        this.led[i].classList.add('off');
        this.led[i].classList.add('frameSel');
        framButs.appendChild(this.led[i]);
        framButs.appendChild(this.nums[i]);
        this.led[i].id = "div-" + i;
        this.nums[i].id = "num-" + i;
        if (i < 25) {
            t = '27px';
            tn = '0px';
        } else {
            t = '77px';
            tn = '50px';
        }

        if (i < 25) {
            l = (i*30) + 'px';
        } else {
            l = ((i-25)*30) + 'px';
        }
```

```
    $("#" + this.led[i].id).css({
        "position": "absolute",
        "top": t,
        "left": l
    });

    $("#" + this.nums[i].id).css({
        "position": "absolute",
        "top": tn,
        "left": l
    });

    this.led[0].classList.add('active');
}

for (i=0; i < 10; i++) {
    this.ledBank[i] = document.createElement('div');
    this.bankNums[i] = document.createElement('p');
    this.bankNums[i].innerHTML += (i+1);
    this.ledBank[i].classList.add('bOff');
    this.ledBank[i].classList.add('bankSel');
    bankButs.appendChild(this.ledBank[i]);
    bankButs.appendChild(this.bankNums[i]);
    this.ledBank[i].id = "bdiv-" + i;
    this.bankNums[i].id = "bnum-" + i;
    t = '27px';
                tn = '0px';
    l = (i*30) + 'px';

    $("#" + this.ledBank[i].id).css({
        "position": "absolute",
        "top": t,
        "left": l
    });

    $("#" + this.bankNums[i].id).css({
        "position": "absolute",
        "top": tn,
        "left": l
    });

    this.ledBank[0].classList.add('bActive');
}
```

```
}

RegisterSet.prototype.reset = function() {

    this.activeBank = 0;
    this.activePage = 0;

    for (x=0; x < 10; x++) {
        if (!x) {
            $('#bdiv-' + x).addClass('bActive');
        } else {
            $('#bdiv-' + x).removeClass('bOn');
            $('#bdiv-' + x).removeClass('bActive');
            $('#bdiv-' + x).removeClass('bActiveFull');
            $('#bdiv-' + x).addClass('bOff');
        }
    }

    for (x=0; x < 50; x++) {
        if (!x) {
            $('#div-' + x).addClass('active');
        } else {
            $('#div-' + x).removeClass('on');
            $('#div-' + x).removeClass('active');
            $('#div-' + x).removeClass('activeFull');
            $('#div-' + x).addClass('off');
        }
    }
}

RegisterSet.prototype.switchRegister = function(reg) {

    // restore reg led for prev register
    $("#div-" + this.activePage).removeClass('activeFull');
    $("#div-" + this.activePage).removeClass('active');
    if (brd.sequence[this.activeBank] && brd.sequence[this.activeBank][this.activePage]) {
                    $("#div-" + this.activePage).addClass('on');
    } else {
                    $("#div-" + this.activePage).addClass('off');
    }
```

```
    // switch active register
    this.activePage = reg;
          // turn on reg led for new register
    $("#div-" + reg).removeClass('on');
    $("#div-" + reg).removeClass('off');
    if (brd.sequence[this.activeBank] && brd.sequence[this.activeBank][this.activePage]) {
            $("#div-" + reg).addClass('activeFull');
    } else {
            $("#div-" + reg).addClass('active');
    }
}

RegisterSet.prototype.switchBank = function(bnk) {

    // restore reg led for prev bank
    $("#bdiv-" + this.activeBank).removeClass('bActiveFull');
    $("#bdiv-" + this.activeBank).removeClass('bActive');
    if (brd.sequence[this.activeBank]) {
            $("#bdiv-" + i).addClass('bOn');
    } else {
            $("#bdiv-" + this.activePage).addClass('bOff');
    }

    // switch active register
    this.activeBank = bnk;
    this.activePage = 0;

    this.update();
}


RegisterSet.prototype.update = function() {

    var i;
    for (i = 0; i < brd.sequence.length; i++) {
        if (brd.sequence[i]) {
            //turn bank on
            $("#bdiv-" + i).removeClass('bActive');
            $("#bdiv-" + i).removeClass('bOff');
            $("#bdiv-" + i).addClass('bOn');
            if (i == this.activeBank) {
                                $("#bdiv-" + i).addClass('bActiveFull');
            }
```

```
    } else {
        // turn bank off
        if (i == this.activeBank) {
                                $("#bdiv-" + i).addClass('bActive');
        }
        $("#bdiv-" + i).addClass('bOff');
        $("#bdiv-" + i).removeClass('bOn');
        $("#bdiv-" + i).removeClass('bActiveFull');
    }
}

if (brd.sequence[this.activeBank]) {
    for (i = 0; i < 50; i ++) {
        if (brd.sequence[this.activeBank][i]) {
            //turn reg on
            $("#div-" + i).removeClass('active');                    // reg populated
            $("#div-" + i).removeClass('off');
            $("#div-" + i).addClass('on');
            if (i == this.activePage) {
                                    $("#div-" + i).addClass('activeFull');
            } else {
                                    $("#div-" + i).removeClass('activeFull');
            }
        } else {
            //turn reg off
            $("#div-" + i).removeClass('activeFull');                // reg not populated
            $("#div-" + i).removeClass('on');
            $("#div-" + i).addClass('off');
            if (i == this.activePage) {
                                    $("#div-" + i).addClass('active');
            } else {
                                    $("#div-" + i).removeClass('active');
            }
        }
    }
} else {                                    // nothing in bank
        for (i = 0; i < 50; i++) {
                if (i == this.activePage) {
                        //turn reg on
                        $("#div-" + i).removeClass('activeFull');
                        $("#div-" + i).removeClass('off');
                        $("#div-" + i).removeClass('on');
                        $("#div-" + i).addClass('active');
```

```
                } else {
                        //turn reg off
                        $("#div-" + i).removeClass('activeFull');
                        $("#div-" + i).removeClass('on');
                        $("#div-" + i).removeClass('active');
                        $("#div-" + i).addClass('off');
                }
        }
    }
}


RegisterSet.prototype.newSet = function() {

                var x  = 0;
                while (!brd.sequence[x]) x++;

                this.activeBank = x;
                this.activePage = 0;

                for (x=0; x < 10; x++){

                    if (brd.sequence[x]) {
                                $('#bdiv-' + x).removeClass('bOff');
                                $('#bdiv-' + x).addClass('bOn');
                                if (x == this.activeBank) $('#bdiv-' + x).addClass('bActiveFull');
                    } else {
                                $('#bdiv-' + x).removeClass('bOn');
                                $('#bdiv-' + x).removeClass('bActiveFull');
                                $('#bdiv-' + x).addClass('bOff');
                                if (x == this.activeBank) $('#bdiv-' + x).addClass('bActive');
                                else $('#bdiv-' + x).removeClass('bActive');
                    }
                }


                for (x=0; x < 50; x++){

                    if (brd.sequence[this.activeBank][x]) {
                                $('#div-' + x).removeClass('off');
                                $('#div-' + x).addClass('on');
                    } else {
                                $('#div-' + x).removeClass('on');
```

```
                        $('#div-' + x).removeClass('active');
                        $('#div-' + x).removeClass('activeFull');
                        $('#div-' + x).addClass('off');
                    }

                }


                if (brd.sequence[this.activeBank][this.activePage]) {
                    $("#div-" + this.activePage).addClass('activeFull');

                    var y;
                    for (y=0 ; y < (brd.sequence[this.activeBank][this.activePage].length) ; y++) {
                        brd.cell[brd.sequence[this.activeBank][this.activePage][y].r-
1][brd.sequence[this.activeBank][this.activePage][y].c-1].plot(brd.sequence[this.activeBank][this.activePage][y].t);
                    }
                } else {
                    $("#div-" + this.activePage).addClass('active');
                }

}
```

B6 EventHandling.js

```
$(document).ready(function(){

    var mouseDown = 0;
    var mouseWasDown = 0;
    var rightMouseDown = 0;
    var multiBank = 0;
    var rate;
    var color = false;
    var playingInColor = false;
    var ferase = false;
    var timer2;

    //var timer;

    document.oncontextmenu = function() {return false;};

    $(document).keydown(function(e) {

        if (e.which == 32) {
            if ($('#modalMsg3').css('visibility') != 'visible') {
                    $('#rotateBut').click();
                    return false;
            }
            return false;
        } else if ((brd.turboBuild)&&(!brd.turboBuild.dormant)) {
            var kp = 0;
            if (e.which == 39) kp = 1;     // right-east
            if (e.which == 40) kp = 2;     // down-south
            if (e.which == 37) kp = 3;     // left-west
            if (e.which == 38) kp = 4;     // up-north
            if (e.which == 8) kp = 5;      // backspace-undo
            if (kp) brd.turboBuild.move(kp);
        } else if (brd.multiSelect) {

            var rlo = brd.h, rhi = 0, clo = brd.w, chi = 0;
            for (var i = 0; i < brd.multiSelect.selectList.length; i++) {
                    if ((brd.multiSelect.selectList[i].row - 1) > rhi) rhi = (brd.multiSelect.selectList[i].row - 1);
                    if ((brd.multiSelect.selectList[i].row - 1) < rlo) rlo = (brd.multiSelect.selectList[i].row - 1);
                    if ((brd.multiSelect.selectList[i].col - 1) > chi) chi = (brd.multiSelect.selectList[i].col - 1);
                    if ((brd.multiSelect.selectList[i].col - 1) < clo) clo = (brd.multiSelect.selectList[i].col - 1);
```

```
        }

        if (e.which == 39) {                        // right-east
            var legit = true;
            for (i = 0; i < brd.multiSelect.selectList.length; i++) {
                if ((brd.multiSelect.selectList[i].col == brd.w)
                    || (brd.cell[brd.multiSelect.selectList[i].row-1][brd.multiSelect.selectList[i].col].type
                        && (!brd.multiSelect.isSelected(brd.cell[brd.multiSelect.selectList[i].row-
1][brd.multiSelect.selectList[i].col]))
                        )
                    ) {
                        legit = false;
                    }
            }
            if (legit) {

                var temp;
                for (var j = (chi+1); j >= clo; j--) {
                    for (i = 0; i < brd.multiSelect.selectList.length; i++) {
                        if  (brd.multiSelect.selectList[i].col == j) {
                            brd.cell[brd.multiSelect.selectList[i].row-
1][brd.multiSelect.selectList[i].col].plot(brd.multiSelect.selectList[i].type);
                            temp = brd.multiSelect.selectList[i];
                            brd.multiSelect.selectList[i] = brd.cell[brd.multiSelect.selectList[i].row-
1][brd.multiSelect.selectList[i].col];
                            temp.plot(0);
                        }
                    }
                }
                brd.draw();
                brd.drawSelect();
            }
            return false;
        }
        if (e.which == 40) {                        // down-south
            var legit = true;
            for (i = 0; i < brd.multiSelect.selectList.length; i++) {
                if ((brd.multiSelect.selectList[i].row == brd.h)
                    || (brd.cell[brd.multiSelect.selectList[i].row][brd.multiSelect.selectList[i].col-1].type &&
(!brd.multiSelect.isSelected(brd.cell[brd.multiSelect.selectList[i].row][brd.multiSelect.selectList[i].col -1])))) legit =
false;
            }
```

```
                if (legit) {
                    var temp;
                    for (var j = (rhi+1); j >= rlo; j--) {
                        for (i = 0; i < brd.multiSelect.selectList.length; i++) {
                            if  (brd.multiSelect.selectList[i].row == j) {
                                brd.cell[brd.multiSelect.selectList[i].row][brd.multiSelect.selectList[i].col-
1].plot(brd.multiSelect.selectList[i].type);
                                temp = brd.multiSelect.selectList[i];
                                brd.multiSelect.selectList[i] =
brd.cell[brd.multiSelect.selectList[i].row][brd.multiSelect.selectList[i].col-1];
                                temp.plot(0);
                            }
                        }
                    }
                    brd.draw();
                    brd.drawSelect();
                }

                return false;
            }
            if (e.which == 37) {                        // left-west
                var legit = true;
                for (i = 0; i < brd.multiSelect.selectList.length; i++) {
                    if ((brd.multiSelect.selectList[i].col == 1)
                        || (brd.cell[brd.multiSelect.selectList[i].row-1][brd.multiSelect.selectList[i].col-2].type &&
(!brd.multiSelect.isSelected(brd.cell[brd.multiSelect.selectList[i].row-1][brd.multiSelect.selectList[i].col -2])))) legit
= false;
                }
                if (legit) {
                    var temp;
                    for (var j = (clo); j <= (chi+1); j++) {
                        for (i = 0; i < brd.multiSelect.selectList.length; i++) {
                            if  (brd.multiSelect.selectList[i].col == j) {
                                brd.cell[brd.multiSelect.selectList[i].row-1][brd.multiSelect.selectList[i].col-
2].plot(brd.multiSelect.selectList[i].type);
                                temp = brd.multiSelect.selectList[i];
                                brd.multiSelect.selectList[i] = brd.cell[brd.multiSelect.selectList[i].row-
1][brd.multiSelect.selectList[i].col-2];
                                temp.plot(0);
                            }
                        }
                    }
                    brd.draw();
```

```
                    brd.drawSelect();
                }
                return false;
            }
            if (e.which == 38) {                        // up-north
                var legit = true;
                for (i = 0; i < brd.multiSelect.selectList.length; i++) {
                    if ((brd.multiSelect.selectList[i].row == 1)
                        || (brd.cell[brd.multiSelect.selectList[i].row-2][brd.multiSelect.selectList[i].col-1].type &&
(!brd.multiSelect.isSelected(brd.cell[brd.multiSelect.selectList[i].row-2][brd.multiSelect.selectList[i].col -1])))) legit
= false;
                }
                if (legit) {
                    var temp;
                    for (var j = (rlo); j <= (rhi+1); j++) {
                        for (i = 0; i < brd.multiSelect.selectList.length; i++) {
                            if  (brd.multiSelect.selectList[i].row == j) {
                                brd.cell[brd.multiSelect.selectList[i].row-2][brd.multiSelect.selectList[i].col-
1].plot(brd.multiSelect.selectList[i].type);
                                temp = brd.multiSelect.selectList[i];
                                brd.multiSelect.selectList[i] = brd.cell[brd.multiSelect.selectList[i].row-
2][brd.multiSelect.selectList[i].col-1];
                                temp.plot(0);
                            }
                        }
                    }
                    brd.draw();
                    brd.drawSelect();
                }
                return false;
            }
        }
    });


    $('canvas').click(function(e) {
        var cel = brd.hitXY(e);
                if ((!e.ctrlKey) && (cel == brd.selected)) {
                    if (mouseWasDown) mouseWasDown--;
                    else cel.rotateTile();

                } else {
```

```
if (e.ctrlKey && cel.type) {
    if (brd.selected) {
        if (!brd.multiSelect ) {
                brd.multiSelect = new SelectStruc();        // Turn MSelect on
                brd.setSelect(brd.selected);
                brd.setSelect(cel);
                brd.drawSelect();
                brd.selected = null;
        }
    } else {
        if (brd.multiSelect) {
                if (brd.multiSelect.isSelected(cel)) {
                    brd.multiSelect.remove(cel);
                    brd.draw();
                    brd.drawSelect(brd.multiSelect.selectList[(brd.multiSelect.numSelected-1)]);
                } else {
                    brd.setSelect(cel);                 // MSelect already on
                    brd.drawSelect();
                }
        }
    }
} else if (e.shiftKey && cel.type) {

    if (brd.selected)  {
        var hcol, hrow, lcol, lrow;

        if (cel.row > brd.selected.row) {hrow = cel.row; lrow = brd.selected.row;}
        if (cel.row < brd.selected.row) {lrow = cel.row; hrow = brd.selected.row;}
        if (cel.col > brd.selected.col) {hcol = cel.col; lcol = brd.selected.col;}
        if (cel.col < brd.selected.col) {lcol = cel.col; hcol = brd.selected.col;}

        if (lrow || lcol) {
                brd.multiSelect = new SelectStruc();        // Turn MSelect on
                brd.selected = null;
                var i, j;
                for (i = lrow; i <= hrow; i++) {
                    for (j = lcol; j <= hcol; j++) {
                            brd.setSelect(brd.cell[i-1][j-1]);
                            brd.drawSelect();
                    }
                }
                //brd.drawSelect();
        }
```

```
                    }

                } else {
                    if (brd.multiSelect) brd.multiSelect = null;
                    brd.setSelect(cel);
                    brd.drawSelect();
                }
            }

        mouseDown = 0;
    });


  $('canvas').mousedown(function(e){
        var cel = brd.hitXY(e);
        mouseDown = 1;

        if ( e.button == 2 ) {
            rightMouseDown = 1;
            if (brd.turboBuild && !brd.turboBuild.dormant) brd.turboBuild.move(5);
            else {
                $('canvas').addClass('eraserTip');
                cel.plot(0);
                brd.draw();
            }
        } else {
            var pickup = false;

            if (((cel.row > 1) && (brd.cell[cel.row-2][cel.col-1].type) && brd.cell[cel.row-2][cel.col-1].isAnEnd()) ||
                ((cel.row < (brd.h-1)) && (brd.cell[cel.row][cel.col-1].type) && brd.cell[cel.row][cel.col-1].isAnEnd())
||
                ((cel.col > 1) && (brd.cell[cel.row-1][cel.col-2].type) && brd.cell[cel.row-1][cel.col-2].isAnEnd()) ||
                ((cel.col < (brd.w-1)) && (brd.cell[cel.row-1][cel.col].type) && brd.cell[cel.row-1][cel.col].isAnEnd()))
pickup = true;

            if (!cel.type) {
                brd.turboBuild = new TurboBuild(brd, cel);
                if (pickup) brd.turboBuild.edgeFind(cel);
            }
        }
        return true;
    });
```

```
$('canvas').mousemove(function(e){

    if (rightMouseDown == 1) {
        var cel = brd.hitXY(e);
        $('canvas').addClass('eraserTip');
        if (cel.type) cel.plot(0);
        brd.draw();
    } else if (mouseDown) {
        var cel = brd.hitXY(e);
        var kp = 0;
        var dist = 1;
        if ( e.button == 2 ) {
            $('canvas').addClass('eraserTip');
            cel.plot(0);                                // right button / delete cell
        } else {
            if (brd.turboBuild && (!brd.turboBuild.dormant)) {        // left button / turboBuild

                if (cel.row == brd.turboBuild.nextMove.row) {
                    if (!(cel.col == brd.turboBuild.nextMove.col)) {
                        if (cel.col > brd.turboBuild.nextMove.col) {
                            kp = 1;
                            dist = (cel.col - brd.turboBuild.nextMove.col);
                        } else {
                            kp = 3;
                            dist = (brd.turboBuild.nextMove.col - cel.col);
                        }
                    }
                }
                if (cel.col == brd.turboBuild.nextMove.col) {
                    if (!(cel.row == brd.turboBuild.nextMove.row)) {
                        if (cel.row > brd.turboBuild.nextMove.row) {
                            kp = 2;
                            dist = (cel.row - brd.turboBuild.nextMove.row);
                        } else {
                            kp = 4;
                            dist = (brd.turboBuild.nextMove.row - cel.row);
                        }
                    }
                }
                var step;
```

```
                if ((brd.ss < 30) && brd.turboBuild.lastDir && kp && (kp != brd.turboBuild.lastDir)) {

                    var offset = 0;
                    if (brd.ss == 20) offset = 5;
                    if (brd.ss == 15) offset = 7;

                    if (((Math.ceil((e.pageX - $('canvas').offset().left+offset)/brd.ss) ==
brd.turboBuild.nextMove.col) || (Math.ceil((e.pageX - $('canvas').offset().left-offset)/brd.ss) ==
brd.turboBuild.nextMove.col))
                        && (cel.row == brd.turboBuild.nextMove.row)) kp = 0;
                    else if (((Math.ceil((e.pageY - $('canvas').offset().top+offset)/brd.ss) ==
brd.turboBuild.nextMove.row) || (Math.ceil((e.pageY - $('canvas').offset().top-offset)/brd.ss) ==
brd.turboBuild.nextMove.row))
                        && (cel.col == brd.turboBuild.nextMove.col)) kp = 0;
                }
                if (kp) brd.turboBuild.move(kp);
            }
        }
        return true;
    }
});


$('canvas').mouseup(function(e) {
    if (e.which == 3) {
        $('canvas').removeClass('eraserTip');
        rightMouseDown = 0;
    } else {
            if (mouseDown) {
                mouseDown = 0;
                mouseWasDown = 1;
            }
            if ((brd.turboBuild != null) && (!brd.turboBuild.dormant)) {
                brd.cell[brd.turboBuild.nextMove.row-1][brd.turboBuild.nextMove.col-1].clear();
                brd.turboBuild.dormant = true;
            }
    }
});


$('#colorBut').click(function(e){

    if (color) {
```

```
            color = false;
            $('#colorBut').removeClass('cbSelected');
        } else if (brd.regset.playing) {
            color = true;
            $('#colorBut').addClass('cbSelected');
        } else {
            var a = new Analysis(brd);
            if (!a.isValid()) alert("Invalid knot.");
            if (a.tricolorable) {
                    for(var ix in a.segments) a.segments[ix].drawColor();
                    brd.disPane.add("Structure tricolored.</br></br>");
            } else brd.disPane.add("Structure not tricolorable.</br></br>");
        }
    });


    $('#invBut').click(function(e){
        for (var i = -1; ++i < brd.h;) {
            for (var j = -1; ++j < brd.w;)  if (brd.cell[i][j].type > 6) {brd.cell[i][j].rotateTile(); brd.draw();
brd.drawSelect();}
        }
    });


    $('#shiftBut').click(function(e){
        var arrows = document.getElementById('shiftButtons');

        if (arrows.style.display == 'none') {
            arrows.style.display = 'block';
        } else {
            arrows.style.display = 'none';
        }
    });


    $('#rotateBut').click(function(e){
        if (brd.multiSelect == null) {
            if (brd.selected) brd.cell[brd.selected.row-1][brd.selected.col-1].rotateTile();
        } else {
            var index;
            for (index = 0; index < brd.multiSelect.numSelected; index++) {
                brd.cell[brd.multiSelect.selectList[index].row-1][brd.multiSelect.selectList[index].col-1].rotateTile();
            }
```

```
    }
    brd.draw();
    brd.drawSelect();
});


$('#eraseBut').click(function(e){

    if (!brd.multiSelect) {
        if (brd.selected) brd.cell[brd.selected.row-1][brd.selected.col-1].plot(0);
    } else {
        var index;
        for (index = 0; index < brd.multiSelect.numSelected; index++) {
            brd.cell[brd.multiSelect.selectList[index].row-1][brd.multiSelect.selectList[index].col-1].plot(0);
        }
        brd.multiSelect=null;
        brd.draw();
        brd.drawSelect();
    }
});


$('#clearBut').click(function(e){

    $('#modalMsg2').css({'visibility' : 'visible',
                                        'width' : '250px'});
    $('#Confirm2').css('left', '150px');

    document.getElementById('mmtext2').innerHTML = "Clear screen:<br>Are you sure?";

    $('div').each( function( ev) {
            if (!($( this ).is("#modalMsg2"))) $(this).addClass('disabledElement');
    });

    $('#Confirm2, #Cancel2').click(function () {
        $('#modalMsg2').css('visibility' , 'hidden');
        $('#modalMsg2').css('width' , '200px');
        $('#Confirm2').css('left', '100px');

        $('div').each( function( ev) {$(this).removeClass('disabledElement');});

        if (this.id == 'Confirm2') {
                var i, j;
```

```
                for (i=0; i < brd.h; i++) {
                    for (j=0; j < brd.w; j++) {
                        brd.cell[i][j].plot(0);
                    }
                }
                brd.draw();
                brd.disPane.add("Screen cleared.</br></br>");
        }
    });
});


$('#saveBut').click(function(e){

    var i;
    var toSave = false;

    for (i=0; i < brd.sequence.length; i++) {
        if (brd.sequence[i]) toSave = true;
    }

    if (toSave) {

            $('#modalMsg3').css('visibility' , 'visible');

            $('div').each( function( ev) {
                    if (!($( this ).is("#modalMsg3"))) $(this).addClass('disabledElement');
            });

            var fn = null;

            // Click handler for modal confirmation window
            var f = function(event) {
                $('#modalMsg3').css('visibility' , 'hidden');

                $('div').each( function(ev) {$(this).removeClass('disabledElement');});

                if (this.id == 'Confirm3') {
                        var re = new RegExp("[^a-zA-Z0-9 :]");
                        var str = document.getElementById("tb").value;

                        if (re.test(str) || (str.length > 16)) {
```

```
                    alert("Filename must be 1-16 characters long\nand contain only letters, numbers, and
underscores.");
                    $('#saveBut').click();
                } else fn = str;

                if (fn) {
                    var cp = {};
                    cp.size = brd.ss;
                    cp.seq = brd.sequence;

                    var st = JSON.stringify(cp);
                    var blob = new Blob([st], {type: 'text/plain'});

                    var sBut = document.getElementById('saveBut');

                    fn += ".txt"

                    if (window.navigator.msSaveOrOpenBlob) {
                        window.navigator.msSaveOrOpenBlob(blob, fn);
                        sBut.blur();
                        return;
                    }

                    var url = URL.createObjectURL(blob);

                    var a = document.createElement('a');
                    a.href = url;
                    a.download = fn;
                    a.textContent = '';
                    document.body.appendChild(a);
                    a.click();
                    sBut.blur();
                    a.parentNode.removeChild(a);

                    brd.disPane.add("'" + fn + "' saved to local disk.</br></br>");
                }
            }
        }

        // Assign click handler
        $('#Confirm3, #Cancel3').click(f);

        // Unbind click handler because it, under certain circumstances,
```

```
                // fires multiple times mysteriously.
                $('#Confirm3, #Cancel3').unbind().click(f);

        }
    });


    $('#loadBut').click(function(e){
        $('#fileOpen').click();
    });


    $('#fileOpen').change(function(e){

        var file = e.target.files[0];
        var reader = new FileReader();

        reader.onload = (function(theFile) {

                return function(ev) {

                    var cp =  JSON.parse(ev.target.result);

                    brd.ss = cp.size;
                    brd.sequence = cp.seq;

// temp duper
/*

var newone = {};
newone.size = brd.ss;

var newseq = cp.seq;

var temp = new Array();

var i;

for (i = 4; i < 8; i++) {
    temp[i] = new Array();
    newseq[i] = newseq[i-4];
    for (var x = 0; x < 50; x++) temp[i][x] = newseq[i][49-x];
}
```

```
for (i=4; i < 8; i++) {
        newseq[i] = temp[11-i];
}


newone.seq = newseq;


//---------------------------------------------------------------------


                                var st = JSON.stringify(newone);
                                var blob = new Blob([st], {type: 'text/plain'});

                                var sBut = document.getElementById('saveBut');

                                if (window.navigator.msSaveOrOpenBlob) {
                                    window.navigator.msSaveOrOpenBlob(blob, "moddedkws.txt");
                                    sBut.blur();
                                    return;
                                }

                                var url = URL.createObjectURL(blob);

                                var a = document.createElement('a');
                                a.href = url;
                                a.download = "moddedkws.txt";
                                a.textContent = '';
                                document.body.appendChild(a);
                                a.click();
                                sBut.blur();
                                a.parentNode.removeChild(a);

//temp duper end */


                        brd.w = canvas.width/brd.ss;
                        brd.h = canvas.height/brd.ss;
                        cv.clearRect(0, 0, canvas.width, canvas.height);

                        for (i=0; i < brd.h; i++) {
                                brd.cell[i] = new Array(brd.w);
```

```
                for (j=0; j < brd.w; j++) {
                        brd.cell[i][j] = new Cell(brd, i, j);
                }
            }
            brd.draw();

            // Set draggable tiles to new cell size
            $('.draggableTile').each( function(ev) {
                $(this).css('height', brd.ss+'px');
                $(this).css('width', brd.ss+'px');
            });

            // Display new cell size on size selection button
            document.getElementById('txt').innerHTML = (brd.ss + 'px');

            brd.regset.newSet();


        };
    })(file);

        reader.readAsText(file);
    });


    $('#resTogBut').click(function(e){


        $('#modalMsg').css({'visibility' : 'visible', 'width' : '310px'});
        $('#Confirm').css('left', '210px');

        document.getElementById('mmtext').innerHTML = "Changing cell size will erase<br>all unsaved work! Are
you sure?";

        $('div').each( function( ev) {
            if (!($( this ).is("#modalMsg"))) $(this).addClass('disabledElement');
        });


        // Click handler for modal confirmation window
        var f = function(event) {
            $('#modalMsg').css('visibility' , 'hidden');
            $('#modalMsg').css('width' , '200px');
```

```
    $('#Confirm').css('left', '100px');

    $('div').each( function(ev) {$(this).removeClass('disabledElement');});

    if (this.id == 'Confirm') {

        // Cycle through available resolutions
        if (brd.ss == 30) brd.ss = 20;
        else if (brd.ss == 20) brd.ss = 15;
        else if (brd.ss == 15) brd.ss = 30;

        for (var x=-1; ++x < 10;) brd.sequence[x] = null;

        cv.clearRect(0, 0, canvas.width, canvas.height);
        brd.w = canvas.width/brd.ss;
        brd.h = canvas.height/brd.ss;

        for (i=0; i < brd.h; i++) {
                brd.cell[i] = new Array(brd.w);
                for (j=0; j < brd.w; j++) {
                        brd.cell[i][j] = new Cell(brd, i, j);
                }
        }
        brd.draw();

        // Set draggable tiles to new cell size
        $('.draggableTile').each( function(ev) {
                $(this).css('height', brd.ss+'px');
                $(this).css('width', brd.ss+'px');
        });

        brd.disPane.add("Cell size changed to " + brd.ss + "px.</br></br>");

        // Display new cell size on size selection button
        document.getElementById('txt').innerHTML = (brd.ss + 'px');

        brd.regset.reset();
    }
}

me = 0;

// Assign click handler
```

```
        $('#Confirm, #Cancel').click(f);

        // Unbind click handler to offset 'bound handler accumulation'
        $('#Confirm, #Cancel').unbind().click(f);
    });


    var fch = function(event) {

        var d = document.getElementById(event.target.id);

        if (brd.regset.dowkerCheckActive  || brd.regset.playing) return false;
                // if we are in fact switching registers
                if ((! (d.classList.contains('active') || d.classList.contains('activeFull'))) && !ferase) {
                    // grab the register number from the element id
                    brd.regset.switchRegister(parseInt(d.id.substring(4)));
                    // make sure there's something there
                    if ( brd.sequence[brd.regset.activeBank]  &&
brd.sequence[brd.regset.activeBank][brd.regset.activePage]) {
                        // render away!
                        var i, j;
                        for (i=0; i < brd.h; i++) {
                            for (j=0; j < brd.w; j++) {
                                brd.cell[i][j].plot(0);
                            }
                        }
                        brd.draw();

                        var x;
                        for (x=0; x < (brd.sequence[brd.regset.activeBank][brd.regset.activePage].length); x++){
                            brd.cell[brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].r-
1][brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].c-
1].plot(brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].t);
                        }
                    }

                    mouseDown = 1;
                } else if (ferase) {
                    // d id div object
                    var ri = parseInt(d.id.substring(4));      // ri is the index


                    if (brd.sequence[brd.regset.activeBank] && brd.sequence[brd.regset.activeBank][ri]) {
```

```
                    var i, j;
                    for (i=0; i < brd.h; i++) {
                        for (j=0; j < brd.w; j++) {
                            brd.cell[i][j].plot(0);
                        }
                    }
                    brd.draw();

                    var x;
                    for (x=0; x < (brd.sequence[brd.regset.activeBank][ri].length); x++){
                        brd.cell[brd.sequence[brd.regset.activeBank][ri][x].r-
1][brd.sequence[brd.regset.activeBank][ri][x].c-1].plot(brd.sequence[brd.regset.activeBank][ri][x].t);
                    }



                    //---------------------------------------------------------------


                    $('#modalMsg').css({'visibility' : 'visible',
                                                'width' : '250px'});
                    $('#Confirm').css('left', '150px');

                    document.getElementById('mmtext').innerHTML = "Erase register " + (ri + 1) + ":<br>Are
you sure?";

                    $('div').each( function( ev) {
                            if (!($( this ).is("#modalMsg"))) $(this).addClass('disabledElement');
                    });

                    var fnctn = function () {
                        $('#modalMsg').css('visibility' , 'hidden');
                        $('#modalMsg').css('width' , '200px');
                        $('#Confirm').css('left', '100px');

                        $('div').each( function( ev) {
                                if (!($( this ).is("#modalMsg"))) $(this).removeClass('disabledElement');
                        });

                        if (this.id == 'Confirm') {
```

```
                    brd.disPane.add("Register " + (ri + 1) + " deleted.</br></br>");
                    brd.msgDisplay((("Register " + (ri + 1) + " deleted.")), 1500, 30, 300);
                    brd.sequence[brd.regset.activeBank][ri] = null;

                    var otherPageOn = false;
                    for (i = 0; i < 50; i++) {
                        if (brd.sequence[brd.regset.activeBank][i]) otherPageOn = true;
                    }
                    if (!otherPageOn) {             // No other pages active in bank, turn bank off
                            brd.sequence[brd.regset.activeBank] = null;
                    }

                    brd.regset.update();

                }

                for (i=0; i < brd.h; i++) {
                    for (j=0; j < brd.w; j++) {
                        brd.cell[i][j].plot(0);
                    }
                }
                brd.draw();

                for (x=0; x < (brd.sequence[brd.regset.activeBank][brd.regset.activePage].length);
x++){
                    brd.cell[brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].r-
1][brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].c-
1].plot(brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].t);
                }

            }

            $('#Confirm, #Cancel').click(fnctn);
            $('#Confirm, #Cancel').unbind().click(fnctn);

            ferase = false;


            //---------------------------------------------------------------------


        }
```

```
            }
    }


    var ff = function(e){
        if (e.which == 3){
            ferase = true;
            fch(e);
        }
    }


    $('.frameSel').click(fch);
    $('.frameSel').mousedown(ff);


    $('.bankSel').click(function(event) {

        if (brd.regset.dowkerCheckActive || brd.regset.playing) return false;

        var d = document.getElementById(event.target.id);

        if (! (d.classList.contains('bActive') || d.classList.contains('bActiveFull'))) {          // if we are in fact changing
banks

            if (brd.sequence[brd.regset.activeBank]  &&
brd.sequence[brd.regset.activeBank][brd.regset.activePage] )  {

                    var i, j;
                    for (i=0; i < brd.h; i++) {
                        for (j=0; j < brd.w; j++) {
                            brd.cell[i][j].plot(0);
                        }
                    }
                    brd.draw();

                    var x;
                    for (x=0; x < (brd.sequence[brd.regset.activeBank][brd.regset.activePage].length); x++){
                        brd.cell[brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].r-
1][brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].c-
1].plot(brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].t);
                    }
```

```
        }

        mouseDown = 1;
        brd.regset.switchBank(parseInt(d.id.substring(5)));
    }
});


$(document).on("click", ".active,.activeFull", function(event){

    if (brd.regset.dowkerCheckActive) return false;
    if (brd.regset.dowkerCheckWasActive) {
        brd.regset.dowkerCheckWasActive = 0;
        return false;
    }

    if (mouseDown) mouseDown = 0;
    else {

        var capture = new Array();
        var square = {};
        var x, y;

        for (x = 0; x < brd.h; x++) {                    // Attempt screen capture
            for (y = 0; y < brd.w; y++) {
                if (brd.cell[x][y].type) {
                    square = {
                        r: (x+1),
                        c: (y+1),
                        t: brd.cell[x][y].type};
                    capture.push(square);
                }
            }
        }

        if (capture[0]) {              // if there's anything on the screen

            brd.disPane.add("Page saved in register " + (brd.regset.activePage + 1) + ".</br></br>");
            brd.msgDisplay(("Page saved in register " + (brd.regset.activePage + 1) + "."), 1500, 30, 300);

            if (!brd.sequence[brd.regset.activeBank]) {
                brd.sequence[brd.regset.activeBank] = new Array();
                for (i = 0; i < 50; i++)  brd.sequence[brd.regset.activeBank][i] = null;
```

```
            }

            brd.sequence[brd.regset.activeBank][brd.regset.activePage] = capture;

        } else if ($("#div-" + brd.regset.activePage).hasClass('on')) {                                    // ERASE!
            brd.disPane.add("Register " + (brd.regset.activePage + 1) + " deleted.</br></br>");
            brd.msgDisplay((("Register " + (brd.regset.activePage + 1) + " deleted.")), 1500, 30, 300);
            brd.sequence[brd.regset.activeBank][brd.regset.activePage] = null;

            var otherPageOn = false;
            for (i = 0; i < 50; i++) {
                if (brd.sequence[brd.regset.activeBank][i]) otherPageOn = true;
            }
            if (!otherPageOn) {             // No other pages active in bank, turn bank off

                    brd.sequence[brd.regset.activeBank] = null;
            }
        }

        brd.regset.update();
    }
});


$('#prevBut').click(function(e) {

    if (brd.sequence[brd.regset.activeBank]) {
            var i = brd.regset.activePage;
            var count = 0;
            i--;

            while(!brd.sequence[brd.regset.activeBank][i] && count < 51) {
                    if (i < 1) {                                        // if we are on the last register
                            var x;
                            var totalBanks = 0;
                            var nextActiveBank;
                            for (x = 0; x < 10; x++) if (brd.sequence[x]) totalBanks++;
                            //  if multibank on, and there's another bank to go to
                            if (brd.regset.playing && (!$('#localBox').is(":checked")) && totalBanks > 1)  {
                                var y = brd.regset.activeBank - 1;              // then switch to next occupied bank
                                if (y == -1) y = 9;
                                while (y != brd.regset.activeBank && (!brd.sequence[y])) {
```

```
                        y--; if (y == -1) y = 9;      // Find next occupied bank
                    }

                    brd.regset.switchBank(y);                    // and go there

                    i = 49;                                      // Set local register
                    count = 0;              // reset 'count', as search for a populated register will continue
                } else i = 49;
            } else i--;
            count++;
        }

        if (count < 50) {
            brd.regset.switchRegister(i);

                var i, j;
                for (i=0; i < brd.h; i++) {
                    for (j=0; j < brd.w; j++) {
                        brd.cell[i][j].plot(0);
                    }
                }
                brd.draw();

                var x;
                for (x=0; x < (brd.sequence[brd.regset.activeBank][brd.regset.activePage].length); x++){
                    brd.cell[brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].r-
1][brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].c-
1].plot(brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].t);
                }

            if (color) new Analysis(brd);
        }
    }
});


    $('#nextBut').click(function(e) {

        if (brd.sequence[brd.regset.activeBank]) {
            var i = brd.regset.activePage;
            var count = 0;
            i++;
```

```
while(!brd.sequence[brd.regset.activeBank][i] && count < 51) {
        if (i > 48) {                                   // if we are on the last register
                var x;
                var totalBanks = 0;
                var nextActiveBank;
                for (x = 0; x < 10; x++) if (brd.sequence[x]) totalBanks++;

                //  if multibank on, and there's another bank to go to
                if (brd.regset.playing && (!$('#localBox').is(":checked")) && totalBanks > 1)  {
                    var y = brd.regset.activeBank + 1;          // then switch to next occupied bank
                    if (y == 10) y = 0;
                    while (y != brd.regset.activeBank && (!brd.sequence[y])){
                        y++; if (y == 10) y = 0;                // Find next occupied bank
                    }

                    brd.regset.switchBank(y);    //  .. and go there

                    i = 0;                                      // Set local register to '0'
                    count = 0;              // reset 'count', as search for a populated register will continue

                } else i = 0;
        } else i++;
        count++;
}

if (count < 50) {
        brd.regset.switchRegister(i);

        var i, j;
        for (i=0; i < brd.h; i++) {
                for (j=0; j < brd.w; j++) {
                        brd.cell[i][j].plot(0);
                }
        }
        brd.draw();

        var x;
        for (x=0; x < (brd.sequence[brd.regset.activeBank][brd.regset.activePage].length); x++){
                brd.cell[brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].r-
1][brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].c-
1].plot(brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].t);
        }
```

```
                if (color) new Analysis(brd);
            }
        }
    });


    $('#playBut, #rvsPlayBut').click(function(e) {

        if (!(((e.target.id == 'playBut') && (brd.regset.playing == 1))||((e.target.id == 'rvsPlayBut') &&
(brd.regset.playing == -1)))) {

            rate = (10000/$('#slider').val());
            var prv = brd.regset.playing;

            if ((e.target.id == 'playBut') && (brd.regset.playing != 1)) {
                brd.regset.playing = 1;

                clearInterval(timer);
                timer = setInterval(function() {
                    document.getElementById("nextBut").click();
                    }, rate);

                if (prv == -1) $('#rvsPlayBut').removeClass('revPlayActive');
                $('#playBut').addClass('playActive');

            } else if ((e.target.id == 'rvsPlayBut') && (brd.regset.playing != -1)) {
                brd.regset.playing = -1;

                clearInterval(timer);
                timer = setInterval(function() {
                    document.getElementById("prevBut").click();
                    }, rate);

                if (prv == 1) $('#playBut').removeClass('playActive');
                $('#rvsPlayBut').addClass('revPlayActive');
            }

            multiBank = 1;

        }
        $('#playBut').removeClass('pHover');
    });
```

```
    $('#slider').change(function(e) {

        rate = (10000/$('#slider').val());

        if (brd.regset.playing) {
                clearInterval(timer);
                timer = setInterval(function() {
                    document.getElementById('nextBut').click();
                }, rate);
        }

    });


    $('#dCompare').click(function(e) {


            var clk = 0;
            var regs = new Array();
            var dowks = new Array();

            var frameClick = function(event) {

                var d = document.getElementById(event.target.id);

                if  (brd.sequence[brd.regset.activeBank][parseInt(d.id.substring(4))]) {          // if register is
populated

                        if ( clk == 0) {
                                regs[0] = parseInt(d.id.substring(4));

                                var j, k;
                                for (j=0; j < brd.h; j++) {
                                        for (k=0; k < brd.w; k++) {
                                                brd.cell[j][k].plot(0);
                                        }
                                }

                                var x;
                                for (x=0; x < (brd.sequence[brd.regset.activeBank][regs[0]].length); x++){
```

```
                        brd.cell[brd.sequence[brd.regset.activeBank][regs[0]][x].r-
1][brd.sequence[brd.regset.activeBank][regs[0]][x].c-1].plot(brd.sequence[brd.regset.activeBank][regs[0]][x].t);
                    }

                    var a = new Analysis(brd);
                    dowks[0] = a.extractDowkers();

                    for (j=0; j < brd.h; j++) {
                        for (k=0; k < brd.w; k++) {
                            brd.cell[j][k].plot(0);
                        }
                    }

                    for (x=0; x < (brd.sequence[brd.regset.activeBank][brd.regset.activePage].length);
x++){
                        brd.cell[brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].r-
1][brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].c-
1].plot(brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].t);
                    }

                    brd.draw();

                    if (brd.strucCreated) {
                        $('#div-' + regs[0]).addClass('noBlink');
                        $('#div-' + regs[0]).removeClass('off');
                        $('#div-' + regs[0]).addClass('on');
                        clk++;
                        brd.disPane.add("Register " + regs[0] + " dowker numbers:</br>");
                        brd.disPane.add(dowks[0] + "</br></br>");
                    } else {brd.msgDisplay("Not a valid structure.", 500, 30, 300); dowks[0]=null;}

                } else {
                    if (parseInt(d.id.substring(4)) != regs[0]) {
                        regs[1] = parseInt(d.id.substring(4));

                        var j, k;
                        for (j=0; j < brd.h; j++) {
                            for (k=0; k < brd.w; k++) {
                                brd.cell[j][k].plot(0);
                            }
                        }

                        var x;
```

```
                                        for (x=0; x < (brd.sequence[brd.regset.activeBank][regs[1]].length); x++){
                                                brd.cell[brd.sequence[brd.regset.activeBank][regs[1]][x].r-
1][brd.sequence[brd.regset.activeBank][regs[1]][x].c-1].plot(brd.sequence[brd.regset.activeBank][regs[1]][x].t);
                                        }

                                        var a = new Analysis(brd);
                                        dowks[1] = a.extractDowkers();

                                        for (j=0; j < brd.h; j++) {
                                                for (k=0; k < brd.w; k++) {
                                                        brd.cell[j][k].plot(0);
                                                }
                                        }

                                        for (x=0; x <
(brd.sequence[brd.regset.activeBank][brd.regset.activePage].length); x++){

    brd.cell[brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].r-
1][brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].c-
1].plot(brd.sequence[brd.regset.activeBank][brd.regset.activePage][x].t);
                                        }

                                        brd.draw();

                                        if (brd.strucCreated) {
                                                dowks[1] = a.extractDowkers();
                                                $('#div-' + regs[1]).addClass('noBlink');
                                                $('#div-' + regs[1]).removeClass('off');
                                                $('#div-' + regs[1]).addClass('on');

                                                clk = 0;
                                                clearInterval(timer2);
                                                document.getElementById('txt2').innerHTML = "OFF";
                                                $('#dCompare').css('background', 'red');
                                                $('.noBlink').removeClass('noBlink');
                                                brd.regset.update();

                                                brd.disPane.add("Register " + regs[1] + " dowker numbers:</br>");
                                                brd.disPane.add(dowks[1] + "</br></br>");


                                                var match = true;
```

```
                                        if (dowks[0].length == dowks[1].length) {
                                                for (var x = 0; x < dowks[0]; x++) {
                                                    if (dowks[0][x] != dowks[1][x]) match = false;
                                                }
                                        } else match = false;

                                        var msg = "Registers " + (regs[0]+1) + " and " + (regs[1]+1) +(match? "
are a" : " do not") + " match.";
                                        brd.msgDisplay(msg, 2000, 30, 350);

                                        $('.frameSel').unbind('click', frameClick);
                                        brd.regset.dowkerCheckActive = 0;
                                        if (regs[1] == brd.regset.activePage) brd.regset.dowkerCheckWasActive
= 1;


                                    } else {brd.msgDisplay("Not a valid structure.", 500, 30, 300); dowks[1] = null;}


                                } else {
                                        $('#div-' + parseInt(d.id.substring(4))).removeClass('noBlink');
                                        regs[0] = null;
                                        clk = 0;
                                }
                        }
                    }
                }

                if (brd.regset.dowkerCheckActive) {

                        document.getElementById('txt2').innerHTML = "OFF";
                        $('#dCompare').css('background', 'red');

                        clearInterval(timer2);
                        $('.noBlink').removeClass('noBlink');
                        brd.regset.dowkerCheckActive = 0;
                        $('.frameSel').unbind('click', frameClick);
                        brd.regset.update();

                } else {

                        var count = 0;
                        if (brd.sequence[brd.regset.activeBank]) {
```

```
        for (var x = 0; x < 50; x++) {
            if  (brd.sequence[brd.regset.activeBank][x]) count++;
        }
}

if ((!brd.regset.playing) && (count > 1)) {

    document.getElementById('txt2').innerHTML = "ON";
    $('#dCompare').css('background', '#00FF00');

    clearInterval(timer2);
    var i = 0;

    brd.regset.dowkerCheckActive = 1;

    var blinkRegs = function() {

        i++;

        if (i%2 == 1) {

            $('.on').addClass('off').removeClass('.on');
            $('.noBlink').removeClass('off').addClass('.on');
            $('.activeFull').addClass('off').removeClass('activeFull');


        } else {
            for (var x = 0; x < 50; x++) {
                if (brd.sequence[brd.regset.activeBank][x]) {
                        $('#div-' + x).addClass('on').removeClass('off');
                }
            }
        }
    }

    timer2 = setInterval(blinkRegs, 300);

    $('.frameSel').bind('click', frameClick);
    $('.frameSel').unbind().click(frameClick);
    $('.frameSel').bind('click', fch);
    $('.frameSel').bind('mousedown', ff);

}
```

```
            }
        return false;
    });


    $('#pauseBut').click(function(e) {
        clearInterval(timer);
        $('#playBut').removeClass('playActive');
    });


    $('#stopBut').click(function(e) {
        clearInterval(timer);
        $('#playBut').removeClass('playActive');
        $('#rvsPlayBut').removeClass('revPlayActive');
        $('#colorBut').removeClass('cbSelected');
        brd.regset.playing = 0;
        color = false;
    });


    $('.seqContButs').mouseenter(function(e) {
$(e.target).addClass('pHover');}).mouseleave(function(e){$(e.target).removeClass('pHover');});


    $('canvas').mouseleave(function(e){

        if ((e.clientX < $('canvas').offset().left) ||
            (e.clientX > (($('canvas').offset().left) + ($('canvas').width()))) ||
            (e.clientY < $('canvas').offset().top) ||
            (e.clientY > (($('canvas').offset().top) + ($('canvas').height())))) {
                var arrows = document.getElementById('shiftButtons');
                arrows.style.display = 'none';

                if ((brd.turboBuild != null) && (!brd.turboBuild.dormant)) {
                    brd.cell[brd.turboBuild.nextMove.row-1][brd.turboBuild.nextMove.col-1].plot(0);
                    brd.turboBuild.dormant = true;
                }
        }
    });


$(window).scroll(function(){
```

```
        $('#buttonTray').css('left', $('#buttonTray').css('left')-$(window).scrollLeft());
        $('#tileMenu').css('left', $('#tileMenu').css('left')-$(window).scrollLeft());
        $('#resTogButton').css('left', $('resTogButton').css('left')-$(window).scrollLeft());
        $('#text-pane').css('left', $('text-pane').css('left')-$(window).scrollLeft());
});


$('#closePane').click(function(e) {
    $('#text-pane').css('visibility', 'hidden');
    $('#closePane').removeClass('opened');
    $('#closePane').addClass('closed');
    brd.disPane.opened = 0;
});

$('#openPane').click(function(e) {
    $('#text-pane').css('visibility', 'visible');
    $('#closePane').addClass('opened');
    $('#closePane').removeClass('closed');
    brd.disPane.opened = 1;
});

$('#sizePane').click(function(e) {
    brd.disPane.changeSize();
});


$('#text-pane').draggable();

$('#shiftButtons').click(function(e){
        var idx = e.target.id;
        if (!brd.multiSelect) {
            brd.shift(idx);
        } else {
            var kp = jQuery.Event("keydown");
            switch (e.target.id) {
                case "upShift"    :   kp.which = 38; break;
                case "downShift"  :   kp.which =  40; break;
                case "leftShift"    :   kp.which =  37; break;
                case "rightShift"   :   kp.which =  39;
            }

            $("input").trigger(kp);
            return false;
```

```
        }
});

$('.draggableTile').draggable({helper: 'clone',
                        opacity: 1,
                        stack: '.draggableTile'
});

$('canvas').droppable({accept: '.draggableTile',
                            tolerance: 'pointer',
                            drop: function(event, ui) {
                                    var loc = brd.hitXY(event);
                                    var d = $(ui.draggable).attr("id");
                                    if ((d == "tileCornerTopRight") || (d == "tile-1")) {
                                        loc.plot(1);
                                    }
                                        if (d == "tile-2") loc.plot(2);
                                        if (d == "tile-3") loc.plot(3);
                                        if (d == "tile-4") loc.plot(4);
                                        if ((d == "tileLineVert") || (d == "tile-5")) {
                                            loc.plot(5);
                                        }
                                        if (d == "tile-6") loc.plot(6);
                                        if ((d == "tileCrossOver")|| (d == "tile-7")) {
                                            loc.plot(7);
                                        }
                                        if (d == "tile-8") loc.plot(8);

                                        brd.setSelect(loc);
                                        brd.drawSelect();
                            }
    });
});
```

B7 AnalysisAndSuperClasses.js

```
//-----------------------------------------------
//
//  Analysis Class
//  An instance of the Analysis class is instantiated
//  by the user after constructing a knot. It runs a
//  couple checks for validity,
//
//-----------------------------------------------


function Analysis(b){
    this.board = b;
    this.NWCorner = null;
    this.crossingNum = null;
    this.crossings = new Array(b.h * b.w);
    this.segments = new Array(b.h * b.w);
    this.tricolorable = false;
    this.unknot = false;

    brd.strucCreated = false;

    if (this.isValid()) {
        this.NWCorner = this.findFirstCorner();
        this.findCrossings();
        if (this.crossingNum){
            this.connectCells(this.board);
            this.traverse(this.NWCorner);
            brd.strucCreated = true;
            if (brd.regset.playing && this.tricolorable) {
                for (var ix in this.segments) this.segments[ix].drawColor();
            }
        }
    }
}

Analysis.prototype.findCrossIndex = function(c){

    var x = 0;
    var cr = null;
    while (x < this.crossingNum){
        if (this.crossings[x].isMe(c)) {cr = x;}
        x++;
    }
    return cr;
}
```

```
Analysis.prototype.isValid = function() {
    var valid = true;
    for (i=0; i < this.board.h; i++) {
        for (j=0; j < this.board.w; j++) {
            if (this.board.cell[i][j].type) {
                if (this.board.cell[i][j].tile.north) {
                    if (i == 0 || !this.board.cell[i-1][j].type) valid = false;
                    else if (this.board.cell[i-1][j].tile.south == 0)  valid = false;
                }
                if (this.board.cell[i][j].tile.east) {
                    if (j == (this.board.w - 1) || !this.board.cell[i][j+1].type) valid = false;
                    else if (this.board.cell[i][j+1].tile.west == 0) valid = false;
                }
                if (this.board.cell[i][j].tile.south) {
                    if (i == (this.board.h - 1) || !this.board.cell[i+1][j].type) valid = false;
                    else if (this.board.cell[i+1][j].tile.north == 0) valid = false;
                }
                if (this.board.cell[i][j].tile.west) {
                    if (j == 0 || !this.board.cell[i][j-1].type) valid = false;
                    else if (this.board.cell[i][j-1].tile.east == 0) valid = false;
                }
            }
        }
    }
    return valid;
}

Analysis.prototype.findFirstCorner = function() {
    var c = null;
    for (i = 0; (!c && (i < brd.h)); i++){
        for (j = 0; (!c && (j < brd.w)); j++) {
            if (brd.cell[i][j].type) c = brd.cell[i][j];
        }
    }
    return c;
}

Analysis.prototype.findCrossings = function() {
    var count = 0;
    for (i = 0; i < this.board.h; i++){
        for (j = 0; j < this.board.w; j++) {
            if (this.board.cell[i][j].type == 7 || this.board.cell[i][j].type == 8) {
                this.crossings[count] = new SuperCrossing(this.board.cell[i][j]);
                count++;
            }
        }
    }
    this.crossingNum = count;
}
```

```
Analysis.prototype.connectCells = function(b) {
    for (i = 0; i < b.h; i++){
        for (j = 0; j < b.w; j++) {
            var c = b.cell[i][j];
            var t = c.type;
            if (t == 0) {
                c.nConnect =
                c.eConnect =
                c.wConnect =
                c.sConnect = null;
            } else if (t < 5) {
                if (t == 1) {
                    c.nConnect = null;
                    c.eConnect = null;
                    c.wConnect = b.cell[i][j-1];
                    c.sConnect = b.cell[i+1][j];
                } else if (t == 2) {
                    c.nConnect = b.cell[i-1][j];
                    c.eConnect = null;
                    c.wConnect = b.cell[i][j-1];
                    c.sConnect = null;
                } else if (t == 3) {
                    c.nConnect = b.cell[i-1][j];
                    c.eConnect = b.cell[i][j+1];
                    c.wConnect = null;
                    c.sConnect = null;
                } else {
                    c.nConnect = null;
                    c.eConnect = b.cell[i][j+1];
                    c.wConnect = null;
                    c.sConnect = b.cell[i+1][j];
                }
            } else if (t < 7) {
                if (t == 5) {
                    c.nConnect = b.cell[i-1][j];
                    c.eConnect = null;
                    c.wConnect = null;
                    c.sConnect = b.cell[i+1][j];
                } else if (t == 6) {
                    c.nConnect = null;
                    c.eConnect = b.cell[i][j+1];
                    c.wConnect = b.cell[i][j-1];
                    c.sConnect = null;
                }
            } else if (t < 9) {
                c.nConnect = b.cell[i-1][j];
                c.eConnect = b.cell[i][j+1];
                c.wConnect = b.cell[i][j-1];
                c.sConnect = b.cell[i+1][j];
```

```
            }
        }
    }
}

Analysis.prototype.extractDowkers = function() {
    var n, x = 0;
    var st = "";
    while (x < this.crossingNum) {
        n = (x*2)+1;
        for (i = 0; i <= (this.crossingNum-1); i++) {
            if (this.crossings[i].oddDowker == n) {
                st += (this.crossings[i].evenDowker + "    ");
            }
        }
        x++;
    }
    return st;
}

Analysis.prototype.traverse = function(c) {
    var currentSeg = 0;
    var dowkers = 1;
    var start = null;
    var newCross;
    var lastCross;
    var current = c;
    var next = c.sConnect;
    while (!( (next.type == 7 && (next.eConnect == current || next.wConnect == current) ) ||
            (next.type == 8 && (next.nConnect == current || next.sConnect == current) ) )) {

        switch (next.type) {
            case 1: if (next.wConnect == current) {
                        current = next;
                        next = current.sConnect;
                    } else {
                        current = next;
                        next = current.wConnect;
                    }
                    break;
            case 2: if (next.wConnect == current) {current = next;
                                                   next = current.nConnect;}
                        else {current = next; next = current.wConnect;} break;
            case 3: if (next.nConnect == current) {current = next;
                                                   next = current.eConnect;}
                        else {current = next; next = current.nConnect;} break;
            case 4: if (next.sConnect == current) {current = next;
                                                   next = current.eConnect;}
                        else {current = next; next = current.sConnect;} break;
            case 5: if (next.sConnect == current) {current = next;
```

```
                                        next = current.nConnect;}
                    else {current = next; next = current.sConnect;} break;
        case 6: if (next.eConnect == current) {current = next;
                                                next = current.wConnect;}
                    else {current = next; next = current.eConnect;} break;
        case 7:
        case 8: //alert ("hit crossing");
                if (next.eConnect == current) {current = next;
                                        next = current.wConnect;}
                    else if (next.nConnect == current){current = next;
                                        next = current.sConnect;}
                    else if (next.wConnect == current){current = next;
                                        next = current.eConnect;}
                    else {current = next;
                            next = current.nConnect;}
    }
}
var temp = current;
current = next;
next = temp;
start = current;
newCross = this.findCrossIndex(this.board.cell[start.row-1][start.col-1]);
this.crossings[newCross].placeDowker(dowkers);
dowkers++;

this.segments[currentSeg] = new Segment(this.crossings[newCross], currentSeg);
if ((next.wConnect == current)||(next.nConnect == current)) {
    this.crossings[newCross].seSeg = this.segments[currentSeg];
} else {    // assign '0' to first segment of super-crossing
    this.crossings[newCross].nwSeg = this.segments[currentSeg];
}

while (currentSeg < this.crossingNum) {

    // Add inside tiles (superlines/corners) to segment, increment
    var sg = this.segments[currentSeg].numElements - 1;
    if (next.type < 5) {
        this.segments[currentSeg].insideTiles[sg] = new SuperCorner(next, currentSeg);
        this.segments[currentSeg].numElements++;
    } else if (next.type < 7) {
        this.segments[currentSeg].insideTiles[sg] = new SuperLine(next, currentSeg);
        this.segments[currentSeg].numElements++;
    }

    switch (next.type) {
        case 1: if (next.wConnect == current) {current = next;
                                        next = current.sConnect;}
                    else {current = next; next = current.wConnect;} break;
        case 2: if (next.wConnect == current) {current = next;
                                        next = current.nConnect;}
```

```
            else {current = next; next = current.wConnect;} break;
case 3: if (next.nConnect == current) {current = next;
                                       next = current.eConnect;}
            else {current = next; next = current.nConnect;} break;
case 4: if (next.sConnect == current) {current = next;
                                       next = current.eConnect;}
            else {current = next; next = current.sConnect;} break;
case 5: if (next.sConnect == current) {current = next;
                                       next = current.nConnect;}
            else {current = next; next = current.sConnect;} break;
case 6: if (next.eConnect == current) {current = next;
                                           next = current.wConnect;}
            else {current = next; next = current.eConnect;} break;
case 7:
case 8: {
    var over = 0;
    lastCross = newCross;
    newCross = this.findCrossIndex(this.board.cell[next.row-1][next.col-1]);

    //if we got here through a n or w segment
    if (this.crossings[lastCross].nwSeg == currentSeg) {
        if (this.crossings[lastCross].type == 7) {
            // if it's an overcrossing than set west
            this.crossings[lastCross].westDest = this.crossings[newCross];
        } else {
            // if undercrossing than set north
            this.crossings[lastCross].northDest = this.crossings[newCross];
        }
    } else if (this.crossings[lastCross].seSeg == currentSeg) {
        if (this.crossings[lastCross].type == 7) {
            this.crossings[lastCross].eastDest = this.crossings[newCross];
        } else {
            this.crossings[lastCross].southDest = this.crossings[newCross];
        }
    }

    //segs
    if (next.eConnect == current) {
        this.crossings[newCross].eastDest = this.crossings[lastCross];
        this.crossings[newCross].westDest = 1;
        current = next;
        next = current.wConnect;          // step thru east to west
        if (current.type == 7) {
            // if overcrossing increment segment
            this.crossings[newCross].seSeg = this.segments[currentSeg];
            this.segments[currentSeg].upperBound = this.crossings[newCross];
            this.segments[currentSeg].numElements++;
            lo = this.segments[currentSeg].lowerBound;
            hi = this.segments[currentSeg].upperBound;
            for (i = 0; i < (this.segments[currentSeg].numElements - 2); i++) {
```

```
            if (this.segments[currentSeg].insideTiles[i].type > 6) {
                if (this.segments[currentSeg].insideTiles[i].type == 7) {
                    if (this.segments[currentSeg].insideTiles[i].northDest == lo) {
                        this.segments[currentSeg].insideTiles[i].southDest = hi;
                    } else {
                        this.segments[currentSeg].insideTiles[i].northDest = hi;
                    }
                } else {
                    if (this.segments[currentSeg].insideTiles[i].eastDest == lo) {
                        this.segments[currentSeg].insideTiles[i].westDest = hi;
                    } else {
                        this.segments[currentSeg].insideTiles[i].eastDest = hi;
                    }
                }
            }
        }
        currentSeg++;
        if (currentSeg < this.crossingNum) {   // init undiscovered segment
            this.segments[currentSeg] = new Segment(this.crossings[newCross], currentSeg);
            this.crossings[newCross].nwSeg = this.segments[currentSeg];
        }
    } else {
        over = 1;
        this.crossings[newCross].overSeg = this.segments[currentSeg];
        this.crossings[newCross].eastDest = this.segments[currentSeg].lowerBound;
        this.segments[currentSeg].insideTiles[sg] = this.crossings[newCross];
        this.segments[currentSeg].numElements++;
    }
} else if (next.nConnect == current) {
    this.crossings[newCross].northDest = this.crossings[lastCross];
    this.crossings[newCross].southDest = 1;
    current = next;
    next = current.sConnect;          // step thru north to south
    if (current.type == 8) {
        // if undercrossing increment segment
        this.crossings[newCross].nwSeg = this.segments[currentSeg];
        this.segments[currentSeg].upperBound = this.crossings[newCross];
        this.segments[currentSeg].numElements++;
        lo = this.segments[currentSeg].lowerBound;
        hi = this.segments[currentSeg].upperBound;
        for (i = 0; i < (this.segments[currentSeg].numElements - 2); i++) {
            if (this.segments[currentSeg].insideTiles[i].type > 6) {
                if (this.segments[currentSeg].insideTiles[i].type == 7) {
                    if (this.segments[currentSeg].insideTiles[i].northDest == lo) {
                        this.segments[currentSeg].insideTiles[i].southDest = hi;
                    } else {
                        this.segments[currentSeg].insideTiles[i].northDest = hi;
                    }
                } else {
                    if (this.segments[currentSeg].insideTiles[i].eastDest == lo) {
```

```
                        this.segments[currentSeg].insideTiles[i].westDest = hi;
                    } else {
                        this.segments[currentSeg].insideTiles[i].eastDest = hi;
                    }
                }
            }
        }
        currentSeg++;
        if (currentSeg < this.crossingNum) {     // init undiscovered segment
            this.segments[currentSeg] = new Segment(this.crossings[newCross], currentSeg);
            this.crossings[newCross].seSeg = this.segments[currentSeg];
        }
    } else {
        over = 1;
        this.crossings[newCross].overSeg = this.segments[currentSeg];
        this.crossings[newCross].northDest = lo;
        this.segments[currentSeg].insideTiles[sg] = this.crossings[newCross];
        this.segments[currentSeg].numElements++;
    }
} else if (next.wConnect == current) {
    this.crossings[newCross].westDest = this.crossings[lastCross];
    this.crossings[newCross].eastDest = 1;
    current = next;
    next = current.eConnect;          // step thru west to east
    if (current.type == 7) {
        // if overcrossing increment segment
        this.crossings[newCross].nwSeg = this.segments[currentSeg];
        this.segments[currentSeg].upperBound = this.crossings[newCross];
        this.segments[currentSeg].numElements++;
        lo = this.segments[currentSeg].lowerBound;
        hi = this.segments[currentSeg].upperBound;
        for (i = 0; i < (this.segments[currentSeg].numElements - 2); i++) {
            if (this.segments[currentSeg].insideTiles[i].type > 6) {
                if (this.segments[currentSeg].insideTiles[i].type == 7) {
                    if (this.segments[currentSeg].insideTiles[i].northDest == lo) {
                        this.segments[currentSeg].insideTiles[i].southDest = hi;
                    } else {
                        this.segments[currentSeg].insideTiles[i].northDest = hi;
                    }
                } else {
                    if (this.segments[currentSeg].insideTiles[i].eastDest == lo) {
                        this.segments[currentSeg].insideTiles[i].westDest = hi;
                    } else {
                        this.segments[currentSeg].insideTiles[i].eastDest = hi;
                    }
                }
            }
        }
        currentSeg++;
        if (currentSeg < this.crossingNum) {     // init undiscovered segment
```

```
            this.segments[currentSeg] = new Segment(this.crossings[newCross], currentSeg);
            this.crossings[newCross].seSeg = this.segments[currentSeg];
        }
    } else {
        over = 1;
        this.crossings[newCross].overSeg = this.segments[currentSeg];
        this.crossings[newCross].westDest = this.segments[currentSeg].lowerBound;
        this.segments[currentSeg].insideTiles[sg] = this.crossings[newCross];
        this.segments[currentSeg].numElements++;
    }
} else {
    this.crossings[newCross].southDest = this.crossings[lastCross];
    this.crossings[newCross].eastDest = 1;
    current = next;
    next = current.nConnect;    // step thru south to north
    if (current.type == 8) {
        // if undercrossing increment segment
        this.crossings[newCross].seSeg = this.segments[currentSeg];
        this.segments[currentSeg].upperBound = this.crossings[newCross];
        this.segments[currentSeg].numElements++;
        lo = this.segments[currentSeg].lowerBound;
        hi = this.segments[currentSeg].upperBound;
        for (i = 0; i < (this.segments[currentSeg].numElements - 2); i++) {
            if (this.segments[currentSeg].insideTiles[i].type > 6) {
                if (this.segments[currentSeg].insideTiles[i].type == 7) {
                    if (this.segments[currentSeg].insideTiles[i].northDest == lo) {
                        this.segments[currentSeg].insideTiles[i].southDest = hi;
                    } else {
                        this.segments[currentSeg].insideTiles[i].northDest = hi;
                    }
                } else {
                    if (this.segments[currentSeg].insideTiles[i].eastDest == lo) {
                        this.segments[currentSeg].insideTiles[i].westDest = hi;
                    } else {
                        this.segments[currentSeg].insideTiles[i].eastDest = hi;
                    }
                }
            }
        }
        currentSeg++;
        if (currentSeg < this.crossingNum) {     // init undiscovered segment
            this.segments[currentSeg] = new Segment(this.crossings[newCross], currentSeg);
            this.crossings[newCross].nwSeg = this.segments[currentSeg];
        }
    } else {
        over = 1;
        this.crossings[newCross].overSeg = this.segments[currentSeg];
        this.crossings[newCross].southDest = this.segments[currentSeg].lowerBound;
        this.segments[currentSeg].insideTiles[sg] = this.crossings[newCross];
        this.segments[currentSeg].numElements++;
```

```
                    }
                }       // overcrossing added to 'inside tiles'
                if (currentSeg < this.crossingNum) {
                    // add dowker # to supercrossing
                    this.crossings[newCross].placeDowker(dowkers);
                    dowkers++;
                }

            }   // close case 7-8
        }    // close main switch
    }
    if (this.crossingNum > 0) this.tricolorable = this.isTricolorable();
}

Analysis.prototype.isTricolorable = function() {
    var indx = 0;
    this.isColorable.activation = 0;

    // only attempt to tricolor from crossings in which segments are each unique
    while (((this.crossings[indx].overSeg == this.crossings[indx].nwSeg) ||
            (this.crossings[indx].nwSeg == this.crossings[indx].seSeg) ||
            (this.crossings[indx].overSeg == this.crossings[indx].seSeg))
             && (indx < (this.crossingNum-1))) indx++;

    // set three arbitrary colors
    this.crossings[indx].overSeg.setColor(3);   // set overSeg red
    if (!this.checkForConflicts()) {
        this.crossings[indx].nwSeg.setColor(2);   // set nwSeg green
        if (!this.checkForConflicts()) {
            this.crossings[indx].seSeg.setColor(1);  // set seSeg blue
        }
    }
    singCol = false;
    // if there was a conflict, or if there's a subsequent conflict down the line
    if (!this.checkForConflicts() && this.isColorable(this.findNextCrossing())) {
        this.tricolorable = true;  // GOT IT!  Found a working color pattern
        return true;
    } else {
        var ind;
        // no 3-color solution from this crossing
        // so reset everything, we'll try making them all the same
        for (ind = 0; ind < this.crossingNum; ind++) {
            this.segments[ind].resetColor();
        }

        this.crossings[indx].overSeg.setColor(1);   // set overSeg red
        if (!this.checkForConflicts()) {
            this.crossings[indx].nwSeg.setColor(1);   // set nwSeg green
            if (!this.checkForConflicts()) {
                this.crossings[indx].seSeg.setColor(1);  // set seSeg blue
```

```
            }
        }

        singCol = true;
        colsAdded = false;

        // if there was a conflict, or if there's a subsequent conflict down the line
        if (!this.checkForConflicts() && this.isColorable(this.findNextCrossing())) {
            // Because we began our attempt with a unicolor scheme, verify that the other
            // two colors were eventually introduced.
            var finalCheck = false;       // SImplify this now w colsAdded bool
            var initColor = 1;
            var ii;  // loop var
            for (ii = 0; ii < this.crossingNum; ii++) {
                if (this.segments[ii].color > 1) finalCheck = true;
            }
            if (finalCheck) {
                // GOT IT!  Found a working color pattern
                this.tricolorable = true;
                return true;
            } else {
                // no 3-color solution from this crossing
                // so reset everything, we'll try making them all the same
                var ind;
                for (ind = 0; ind < this.crossingNum; ind++) {
                    this.segments[ind].resetColor();
                }
                this.tricolorable = false;
                return false;
            }
        } else {
            // no 3-color solution from this crossing
            // so reset everything, we'll try making them all the same
            var ind;
            for (ind = 0; ind < this.crossingNum; ind++) {
                this.segments[ind].resetColor();
            }
            this.tricolorable = false;
            return false;
        }
    }
}


Analysis.prototype.isColorable = function(c) {

    var temp, co = 0, xc = 0;

    if (c == null ) {
        if ((this.crossingNum > 2) && (!this.checkForConflicts())) return true;
```

```
    else return false;
}

if (c.overSeg.color) xc++;
if (c.nwSeg.color) xc++;
if (c.seSeg.color) xc++;
if (xc == 3) {
    return true;
} else if (xc == 2) {        // ** IF CROSSING HAS 2 COLORS **

    if (c == null ) {
        if (!this.checkForConflicts()) return true;
        else return false;
    }

    if (!c.overSeg.color) {             // IF IT'S overSeg UNCOLORED
        // then if unders are different, set over to 3rd color
        if (c.nwSeg.color != c.seSeg.color) {
            co = 6 - (c.nwSeg.color + c.seSeg.color);
        } else {
            co = c.nwSeg.color;   //else set over to same color.
        }
        c.overSeg.setColor(co);
        if (!this.checkForConflicts()
            && this.isColorable(this.findNextCrossing())) {
            return true;
        } else {
            c.overSeg.resetColor();
            return false;
        }
    } else if (!c.nwSeg.color) {       // IF IT'S nwSeg UNCOLORED
        // then if others are diff set nw to 3rd color
        if (c.overSeg.color != c.seSeg.color) {
            co = 6 - (c.overSeg.color + c.seSeg.color);
        } else {
            //else set nw to same
            co = c.overSeg.color;
        }
        c.nwSeg.setColor(co);
        if (!this.checkForConflicts()
            && this.isColorable(this.findNextCrossing())) {
            return true;
        } else {
            c.nwSeg.resetColor();
            return false;
        }
    } else {         // IF IT'S seSeg UNCOLORED
        if (c.overSeg.color != c.nwSeg.color) {
            //then if others are diff set se to 3rd color
            co = 6 - (c.overSeg.color + c.nwSeg.color);
```

```
        } else {
            //else set se to same
            co = c.overSeg.color;
        }
        c.seSeg.setColor(co);

        if (!this.checkForConflicts()
            && this.isColorable(this.findNextCrossing())) {
            return true;
        } else {
            c.seSeg.resetColor();
            return false;
        }
    }

} else {        // ** ELSE CROSSING HAS 1 COLOR **
    if (c.overSeg.color) {        // overSeg only colored stub on this s.crossing
        if (c.seSeg == c.nwSeg)  {
            // uncolored stubs belong to the same segment,
            // only one option possible. this might never be executed. EVER.
            c.seSeg.setColor(c.overSeg.color);
            if (this.checkForConflicts()) {
                c.seSeg.resetColor();
                return false;
            }
        } else {       // uncolored stubs belong to different segments
            c.nwSeg.setColor((c.overSeg.color % 3) + 1);
            c.seSeg.setColor((c.nwSeg.color % 3) + 1);
            if (singCol && !colsAdded) colsAdded = true;

            if (!this.checkForConflicts()
                && this.isColorable(this.findNextCrossing())) {
                return true;
            } else {
                // if the first combo no good
                temp = c.nwSeg.color;                  // flip
                c.nwSeg.setColor(c.seSeg.color);           //  n
                c.seSeg.setColor(temp);                    //  retry
                if (singCol && !colsAdded) colsAdded = true;

                if (!this.checkForConflicts()
                    && this.isColorable(this.findNextCrossing())) {
                        return true;
                } else {
                  // hey, who knows- maybe
                    // they're all the same color.
                    c.nwSeg.setColor(c.overSeg.color);
                    c.seSeg.setColor(c.overSeg.color);
                    colsAdded = false;
                    if (!this.checkForConflicts()
```

```
                    && this.isColorable(this.findNextCrossing())) {
                    return true;
                } else {
                    c.nwSeg.resetColor();
                    c.seSeg.resetColor();
                    return false;
                }
            }
        }
    }
} else if (c.nwSeg.color) {  // nwSeg only colored stub on this s.crossing
    if (c.overSeg == c.seSeg)  {
        // uncolored stubs belong to the same segment, only one option possible
        c.seSeg.setColor(c.nwSeg.color);
        if (!this.checkForConflicts() && this.isColorable(this.findNextCrossing())) {
            return true;
        } else {
            c.seSeg.resetColor();
            return false;
        }
    }
    else {
        // uncolored stubs belong to different segments
        c.overSeg.setColor(((c.nwSeg.color%3)+1));
        c.seSeg.setColor(((c.overSeg.color%3)+1));
        if (singCol && !colsAdded) colsAdded = true;

        if (!this.checkForConflicts() && this.isColorable(this.findNextCrossing())) {
            return true;
        } else {
            temp = c.overSeg.color;                    // flip
            c.overSeg.setColor(c.seSeg.color);              //  n
            c.seSeg.setColor(temp);                          //  retry

            if (singCol && !colsAdded) {
                colsAdded = true;
            }

            if (!this.checkForConflicts() && this.isColorable(this.findNextCrossing())) {
                return true;
            } else {
                c.overSeg.setColor(c.nwSeg.color);      // ...and another test for
                c.seSeg.setColor(c.nwSeg.color);        // unified color scheme.
                colsAdded = false;

                if (!this.checkForConflicts() && this.isColorable(this.findNextCrossing())) {
                    return true;
                } else {
                    c.overSeg.resetColor();
                    c.seSeg.resetColor();
```

```
                    return false;
                }
            }
        }
    }
} else {
    // seSeg only colored stub on this s.crossing
    if (c.overSeg == c.nwSeg)  {
        // uncolored stubs belong to the same segment, only one option possible
        c.nwSeg.setColor(c.seSeg.color);
        if (!this.checkForConflicts() && this.isColorable(this.findNextCrossing())) {
            return true;
        } else {
            c.nwSeg.resetColor();
            return false;
        }
    } else {
        // uncolored stubs belong to different segments
        // uncolored segs
        c.overSeg.setColor((c.seSeg.color%3)+1);
        c.nwSeg.setColor((c.overSeg.color%3)+1);
        if (singCol && !colsAdded) colsAdded = true;

        if (!this.checkForConflicts()  &&  this.isColorable(this.findNextCrossing())) {
            return true;
        } else {
            temp = c.overSeg.color;                    // flip
            c.overSeg.setColor(c.nwSeg.color);         //   n
            c.nwSeg.setColor(temp);                        // retry

            if (singCol && !colsAdded) {
                colsAdded = true;
            }

            if (!this.checkForConflicts() && this.isColorable(this.findNextCrossing())) {
                return true;
            } else {
                c.overSeg.setColor(c.seSeg.color);      // ...and YET another test for
                c.nwSeg.setColor(c.seSeg.color);          // unified colors.
                colsAdded = false;

                if (!this.checkForConflicts() && this.isColorable(this.findNextCrossing())) {
                    return true;
                } else {
                    c.nwSeg.resetColor();
                    c.overSeg.resetColor();
                    return false;
                }
            }
        }
```

```
            }
        }
    }    // end 1 color case

}



Analysis.prototype.findNextCrossing = function() {
    var cl = null;
    var indx;

    if (this.crossingNum) {
        for (indx = 0; indx < this.crossingNum; indx++) {

            if ((this.crossings[indx].numColored() == 1)
                || (this.crossings[indx].numColored() == 2)) {
                if (!singCol || colsAdded) {
                    if ((cl == null) || (this.crossings[indx].numColored() > cl.numColored())) {
                            cl = this.crossings[indx];
                    }
                } else {
                    if ((cl == null) || (this.crossings[indx].numColored() < cl.numColored())) {
                        cl = this.crossings[indx];
                    }
                }
            }

        }
    }

    return cl;
}


Analysis.prototype.checkForConflicts = function() {
    var indx, cs;
    var conflict = false;

    for (indx = 0; (!conflict)&&(indx < this.crossingNum); indx++) {
        cs = this.crossings[indx];
        conflict = (((cs.overSeg.color) && (((cs.overSeg.color == cs.nwSeg.color)
            && ((cs.seSeg.color) && (cs.seSeg.color != cs.overSeg.color))) ||
            ((cs.overSeg.color == cs.seSeg.color)
            && ((cs.nwSeg.color) && (cs.nwSeg.color != cs.overSeg.color))))) ||
            ((cs.nwSeg.color) && (cs.nwSeg.color == cs.seSeg.color) &&
             (cs.overSeg.color) && (cs.overSeg.color != cs.nwSeg.color)));
    }

    return conflict;
```

```
}

/*****************************************************************
 *
 *
 *      SEGMENT
 *
 *
 *****************************************************************/

function Segment(c, n){
    this.lowerBound = c;
    this.insideTiles = new Array(1500);
    this.upperBound = null;
    this.numElements = 1;
    this.color = 0;
    this.segNumber = n;
}

Segment.prototype.resetColor = function() {
    this.color = 0;
}

Segment.prototype.setColor = function (c) {
    this.color = c;
}

Segment.prototype.otherEnd = function(crs) {
    if (crs == this.lowerBound) return this.upperBound;
    else return this.lowerBound;
}

Segment.prototype.drawColor = function() {
    var l = (this.lowerBound.col - 1) * brd.ss;
    var t = (this.lowerBound.row - 1) * brd.ss;

    if (this.color) {
        var colr = this.color * 8;

        if (this.lowerBound.nwSeg == this) {
            if (this.lowerBound.type == 7) {
                //l -= 30;
                cv.drawImage(brd.tilePics[7 + colr], 0, 0, 8, 30, l, t, (brd.ss*.3), brd.ss);
            } // west seg
            if (this.lowerBound.type == 8) {
                //t -= 30;
                cv.drawImage(brd.tilePics[8 + colr], 0, 0, 30, 8, l, t, brd.ss, (brd.ss*.3));
            } // north seg
        } else if (this.lowerBound.seSeg == this) {
            if (this.lowerBound.type == 7) {
```

```
            //l += 30;
            cv.drawImage(brd.tilePics[7 + colr], 22, 0, 8, 30, (l+(brd.ss*.7)), t, (brd.ss*.3), brd.ss);
        }  // east seg
        if (this.lowerBound.type == 8) {
            //t += 30;
            cv.drawImage(brd.tilePics[8 + colr], 0, 22, 30, 8, l, (t+(brd.ss*.7)), brd.ss, (brd.ss*.3));
        }  // south seg
    }

    var index = 0;
    var sq = null;

    while (index < (this.numElements - 2)) {
        sq = this.insideTiles[index];
        l = (sq.col - 1) * brd.ss;
        t = (sq.row - 1) * brd.ss;

        if (sq.type < 7) {
            cv.drawImage(brd.tilePics[(sq.type + colr)], 0, 0, 30, 30, l, t, brd.ss, brd.ss);
        } else if (sq.type == 7) {
            cv.drawImage(brd.tilePics[7 + colr], 8, 0, 13, 30, (l+(brd.ss*.3)), t, (brd.ss*.4), brd.ss);
        } else if (sq.type == 8) {
            cv.drawImage(brd.tilePics[8 + colr], 0, 8, 30, 13, l, (t+(brd.ss*.3)), brd.ss, (brd.ss*.4));
        }

        index++;
    }

    l = (this.upperBound.col - 1) * brd.ss;
    t = (this.upperBound.row - 1) * brd.ss;

    if (this.upperBound.nwSeg == this) {
            if (this.upperBound.type == 7) {
                cv.drawImage(brd.tilePics[7 + colr], 0, 0, 8, 30, l, t, (brd.ss/4), brd.ss);
            } // west seg
            if (this.upperBound.type == 8) {
                cv.drawImage(brd.tilePics[8 + colr], 0, 0, 30, 8, l, t, brd.ss, (brd.ss/4));
            } // north seg
    } else if (this.upperBound.seSeg == this) {
            if (this.upperBound.type == 7) {
                cv.drawImage(brd.tilePics[7 + colr], 22, 0, 8, 30, (l+(brd.ss*.75)), t, (brd.ss/4), brd.ss);
            }  // east seg
            if (this.upperBound.type == 8) {
                cv.drawImage(brd.tilePics[8 + colr], 0, 22, 30, 8, l, (t+(brd.ss*.75)), brd.ss, (brd.ss/4));
            }  // south seg
    }
  }
}
```

```
/*********************************************
 *
 *
 *      SUPER-CROSSING
 *
 *
 *********************************************/

 function SuperCrossing(c){
    this.cell = c;
    this.row = c.row;
    this.col = c.col;
    this.type = c.type;
    this.northDest = null;
    this.eastDest = null;
    this.southDest = null;
    this.westDest = null;
    this.overSeg = null;
    this.nwSeg = null;
    this.seSeg = null;
    this.colored = 0;
    this.oddDowker = null;
    this.evenDowker = null;
}

SuperCrossing.prototype.isMe = function(c) {
    return (c == this.cell);
}

SuperCrossing.prototype.placeDowker = function(n) {
    if (n%2 == 0) {
        this.evenDowker = n;
    } else {
        this.oddDowker = n;
    }
}

SuperCrossing.prototype.numColored = function() {
    var numcol = 0;

    if (this.overSeg.color) numcol++;
    if (this.nwSeg.color) numcol++;
    if (this.seSeg.color) numcol++;

    return numcol;
}

 /*********************************************************************
 *
```

```
 *
 *      SUPER-LINE
 *
 *
 ************************************************************************/


 function SuperLine(c, n){
    this.cell = c;
    this.row = c.row;
    this.col = c.col;
    this.type = c.type;
    this.northDest = null;
    this.eastDest = null;
    this.southDest = null;
    this.westDest = null;
    this.segNum = n;
}

SuperLine.prototype.isMe = function(c) {return (c == this.cell);}


 /************************************************************************
 *
 *
 *      SUPER-CORNER
 *
 *
 ************************************************************************/


 function SuperCorner(c, n){
    this.cell = c;
    this.row = c.row;
    this.col = c.col;
    this.type = c.type;
    this.northDest = null;
    this.eastDest = null;
    this.southDest = null;
    this.westDest = null;
    this.segNum = n;
}

SuperCorner.prototype.isMe = function(c) {
    return (c == this.cell);
}
```

References

Adams, C. C. (2001). *The knot book: An elementary introduction to the mathematical theory of knots*. New York, NY: W. H. Freeman and Company.

Hass, J., Lagarias, J.C., & Pippenger, N. (1999). "The computational complexity of knot and link problems." *Journal of the ACM*, 46(2), 185-211.

Scharein, R. (2014). *The KnotPlot site*. Retrieved from: http://www.knotplot.com/