9.Shaping Space

The spatial arrangement of notes in Tinderbox maps can express both explicit and implicit relationships among those notes. Some of those relationships are evident from the outset, while others emerge over time, and some relations we once thought clear and useful may turn out to be muddled or unhelpful. Throughout the life of a Tinderbox project, map views provide a way to visualize a section of our growing and changing collection of notes.

All kinds of concept visualizations set out to find a spatial arrangement that communicates structure and relationships among ideas. Timelines, for example, arrange a list of events, using intervals of space to describe intervals of time[34]. Chart views take the hierarchical structure of a Tinderbox document and arrange it in a genealogical format. A variety of experimental concept visualizations have been proposed, ranging from cone maps and three-dimensional cityscapes to hyperbolic browsers.

Tree charts and timelines determine their layout from the properties of each note, while maps rely on you to choose a position that makes sense. If a timeline describes the advancement of astronomy in the 20th century, then an event in 1950 will appear near its center. The Tinderbox map, on the other hand,

[34] For a richly illustrated overview of timelines through history, see D. Rosenberg and A. Grafton, *Cartographies of time*. Princeton Architectural Press. 2010

avoids automatic layout: when you create a note, you put it where you want it. If you later decide that it belongs somewhere else, you will move it where it belongs. Tinderbox maps acquire meaning because you arrange them meaningfully.

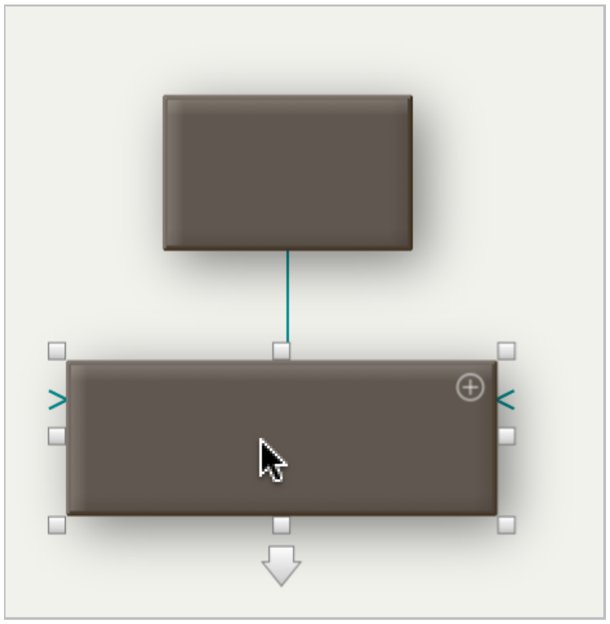
Lining Up

In the earliest Tinderbox experiments, as in early Storyspace, you could place any note anywhere. This gave users complete freedom, but (for some purposes) that freedom was not always convenient. Rectangles are common in the Tinderbox map, and people frequent want adjacent rectangles to be exactly aligned. Positioning notes by hand often leaves small errors. Further, since any location is equally valid, placing a note at `$Xpos=3.473206` is neither more or less common than placing one at `$Xpos=1`.

Tinderbox for many years imposed a grid on the plane of the map, requiring note positions to fall at intervals of ½ or ¼ of a map unit. If you dropped a note at `$Xpos=3.473206`, it would "snap" to `$Xpos=3.5`. If you dragged a resize handle to set the note's height to 1.0937, its `$Height` would be rounded to 1.0 .

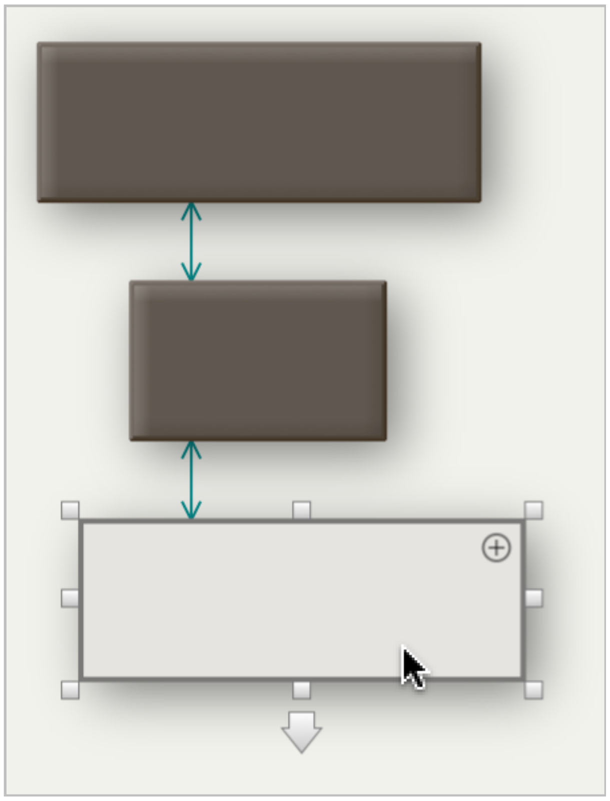
Unwanted alignments imposed by this grid sometimes became a nuisance. Sometimes you didn't want a note to align with its neighbors, but the coarse grid forced the alignment anyway. Over time, Tinderbox adopted a finer grid to provide more flexibility, even though this made it slightly harder to align notes. We began to explore various escape hatches to allow specific notes to ignore the grid entirely.

Shaping Space



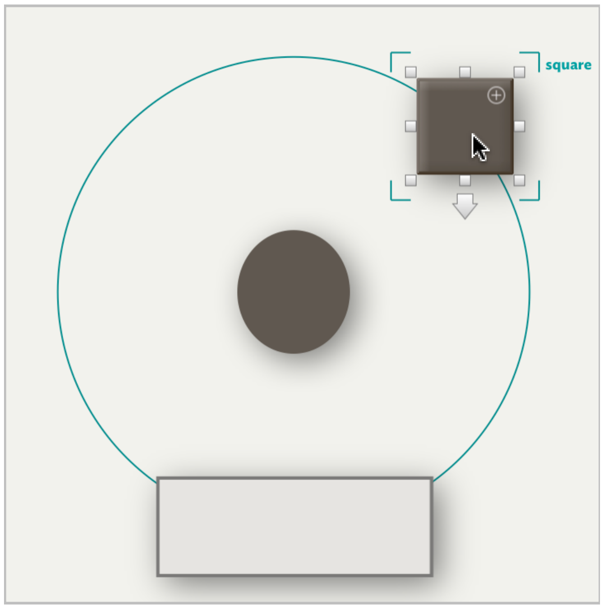
Tinderbox 7 adopts an entirely new approach to regulating geometric relationships among notes. Instead of imposing a global grid to which all notes have to adhere, it tries to recognize *local geometries* and to apply those geometries to notes created in or moved into the locale.

Here, for example, we are dragging a wide, rectangular note beneath a smaller note. Because the horizontal centers of the two notes are nearly aligned, Tinderbox displays a blue guideline and, when the mouse is released, will snap the position of the dragged note so the two centers are aligned exactly. The upper note exerts a sort of gravitational force that leads the dragged note to align with its center. Similarly, Tinderbox will recognize when the edges of the notes are nearly aligned.



An even more intriguing example appears when we drag a note—in this case a light gray note—beneath two other notes. Here, none of the edges align, but Tinderbox notices that the vertical space between this note and the note above it is nearly the same as the space between that note and the topmost note. Tinderbox again draws a guide, and when the mouse is released it will adjust the note position so the vertical spacing is identical. The presence of two nearby notes thus sets up a sort of force field that encourages consistent spacing without imposing any preconception of what that spacing ought to be.

The presence of circular or oval notes suggests opportunities to use polar coordinates. Here, we are moving a square note, and Tinderbox recognizes that the note is near a round note, and that another note is already nearly equidistant from the center of that round note. Releasing the mouse will snap the dragged note so its distance from the nearby circle is identical to the distance between the circle and the center of the gray note beneath it.



Design Note: Kibbitzers vs. Parsers

The ephemeral blue *guides* that shape the map geometry in Tinderbox 7 appear simple, almost as simple as the old, rigid grid. Their impact, however, is subtle, wide-ranging, and not merely cosmetic. The radial guides, for example, offer a new semantic meaning for circular and oval notes while inviting their use as the centerpoint for radial clusters that constitute organized but informal piles (Figure 9-1). A variety of layouts arise naturally from simple, local guides; for example, it is natural to organize groups of notes separated internally by a small (but consistent) space, with the groups themselves divided by a larger (but also consistent) space (Figure 9-2).

The use of local guides to set up local geometries was an alternative to an earlier design that called for a window in which the user would designate the grid to be used for each map. In preliminary sketches, the grid design dialog was either inflexible, giving little control beyond spacing between notes, or became overly elaborate. Worse, the margins of large maps often hold legends, prototypes, todo lists, and metadata—clusters of elements that are not unstructured but that do not share the geometry imposed on the overall map. Tinderbox maps want to express local structure, and that structure ought to be permitted to emerge over time. Just as in actual cities, the grid plan for one neighborhood may not suit another.

Each time a note moves, a crowded map offers many potential alignments. The computer must select the alignments it will suggest from among all conceivable guidelines, and must do so without impeding the animated motion of the objects being moved. In practice, that means it has perhaps 1/60 sec (16.7 msec) to examine the map and suggest potential guidelines.

The first key to implementation is building an efficient representation that tells us which notes are in any particular locale. For example, if a guide wants to align our top with the top of our nearest neighbor, it needs to know which note is closest to the dragged note's position. A simple approach would look at each note in the map and find the closest, but as the number of notes in the map increases, that becomes too slow. Tinderbox's data structure lets us focus only on a subset of notes in the neighborhood of the dragged note: even in a large and crowded map, we only need to examine a fraction of the notes in the map to find the note closest to the cursor.

Each kind of guide is then embodied in a separate advisory object that we call a *kibbitzer*.[35] Whenever a note is dragged, Tinderbox asks each kibbitzer in turn whether it would like to make a suggestion. At any given moment, only a few kibbitzers will have anything to suggest; those that do have a proposal to make are permitted to draw their proposed guidelines. Individual kibbitzers are narrowly focused, which makes them simple to write; typical kibbitzers include:

SquareAspectRatioKibbitzer
VerticalCenterKibbitzer
RightMidpointSpacingKibbitzer
TopAdornmentSpacingKibbitzer

The "top adornment spacing kibbitzer", for example, focuses only on notes that are inside an adornment and near its top edge. The kibbitzer knows how much vertical space should be reserved for the adornment title, and suggests that the top of the note being moved should align with the bottom of the space reserved for the adornment title.

[35] *Kibbitzer* is a Yiddish term for someone who watches a chess or card game and offers advice.

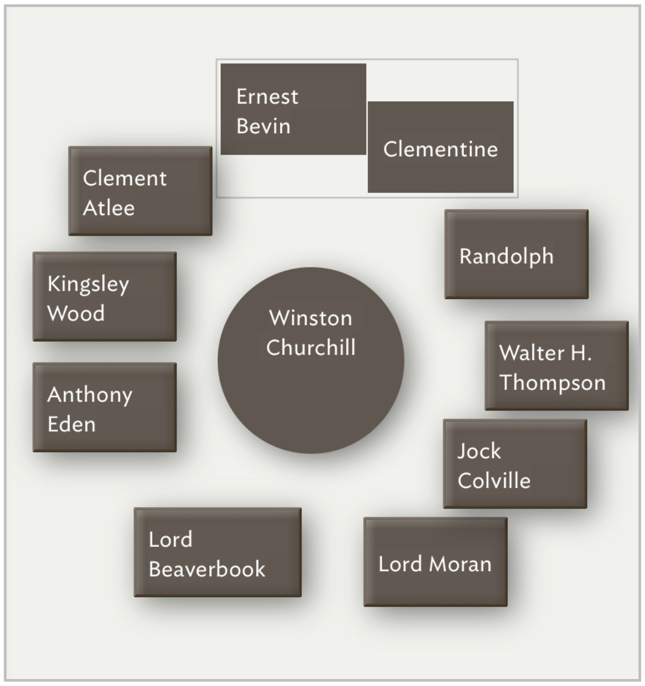


Figure 9-1. Radial guides suggest a map organization in which a central note acquires a halo of subsidiaries.

Most kibbitzers focus on the moving note's immediate neighbors. This limits the number of potential alignments and avoids presenting a confusing welter of guidelines. Even so, several different kibbitzers may have differing proposals for the same coordinate. For example, one kibbitzer might say

> "You're nearly aligned with this note's top."

and a different kibbitzer might say

> "Your vertical space from the note above you is almost—but not quite—the vertical space you used nearby."

Each kibbitzer wants to change the vertical position, but they may suggest different locations. To resolve this conflict, we assign each kibbitzer a priority, and though we display all the guidelines only

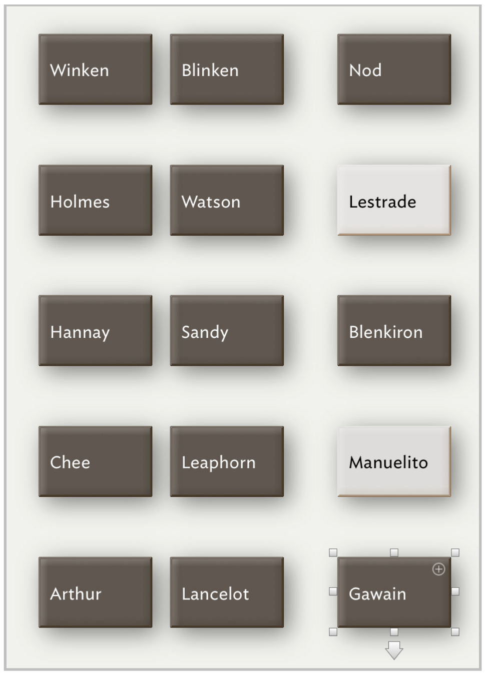
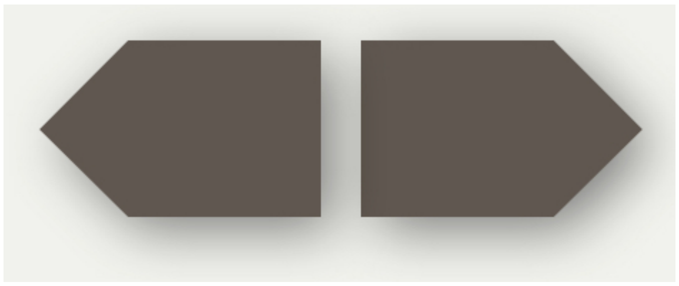


Figure 9-2. Local guides can promote consistency in syncopated and irregular grids.

the kibbitzers with the highest priority will act on the just-moved note.

Parsing Patterns

Spatial hypertext researchers have long been interested in automatically recognizing and exploiting the sort of pattern language described in Chapter 7 (Tinderbox Maps). For example, the computer might automatically recognize an



antisymmetric pair of notes as a higher-level structure. We might well want to move this pair of notes as a unit, for example, rather than moving them separately. We might want to act on them as a unit, to change their color or to update their attributes. The left-hand member of this part might have rules or actions that depend on its right-hand counterpart. This pattern might have a specific semantic meaning; we might use this particular arrangement to represent an author/title pair, or perhaps to represent mutually-exclusive policy options. If so, the computer could help us by automatically recognizing errors in placement or classification, much as the spelling checker recognizes typographic mishaps.

The first automatic pattern recognition system in a hypertext environment was Marshall and Shipman's VIKI, the system that most directly inspired Tinderbox.[36] VIKI called this a *spatial parser,* and the spatial parser was the most distinctive feature of VIKI and its successors. I wanted a spatial parser for Tinderbox, but held off for two reasons:

- **Performance**. VIKI and its successors ran on workstations substantially faster than personal computers, and their users—computer scientists at Xerox PARC, then the world's leading research center for computer science—were more tolerant of minor delays and inconveniences than commercial software users. Lots of things we want to do with parsers, moreover, either require very clever algorithms or would perform far more slowly as the document to be parsed grows.
- **Finickiness and Failure**. When the parser fails to see a pattern we expected it to recognize, it seldom has a good way to explain why it failed[37]. Since the parser can't know that we expected it to recognize anything, it has no reason to explain. If we relax the constraints on the parser, it seems overeager; if we constrain too

[36] C. C. Marshall, Frank M. Shipman III, and J. H. Coombs, 1994. "VIKI: Spatial Hypertext Supporting Emergent Structure". *European Conference on Hypertext ECHT'94.* 13-23. C. C. Marshall and Frank M. Shipman III, 1997. "Spatial hypertext and the practice of information triage". *Proceedings of the Eighth ACM conference on Hypertext*. 124-133.

[37] The vagaries of language learning present the same problem. If a proficient non-native makes a slight error in grammar or pronunciation, it may be easy to understand what they intended to say. More substantial errors, on the other hand, can be very puzzling indeed; that there is not mutual understanding is clear, but how to correct the mistake may be a complete mystery.

tightly, we must do extra work—lining up notes just so—in order to cajole the parser into doing what we want.

In Tinderbox 7, the role typically played by the spatial parser is instead divided. Kibbitzers handle the housekeeping chores of recognizing and maintaining alignments, and *composites* of notes, formed when two or more notes touch, create meaningful structures. These structures can, in turn, be repeatedly instantiated, allowing you to declare that a composite plays a particular role rather than forcing the parser to deduce what you intend.

10. Treemaps

Visualization seeks to convey information in ways that take advantage of human perception. We draw graphs, for example, because people are better at recognizing the direction of trend lines than at understanding long lists of numbers. This is a circumstance that depends on chance and evolution; we can easily imagine beings that would find it easier to study a list of numbers than to recognize the direction of a line on the page. Indeed, it is far more difficult for a computer program to understand the image of even of most well-drawn stock-market graph than to analyze a list of closing prices.

Human vision is powerful but its capacity is finite, and when we consider ways to visualize complex networks of concepts and observations, we always need to keep in mind the limitations of computer displays and of human vision. An outline can show us a useful overview of a document's hierarchy, but outlines can draw only only a few dozen notes on the screen at a time. Maps clarify the relationships among notes, but a map with many notes is harder to understand than a map with only a few. As we have seen, the map of Mary-Kim Arnold's story, "Lust" (Figure4-3) holds only 36 notes and 105 links, yet even this small map is quite difficult to take in at a glance.

An outline can display more items if we make the type smaller, and a map can display more items if we reduce its scale, but

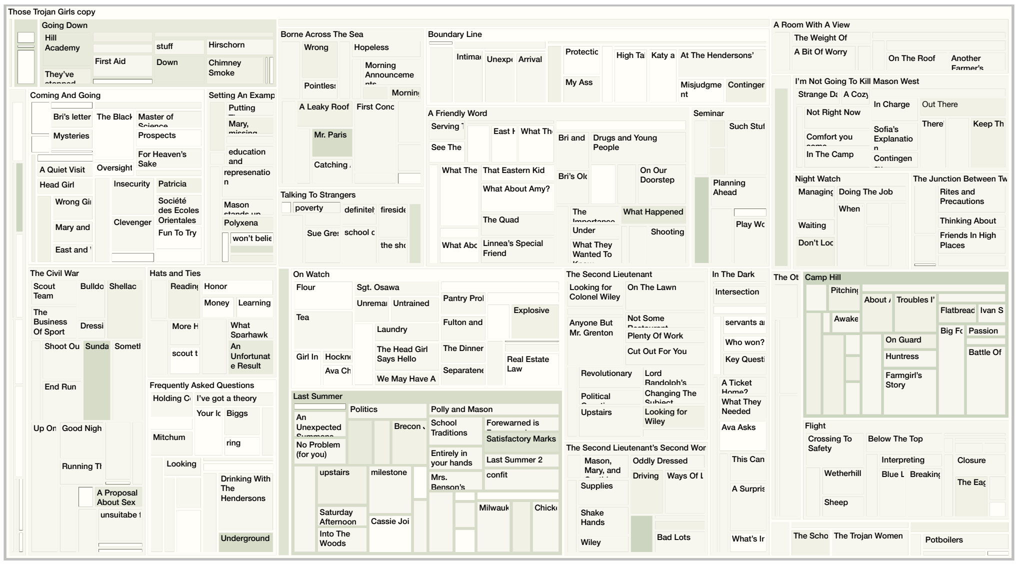


Figure 10-1: A treemap view of *Those Trojan Girls* shows each note as a separate box.

eventually the type will be too small to see. (Type that is less than seven or eight pixels high will be illegible, even for people with the most acute vision.) Larger screens with higher resolution can increase the amount of information displayed at one time, but again, human visual acuity will eventually impose its limits. People cannot read microscopic type, and once the screen fills our visual field—about 114°—we cannot see the entire view at once.

The *treemap* view is designed to represent the largest number of notes, along with their position in the document hierarchy, into the smallest possible area. To construct a treemap of a Tinderbox document, we follow this simple procedure:

- The root of the document (or the parent of the the part of the document we want to examine) is represented by a rectangle that fills the entire available space.
- Each child of the root is assigned a fraction of its parent's rectangle in proportion to the number of descendants that child has.
- Each child, in turn, divides its allotted space proportionately among its children.
- If a note's allotted rectangle is too small to draw, it is omitted.
- We continue until all descendants have been drawn, or until no more notes can be drawn.

In Figure 10-1, most or all of the 405 notes that comprise my hypertext fiction *Those Trojan Girls* appear simultaneously on the screen. In contrast, my outline view shows 24 notes at a time, and a typical map view shows 19 notes, some of which are partially off-screen.

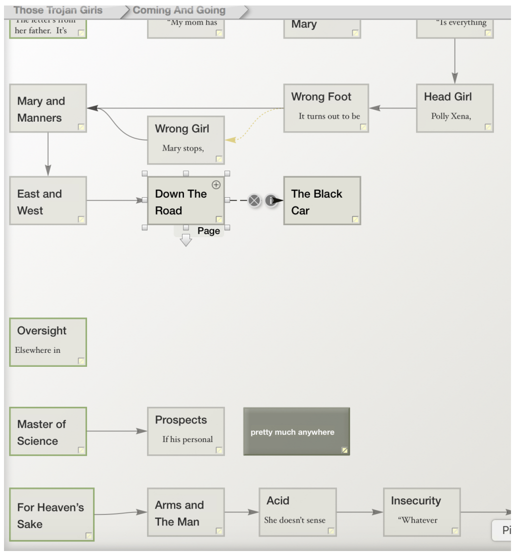


Figure 10-2: a small excerpt from *Those Trojan Girls* in map view. Maps show links and visual attributes, but treemaps can show many more notes at one time.

The compactness of the treemap view is not without disadvantages. Since every pixel of the available space is apportioned to one note or another, there's no place to draw links, plots, badges, or summary tables. In many cases, the available space is inadequate even for drawing the note title.

In general, Tinderbox arranges treemaps so the first child is near the top left of its parent rectangle, and the youngest child is near the bottom right. Tinderbox also tries to ensure that the rectangle assigned to a note isn't too narrow or too short; a square note and a very thin, tall note might have the same area, but the square note leaves space for a title and is easier to click. In order to find a good layout and to avoid wasting space when notes don't fit,

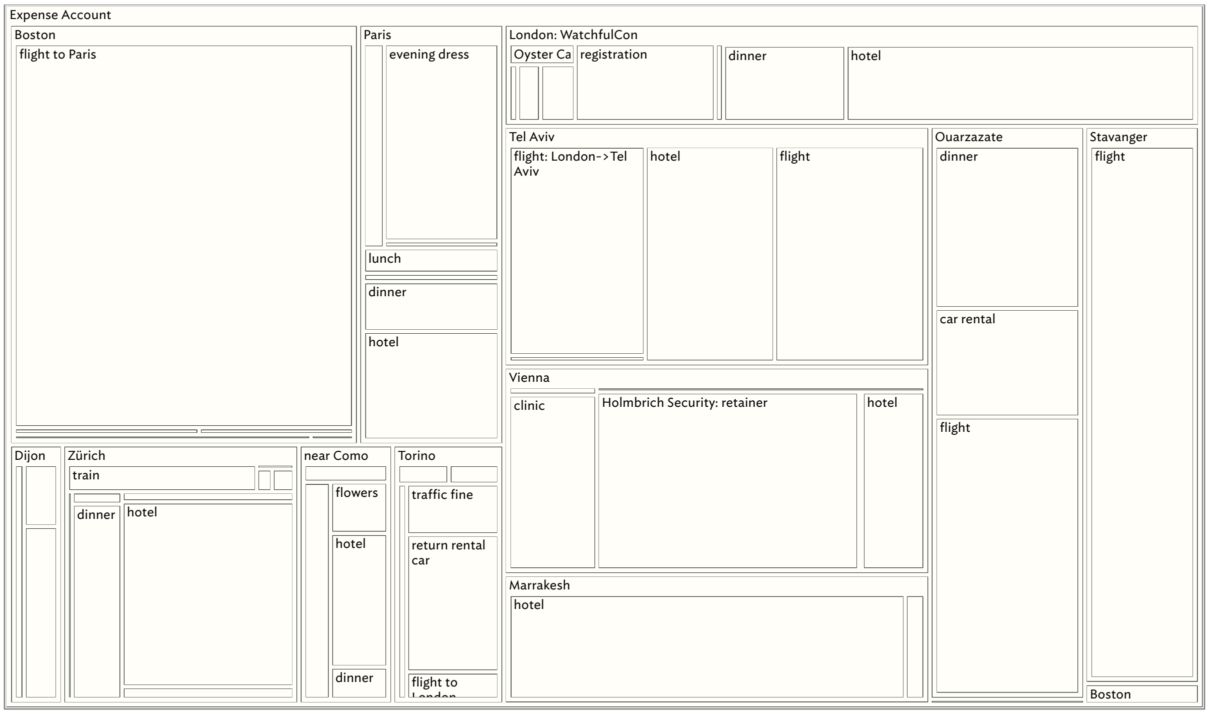


Figure 10-3: A treemap of expenses, weighted by the cost of each purchase.

Tinderbox may place move notes in the treemap left or right, up or down[38].

Weighting Treemaps

Normally, each descendant of a note carries equal weight in the treemap, and the area of each note is proportional to the number of its descendants. Often, though, we might prefer to assign weight to notes based on their properties. In Figure 10-1, the area of each note is proportional to its word count; this lets us see

38 Treemaps were invented by Ben Shneiderman and his students at the university of Maryland in the early 1990s: Ben Shneiderman,. "Tree Visualization with Tree-maps: 2-d Space-filling Approach". *ACM Trans. Graph. 11,* 1, (1902) 92–99. I learned of them through an early SIGCHI paper:—B. Johnson, "TreeViz: Treemap Visualization of Hierachically Structured Information". *CHI'92.* 369–370. The modern layout algorithm emerged a decade later: B. B, Bederson, Ben Shneiderman,, and Martin Wattenberg, "Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies". *ACM Trans. Graph. 21,* 4, (2002) 833–854. The best discussion for the general reader, lavishly illustrated, is Manuel Lima, *The book of trees : visualizing branches of knowledge.* Princeton Architectural Press, 2014.

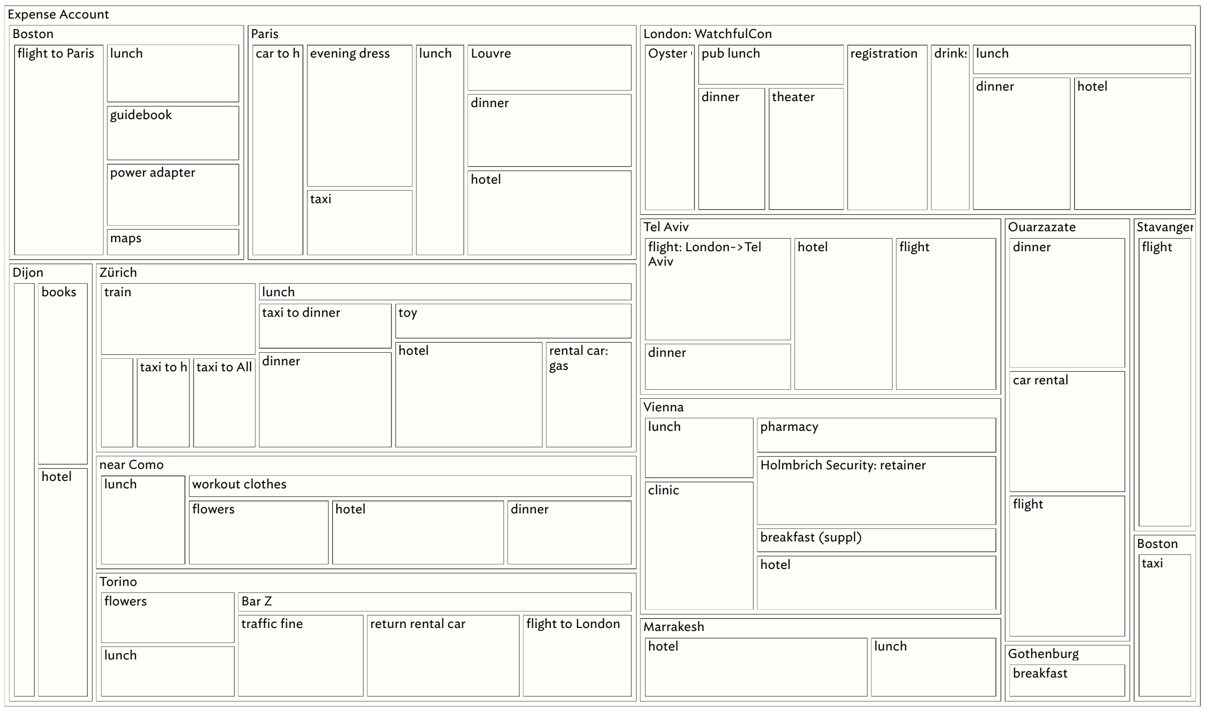


Figure 10-4: The expense report from Figure 10-2, weighted by log($Dollars) to allow us to examine smaller expenses while still giving more space to larger costs.

which chapters are long, which short, and also whether a section contains a few long notes or many brief ones.

In Figure 10-3, based on the Treemaps chapter of Tinderbox's "Getting Started" tutorial, we have a Tinderbox document that records expenses incurred in a fictional business trip. Large expenses, such as long plane flights and legal retainers, are allocated a large space in the treemap, while snacks and sundries are smaller or are omitted entirely.

While this display draws attention to the largest expenditures, it doesn't use space very efficiently: the flight to Paris was costly, but as a result a large chunk of the treemap is empty space. Instead of weighting the treemap by `$Dollars`, the amount spent, Figure 10-4 weights it instead by `log($Dollars)`[39]. Large costs are still given greater

[39] To change the weighting of notes in the treemap, set the weight expression in the treemap tab's Information popup, which appears on clicking the ⓘ button.

weight, but we now have space to display a far greater range of items.

Colored Treemaps

We may also choose a color for different notes in the treemap based on their properties. The Treemap *info* popover lets us choose:

- An *expression* to evaluate for each note, returning a numeric value.
- A *start color*, which will be applied to notes returning the smallest values.
- An *end color,* which will be applied to notes returning the largest values.

In Figure 10-1, for example, notes with many outbound links are green, while those with few or no outbound links are beige.

11. Agents

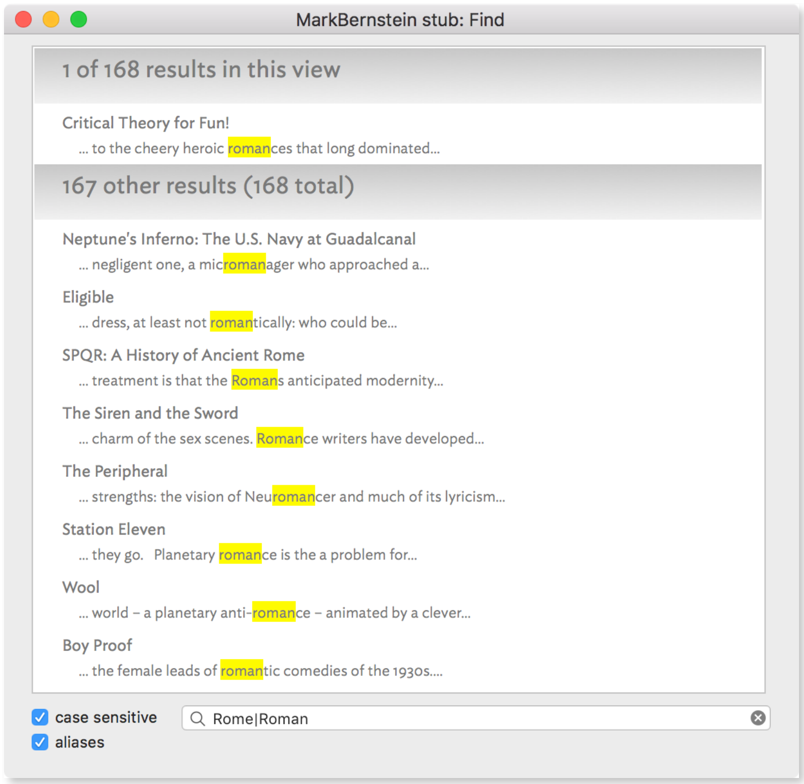
Ephemeral Searches

Tinderbox excels at finding things.

Because Tinderbox offers you lots of ways to organize your notes, you're less likely to misplace old notes or to forget what you were working on last night. Maps provide spatial cues, and though the placement of a note in a map may be arbitrary, its location with respect to other notes and to landmarks like adornments helps you remember where things are.

When searching to relocate information they have seen before, people often use spatial cues that seem irrelevant and that they themselves are surprised they remember. When looking for a quotable passage in a novel, for example, it's not uncommon for people to remember approximately where that passage fell on the page. When searching for a lost memo or a missing book, people may remember aspects of layout and color long after they last saw the document.

Outlines provide a different kind of cue. By breaking long lists into small components, outlines situate things precisely, putting them where they belong—and where you expect to find them later.


MarkBernstein stub: Find

1 of 168 results in this view

Critical Theory for Fun!
... to the cheery heroic romances that long dominated...

167 other results (168 total)

Neptune's Inferno: The U.S. Navy at Guadalcanal
... negligent one, a micromanager who approached a...

Eligible
... dress, at least not romantically: who could be...

SPQR: A History of Ancient Rome
... treatment is that the Romans anticipated modernity...

The Siren and the Sword
... charm of the sex scenes. Romance writers have developed...

The Peripheral
... strengths: the vision of Neuromancer and much of its lyricism...

Station Eleven
... they go. Planetary romance is the a problem for...

Wool
... world – a planetary anti-romance – animated by a clever...

Boy Proof
... the female leads of romantic comedies of the 1930s....

[✓] case sensitive
[✓] aliases
Rome|Roman

Figure 11-1. Find in action. Remember that the Find popover can be torn off to become a window.

Sometimes, though, you might forget where you put a note, or you may need to find a note in an unexpected context. The Tinderbox Find palette offers some powerful features that are sometimes overlooked.

First, remember that you can search for the patterns in the note name and text, or search specifically in either. You can also search the contents of a user attribute.

> **Tip**: In documents that depend on lots of attributes, it's easy to forget that searching Name and Text might not locate the notes you're seeking. For example, if your bibliographic notes have separate attributes for author names, those names might not appear in the text at all.

Some punctuation characters have special meanings when searching. For example, "." matches any character, and "$" matches the end of a paragraph. These special characters are discussed below (see Regular Expressions). If you want to search for characters like "." that have a special meaning in regular expressions, precede them with a backslash '\'.

Not.	Matches "Note" and "Notify"
Not\.	Matches "Not."

Persistent Searches

Tinderbox *agents* are persistent searches. Like the Find window, they scan your Tinderbox document, seeking a specific text pattern and identifying notes that match the pattern. Agents can also look for notes with combinations of properties. But unlike the Find window, agents remain active in your documents indefinitely. Agents hold aliases to all the items that currently match their query.

Agents are notes. Like other Tinderbox notes, each agent has a name, text, and attributes that control the agent's appearance and behavior. Agents have attributes like `$Name`, `$Width` and `$ChildCount` just like other containers, and these can be viewed and exported.

You can use an agent to create alternate views of a container. For example, suppose we have a container of notes about movies we have seen this year. This container happens to be sorted in reverse chronological order, making it easy to review our most recent notes and to add new notes.

```
Films
    The Two Towers
    Six Degrees of Freedom
    Bringing Up Baby
    Apocalypse Now
    ... (many more films)
```

This may well be the most convenient way to add new notes, but when we want to look up a specific movie, *To Have And Have Not*, we need to scan the entire list. For finding a specific movie, an alphabetized list would be much easier to use. An agent can easily build an alphabetically-sorted index:

```
Agent:  Films by Title
Query:  inside(Films)
Sort:     Name
```

This index is always up to date; Tinderbox agents run periodically and update their contents automatically.

Hint: you can toggle **File ▸ Update Agents Automatically** to enable or disable automatic agent updates. In older versions of Tinderbox, agent updates could interrupt other work; because this is no longer an issue, you might consider returning to automatic updates in existing documents in which you have disabled them.

You may reduce the priority of particularly complex agents, so they will run less frequently.

Agent Actions

Like containers, agents may perform actions on notes that they locate. These agent actions set the values of attributes. For example:

```
Query: $Status=="Urgent"
```

Action: $Color="red";

Note: Tinderbox uses == to test two things for equality, and = to assign a value.

This agent finds all notes that have been marked as Urgent and sets their color. Any note that becomes Urgent will be red—and will remain red until some other action changes the color.

Tinderbox actions are usually very simple, but combining these simple actions with agent queries turns out to be remarkably powerful and flexible. Actions are discussed in Chapter 12.

Agents can help discover structure in your notes. It is easy, for example, to build agents that construct topical categories.

Name:	Elizabethan
Query:	$Text.contains(Shakespeare)| $Text.contains(Marlowe)| $Text.contains(Fletcher)

Action: $Drama = true; $Tags = $Tags+ "Exciting"

Here, we collect all notes that mention three Elizabethan playwrights, we mark these notes as concerning Drama, and we add the tag "Exciting" to whatever tags the note already has.

Note: In Tinderbox queries, "|" means "or" and "&" means "and." The operator "!" means "not."

The expression !($Text.contains("Ionesco")) is true if the note's text doesn't mention Ionesco.

Cooperating Agents

It's often helpful to let agents use the work that other agents have already done.