

Lecture 14:

Command Objects & Support for Undo



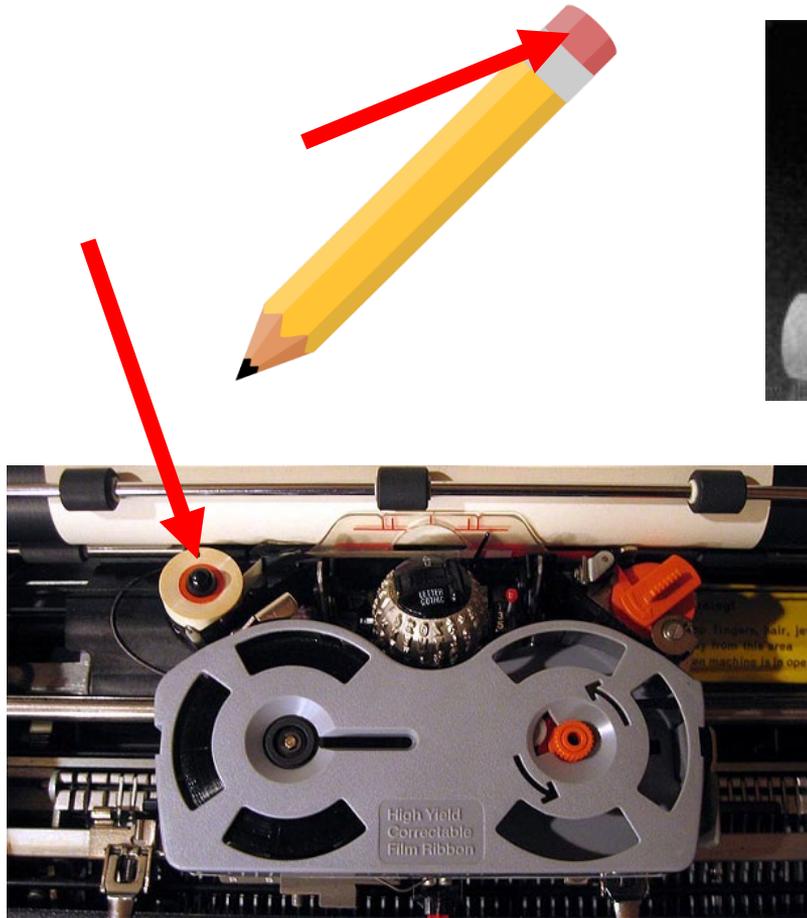
05-431/631 Software Structures for User Interfaces (SSUI)
Fall, 2022



Logistics

- Midterm *now*
- Thanks for attending class anyway
- No class next week
- This lecture is how to do HW 5

Early Undo



**Invented 1951 by
Bette Nesmith Graham**

**IBM Correcting
Selectric II
1973**

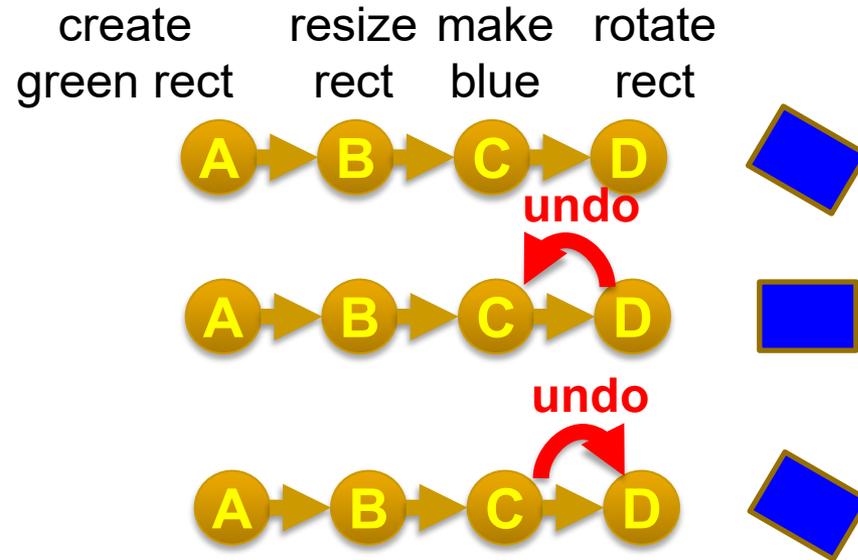


Computer “Undo”

- **Undo** is reversing a previous operation so that it no longer is in effect
 - Usually ^Z
 - For web apps, sometimes the Back button in a browser
- **Cancel** is stopping an operation *while it is in progress*
 - Often ESC key or the “Cancel” button in a dialog box

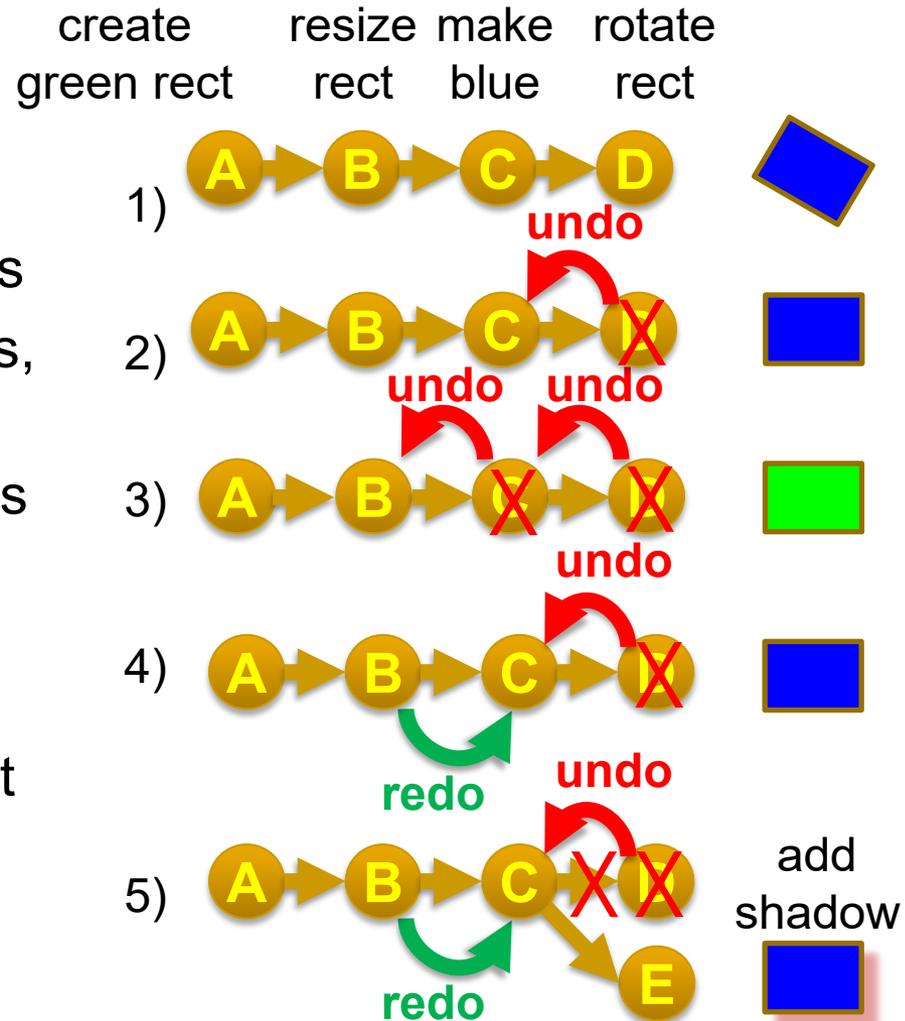
Single Level Undo

- Just toggles the latest item on the list



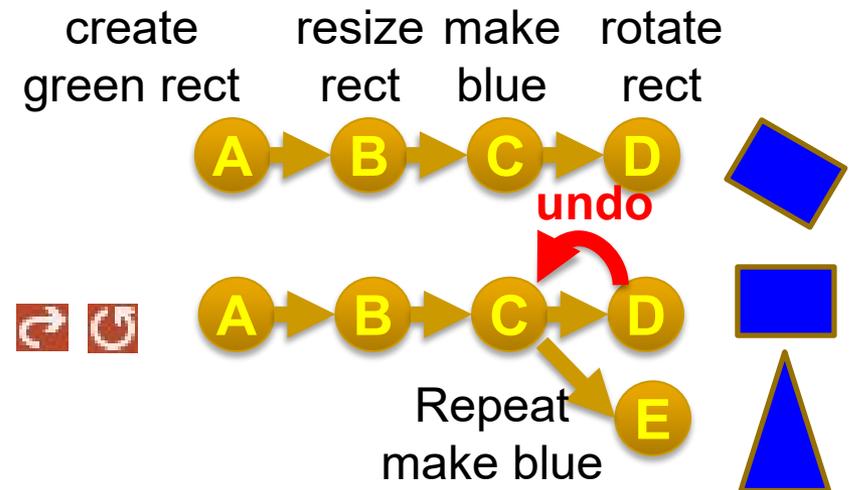
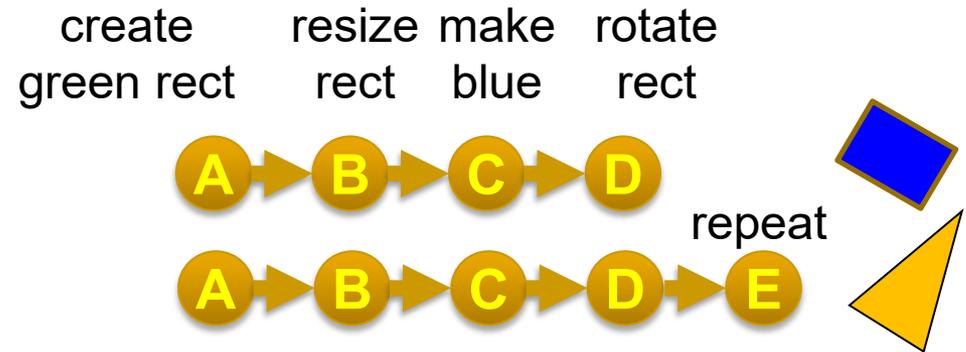
Linear Undo

- Keep a list of all operations
- Undo (^Z) goes backwards, repeatedly
- Redo (^-Shift Z or ^Y) goes forwards *after an undo*
 - Undo the undo
- New operations remove anything undone – it is lost forever



Repeat

- Does the previous operation again on the current selection
- E.g., rotate something else by the same amount
 - Really useful
- Goes on the undo stack just like normal operations
- Typically, uses same shortcut key as Redo
 - But might want to repeat the previous command after an undo
 - Office changes icon
- Repeat is often not available



Complications: Operations not put on Undo Stack

- Scrolling
 - Might be useful to have a “go back”, like with hyperlinks
 - See research later
- Changing the selection
 - not undoable, doesn't change undo stack
 - My Topaz system made this available for undo – see later
- Changing the value of controls, if doesn't affect any objects
 - Changing the color of the next-drawn object
- Copy (as in Cut-Copy-Paste)
 - Clipboard changes are not affected by undo
 - Lots of clever strategies take advantage of this
 - Also not possible since clipboard is global and undo is per-application
- Saving to file is not undoable
 - Old: blocks off all previous operations
 - Current: not put on undo stack so can undo past saves



Complications: operations that are collected

- Multiple characters typed grouped into one undo
 - Similarly, multiple backspaces
- Used of arrow keys to “nudge” graphics often grouped into 1 operation
- Or, one operation causes multiple entries on undo stack: teh_
→ the_ (auto-correct; text)

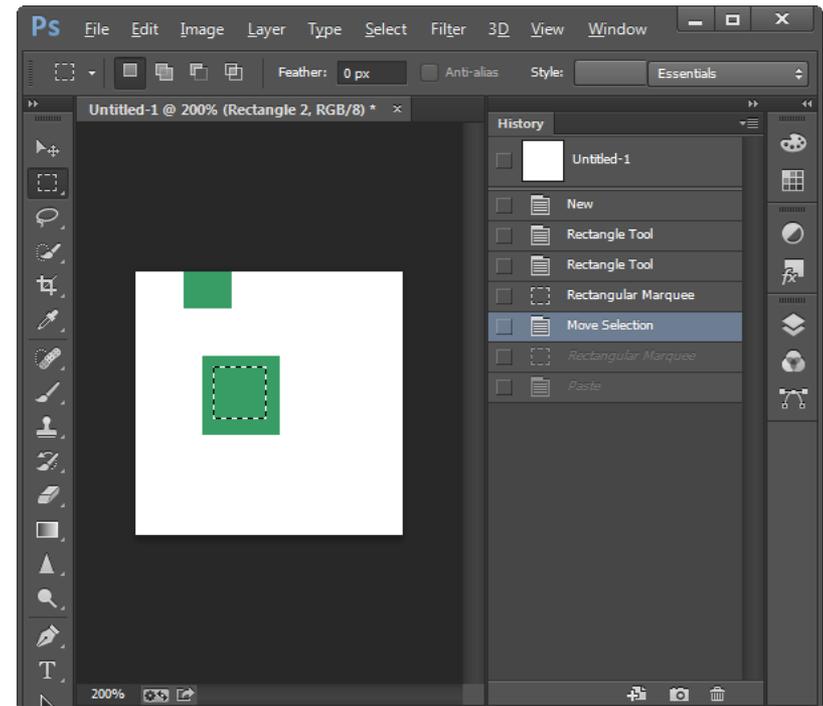
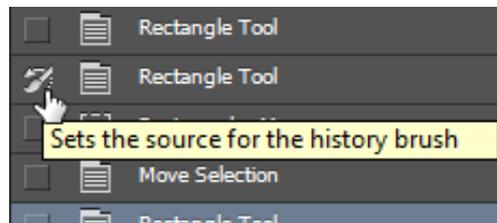


Undo in Various Programs

- See details for how Linear Undo works in **PowerPoint**
 - Good reference for expected behaviors
 - Note how selection changes as a result of undo
- Many programs have “unusual” designs for undo
 - Outlook – single level; undo delete – not selected (so hard to find)
 - Emacs editor – weird “switch directions” undo – forward/backwards
 - PhotoShop – 2 or 3 different undo mechanisms

Adobe PhotoShop

- History pane displays previous operations
- ^Z – one-level undo that toggles undo/redo – until V2019
- Also Shift-^Z, Alt-^Z - linear undo forwards and backwards
 - Redo list erased on new operations
- “History brush”
 - Select point in past and brush area – returns to the way it was in the past
 - Can’t “skip” operations
 - Is selective by region, but not by time



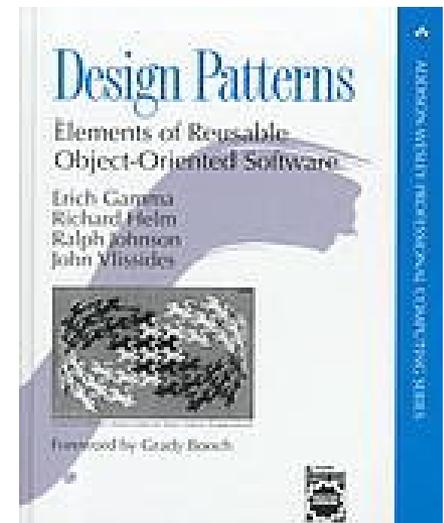


Undo implementations

- Need a central list of operations
- Where to store the old values?
 - With objects that are modified
 - E.g., a rectangle keeps track of all its former locations
 - Called “Memento Pattern” (Wikipedia)
 - But limited in kinds of editors – doesn’t work for text, paint
 - In a global list
 - But what to store for each operation?
 - Using the Command Object pattern
 - Store in the command object itself
 - Then it stays with the operation
 - No confusion about which parameters for which operation

Command Object Pattern

- Wikipedia: “An object is used to encapsulate all information needed to perform an action or trigger an event at a later time. This information includes the method name, the object that owns the method and values for the method parameters.”
- Was in original “Design Patterns” book (1994)
- Better separation between action and widgets
- Clearer place to store information needed for undo



HW 5 design for Command Objects

- Abstract class that all operations extend:
`class CommandObject`
- Methods for Execute, Undo, Redo etc., that specific commands override
- Variables for saved values **in the command object itself**

```
export default class CommandObject {  
  constructor(controls, addToUndoStack = true) {  
    this.undoHandler = controls;  
    this.addToUndoStack = addToUndoStack; // is this the kind of operations that is queued?  
    this.targetObject = undefined; // object this command affected  
    this.newValue = undefined; // new value used by the command  
    this.oldValue = undefined; // previous (old) value for the object  
  }  
}
```

Sub-classes of command object

- Create a subclass of CommandObject for each kind of command

```
import CommandObject from "./CommandObject";

export default class ChangeFillColorCommandObject extends CommandObject {
  constructor(undoHandler) {
    super(undoHandler, true);
  }
}
```

- Also: CreateObjectCommandObject, ChangeBorderColorCommandObject, ChangeBorderWidthCommandObject, etc.



Standard Process for using a Command Object

1. When the user clicks menu item (e.g., to change color), or starts an action (like create object), **allocate** a new command object of the correct type

```
curCmd = new ChangeFillColorCommandObject(undohdlr);
```

2. Call that object's `execute()` method, which will:
 - a) Save all the information needed to undo/redo/repeat the action later
 - b) Perform the action
 - c) Put this command object on the undo list
- Each kind of object will have a *different* execute method
- What does `ChangeFillColorCommandObject.execute()` need to store?

Provided Example:

ChangeFillColorCommandObject

- Command object that is used when change the fill color
- What to store?

Fill color:

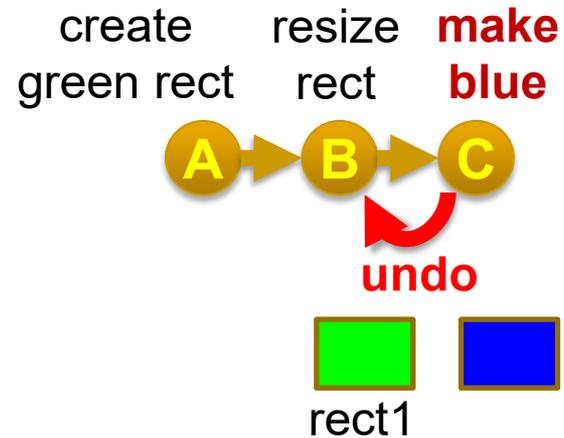
None



```
export default class CommandObject {
  constructor(controls, addToUndoStack = true) {
    this.undoHandler = controls;
    this.addToUndoStack = addToUndoStack; // is this the kind of operations that is queued?
    this.targetObject = undefined; // object this command affected
    this.newValue = undefined; // new value used by the command
    this.oldValue = undefined; // previous (old) value for the object
  }
}
```

Example:

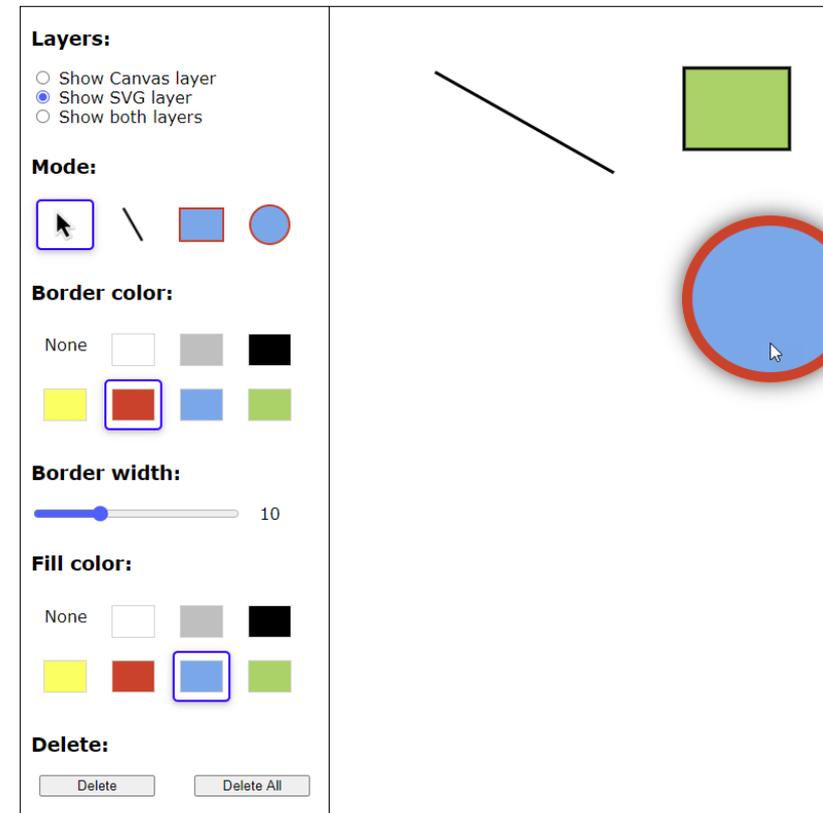
- SVG Change fill color: C
- Target object = `rect1`
- Old value = “green”
- New value = “blue”



```
class ChangeFillColorCommandObject extends CommandObject
```

Values

- `newValue` and `oldValue` often need to be an object with many values
- What to store for *create* in HW 3?
 - *All values used:*
 - Type (line/rect/ellipse)
 - Coordinates for create
 - Border color
 - Border width
 - Fill color
 - For SVG, can store the created object, but not for canvas
- Why can't you just get values from the palette?





Command Object Methods

- Execute / Do
 - The actual operation of the command, like to change the fill color
 1. Gets parameters from the global variables *and saves them in the Command Object itself*
 2. Execute the command
 3. Save the command object on the undo stack
 - Real operation will be a little more complicated
- For `ChangeFillColorCommandObject` :

```
execute() {
    if (selectedObj !== null) { // global variable for selected object
        this.targetObject = selectedObj; // save the object
        this.oldValue = selectedObj.fillColor; //get current color
        this.newValue = fillColorWidget.currentColor; //new color
        selectedObj.fillColor = this.newValue; //actually perform the change
        if (addToUndoStack)
            this.undoHandler.registerExecution({...this}); //load me onto undolist
            // which will also potentially remove pending undone commands
    }
}
```



Other Command Object Methods

- `canExecute()` – whether the execute method will work now
 - For change color – just if there is an object selected
- `canRepeat()` – whether repeat will work now
 - For change color – just if there is an object selected and a previous color

```
canExecute() {
    return selectedObj !== null;
}
canRepeat() {
    return (selectedObj !== null) && this.newValue;
}
```

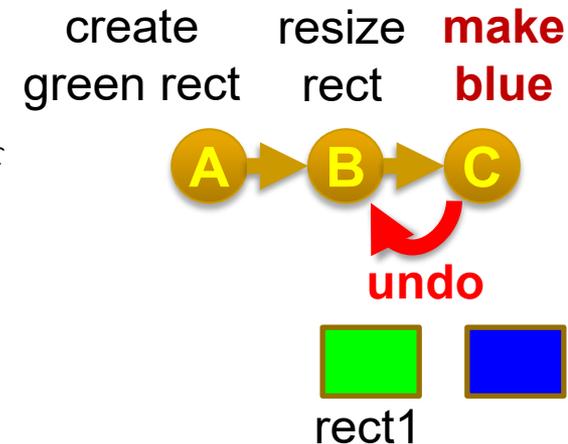
Undo & Redo

- Undo method – make the object have its old value

```
undo() {
  this.targetObject.fillColor = this.oldValue;
  // make sure this object is selected, which will
  // also fix the palette to show this object's color
  becomeSelected(this.targetObject);
  ** now fix the undo stack **
}
```

- Redo = undo the undo

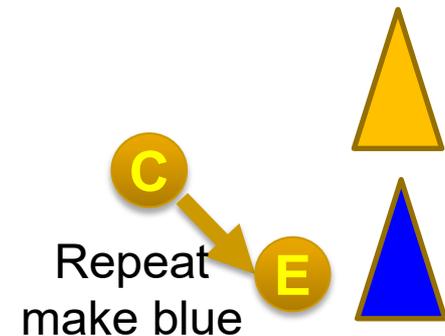
```
redo() {
  this.targetObject.fillColor = this.newValue;
  becomeSelected(this.targetObject);
  ** now fix the undo stack **
}
```



Repeat

- Apply same color to the currently selected object
 - Different object, so might have a different old color
- Remember, this operation is **added to the undo stack**
- Note: *not* the palette's current color – use saved `newColor`
- Need to allocate a new command object for repeat

```
repeat() {  
  if (selectedObj !== null) {  
    this.targetObject = selectedObj; // get new selected obj  
    this.oldValue = selectedObj.fillColor; //obj's current color  
    // no change to newValue - comes from operation that was copied  
    selectedObj.fillColor = this.newValue; //actually change  
    if (addToUndoStack)  
      this.undoHandler.registerExecution({...this});  
  }  
}
```



Change Color Control

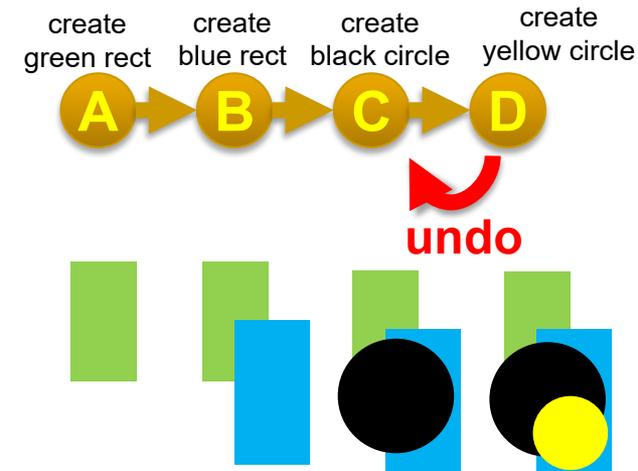
- When the user clicks on a color when an object is selected, that is different from the selected object's color, then:
 - Create a new `ChangeFillColorCommandObject`
 - Call its `execute` method

Fill color:



Implementing Undo for Canvas

- How can “undraw” an operation for the Canvas?
 - Note: *not* part of homework 5
- Just have to save a copy of the canvas before each operation
 - Redo can perform the operation again – **do not** need to store *both* before and after images
 - Optimization – save only the parts of the screen that changed
- Why not redo everything from the beginning each time?
 - Too slow in realistic situations



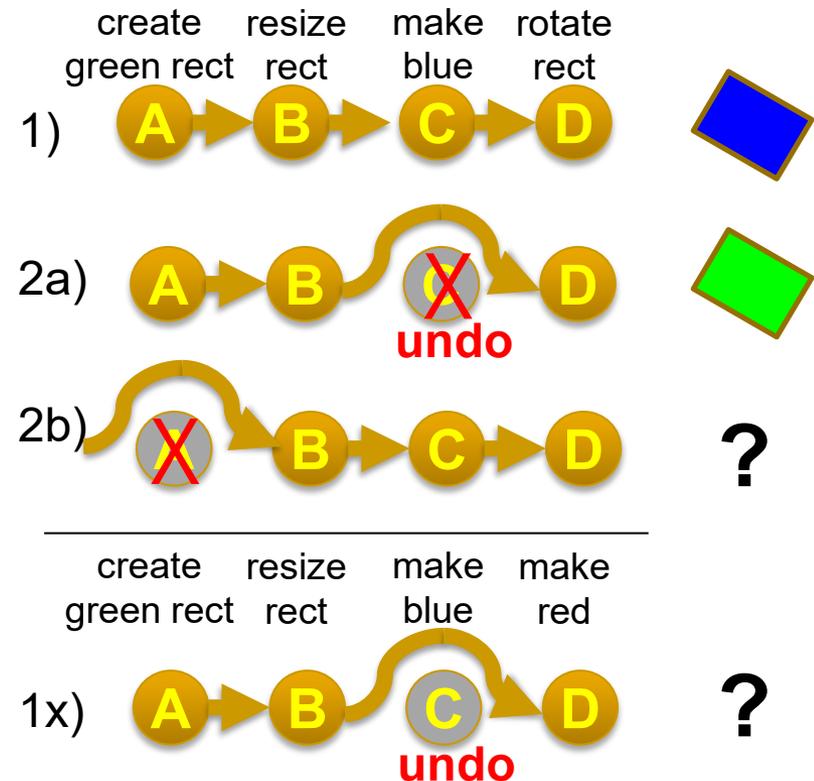


Linear Undo Handler

- Has to keep the undo stack, and keep track of which operation should be undone / redone / repeat
- Methods for
 - register a command object (after executed)
 - doUndo – call this when user hits the undo menu item
 - Undo Available? – controls greying out the undo menu item
 - Just checks if there is a command on the undo stack
 - doRedo, doRepeat, redo/repeat available?

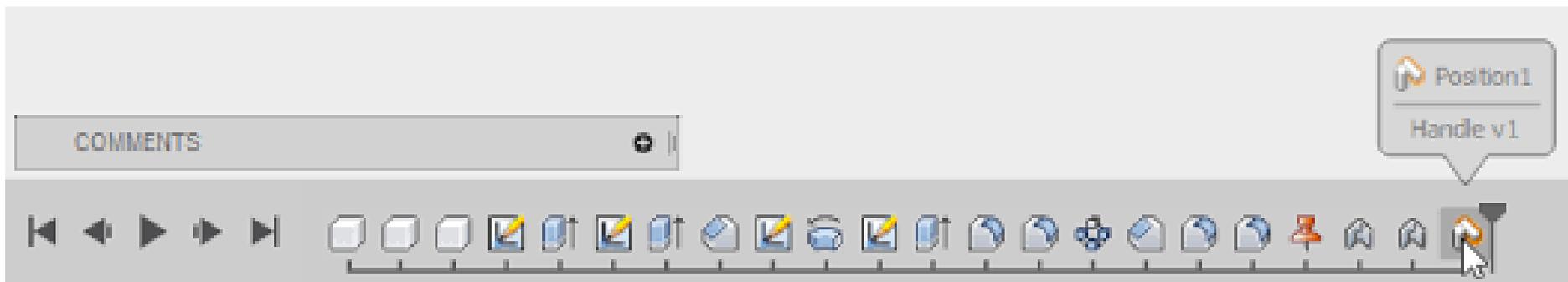
Advanced: Selective Undo

- Reach back into history and **select** which operation to undo
- “**Script model**”
 - As if that operation was just removed
- Often unclear what this means!



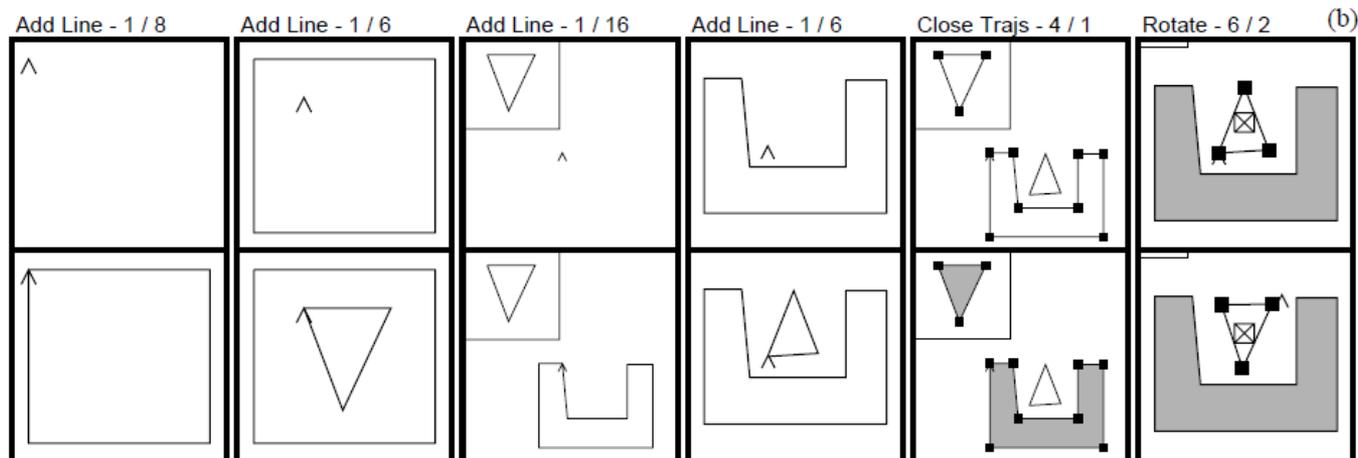
Timeline view in Fusion 360

- Fusion 360 (a CAD software) from AutoDesk
<https://www.autodesk.com/products/fusion-360/blog/master-the-timeline-browser-preferences/>
- Provides graphical timeline for undo
- Complete collection of every change made to your design
 - Selective undo (“suppress”) also affects later operations that depend on it



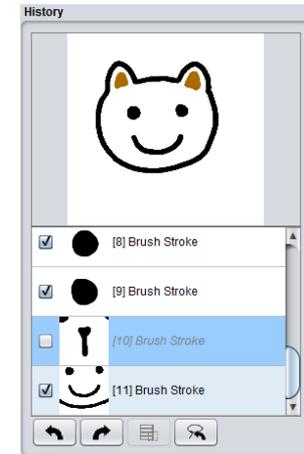
Kurlander's Graphics Histories

- Kurlander, D. and Feiner, S. Editable Graphical Histories. Proc. 1988 IEEE Workshop on Visual Languages. (Pittsburgh, Oct. 10-12, 1988). 127-134. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=18020&isnumber=662>
- [Video](#) (2:42)
- Before and after scenes for each operation
- Can undo back to any point
 - Can then *change things* and redo the operations afterwards
 - Basically, the “script” model of undo/redo



Aquamarine

- Brad A. Myers, Ashley Lai, Tam Minh Le, YoungSeok Yoon, Andrew Faulring, Joel Brandt, "Selective Undo Support for Painting Applications", Proceedings CHI'2015: Human Factors in Computing Systems, Seoul, Korea, April 18-23, 2015. pp. 4227-4236. <http://dl.acm.org/citation.cfm?doid=2702123.2702543>
- **Allowing Quick Undoing of Any Marks And Repairs to Improve Novel Editing**
- **Selective undo of past operations in a paint program using the **script model****
 - Can't use inverse model in paint because can't change affected pixels in current context
 - No dependencies among objects as there are in a drawing program
 - Issue: spatial dependencies:
 - Copy and paste
 - Flood fill (paint bucket)



Short Video: 0:30
Video: 4:35



Selective Undo by Region

- Selective Undo by Region
 - Regular linear undo but only for operations in the region
 - Avoids the ambiguities
 - Available in PhotoShop, our research system for code editing in Azurite:
YoungSeok Yoon and Brad A. Myers. “Supporting Selective Undo in a Code Editor,” *37th International Conference on Software Engineering, ICSE 2015*. Florence, Italy, May 16-24, 2015. 223-233 (volume 1). [pdf](#) and [video](#).



Direct Selective Undo Implementation

- Implementing direct selective undo not much harder than regular undo:
 - Allocates a new command object and adds to end of history list
 - Semantics is based on what the user would want
 - Undo the operation in a new context means to set the object back to its **previous value**
 - Selective Undo is enabled if object **is still available**
 - Undo of create is delete
- Redo the operation means to set the value of the object again;
 - redo of create = a new object
- Repeat = redo on new object

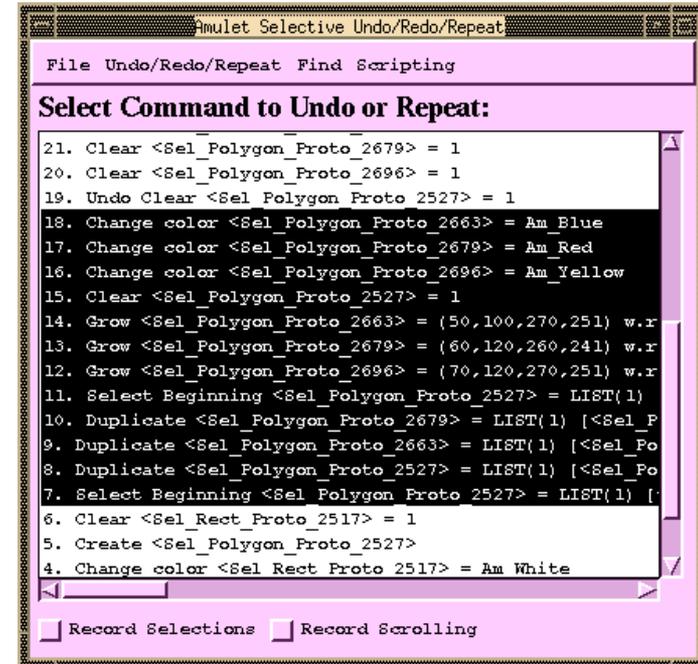
Scripting = “Topaz”

- Brad A. Myers. "Scripting Graphical Applications by Demonstration," *Proceedings CHI'98: Human Factors in Computing Systems*. Los Angeles, CA, April 18-23, 1998. pp. 534-541. [ACM DL](#), or [local pdf](#), and [YouTube video](#) or [local video](#) (3:09).

(Topaz)



- Select set of commands and specify that in a program
- Uses selective repeat
- Can parameterize actions
- Moving which object selected is recorded
 - Forwards, backwards, left, right, up, down, in, out
 - Search for object of a particular type or value
- Little or no change to application if it supports Selective Repeat



```

File Undo/Redo/Repeat Find Scripting

Select Command to Undo or Repeat:

21. Clear <Sel_Polygon_Proto_2679> = 1
20. Clear <Sel_Polygon_Proto_2696> = 1
19. Undo Clear <Sel_Polygon_Proto_2527> = 1
18. Change color <Sel_Polygon_Proto_2663> = Am_Blue
17. Change color <Sel_Polygon_Proto_2679> = Am_Red
16. Change color <Sel_Polygon_Proto_2696> = Am_Yellow
15. Clear <Sel_Polygon_Proto_2527> = 1
14. Grow <Sel_Polygon_Proto_2663> = (50,100,270,251) w.r
13. Grow <Sel_Polygon_Proto_2679> = (60,120,260,241) w.r
12. Grow <Sel_Polygon_Proto_2696> = (70,120,270,251) w.r
11. Select Beginning <Sel_Polygon_Proto_2527> = LIST(1)
10. Duplicate <Sel_Polygon_Proto_2679> = LIST(1) [<Sel_P
9. Duplicate <Sel_Polygon_Proto_2663> = LIST(1) [<Sel_Po
8. Duplicate <Sel_Polygon_Proto_2527> = LIST(1) [<Sel_Po
7. Select Beginning <Sel_Polygon_Proto_2527> = LIST(1) [
6. Clear <Sel_Rect_Proto_2517> = 1
5. Create <Sel_Polygon_Proto_2527>
4. Change color <Sel_Rect_Proto_2517> = Am_White

 Record Selections  Record Scrolling
  
```

Pictures for Scripting: Object Search

Search For

Search for an Object with Values:

Check the:

Type of Object

Location

LEFT

TOP

WIDTH

HEIGHT

X1

Y1

X2

Y2

Colors

FILL_STYLE

LINE_STYLE

Other

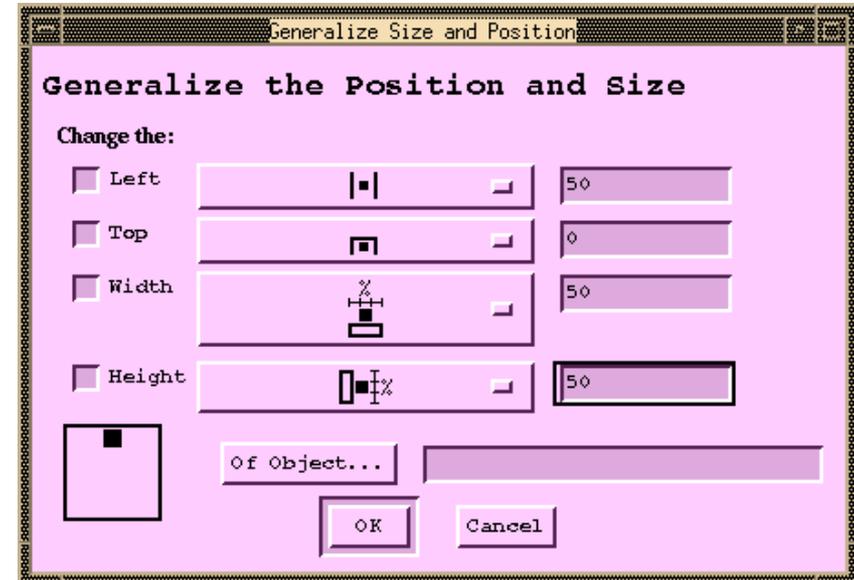
POINT_LIST

TEXT

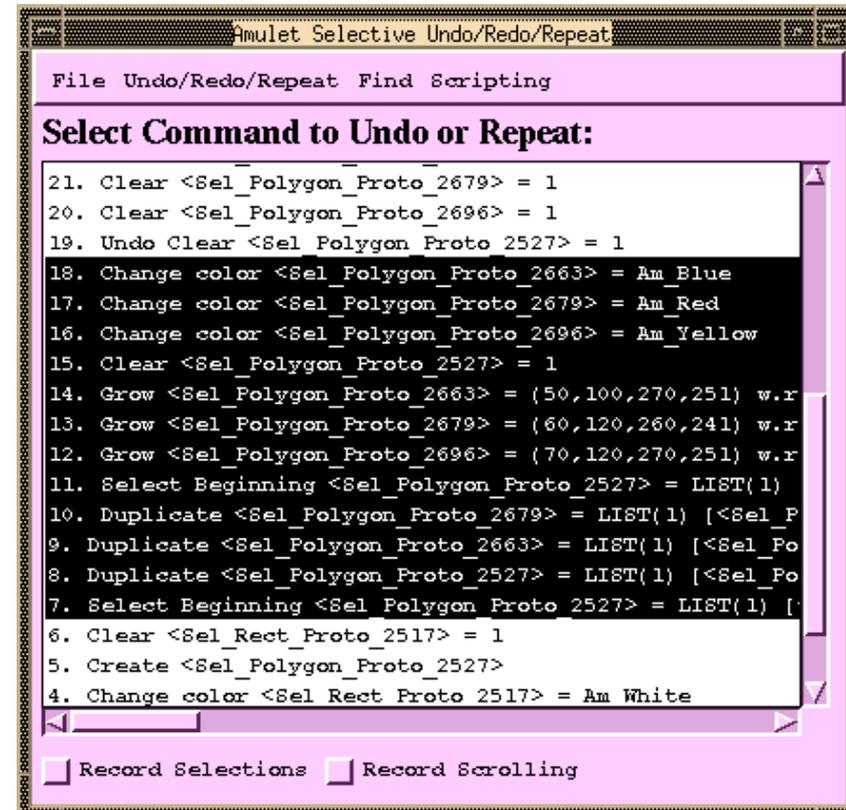
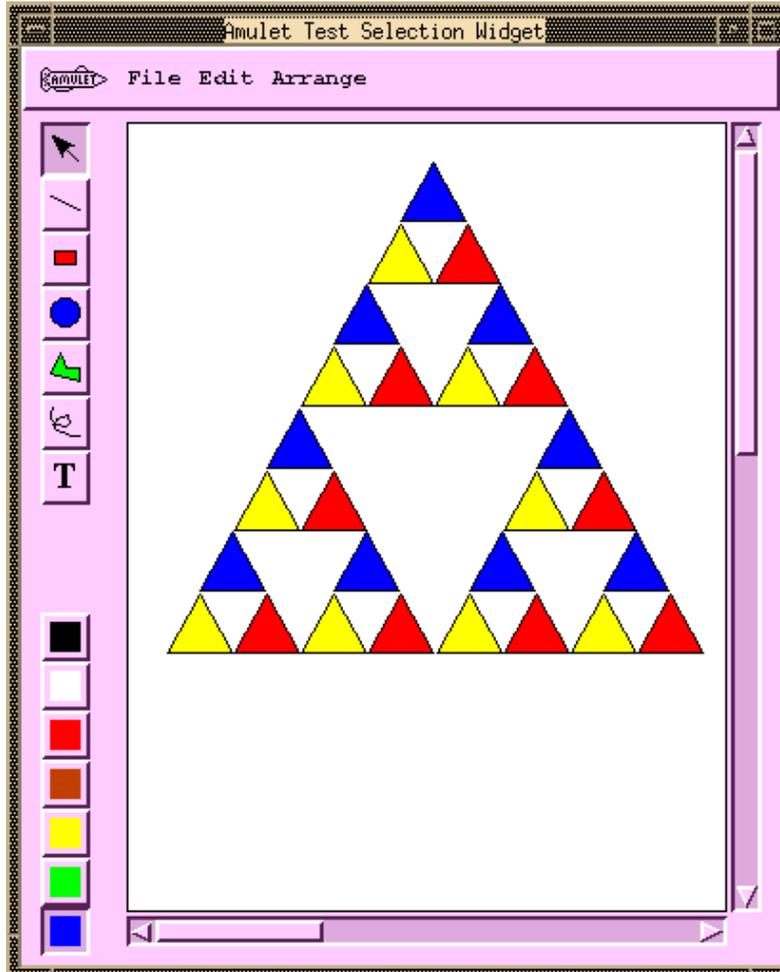
Find by Location:

<input type="button" value="Find up"/>	<input type="button" value="Find left"/>	<input type="button" value="Find Inside"/>
<input type="button" value="Find down"/>	<input type="button" value="Find right"/>	<input type="button" value="Find Outside"/>

Pictures for Scripting: Generalize Position / Size



Pictures for Scripting: Result

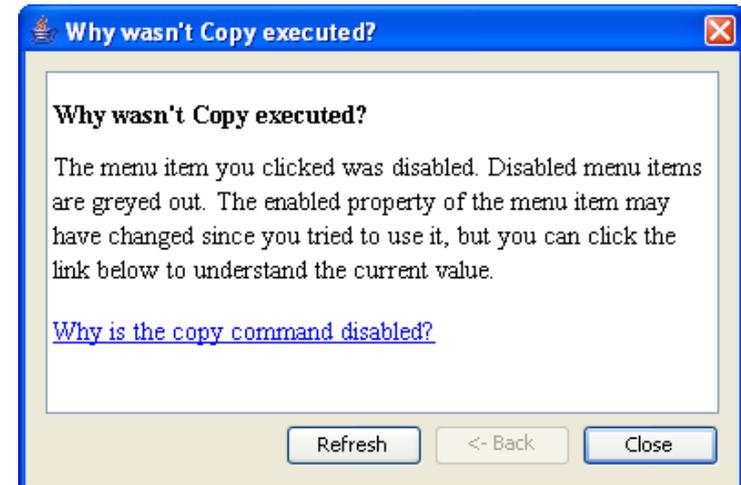
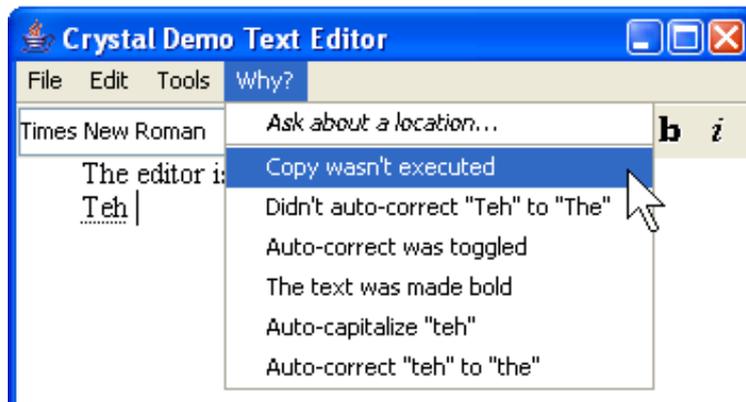


Multi-User Undo

- Required for Google Docs
 - Let's try: <https://tinyurl.com/SSUIUndo>
- if multiple users have overlapping selection regions and one user does Undo – what should be done?
 1. Undo the **globally** last operation
 2. Undo **that user's** last operation
 3. Undo the last operation in the **region** of the user's cursor
- Google Doc is somewhat random
- Old research on correct ways to handle this
 - Summary: it's complicated for text, easier for graphics

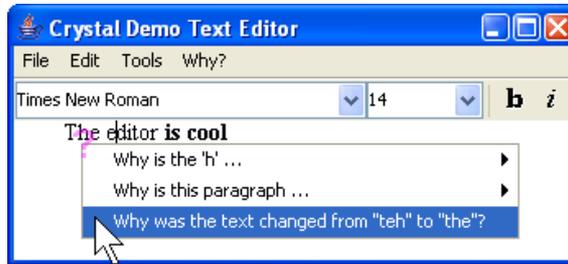
Using Undo History for “Why” Help

- **Crystal: Clarifications Regarding Your Software using Toolkit, Architecture and Language**
- Brad Myers, David A. Weitzman, A.J. Ko, and Duen Horng Chau, "Answering Why and Why Not Questions in User Interfaces," *Proceedings CHI'2006: Human Factors in Computing Systems*. Montreal, Canada, April 22-27, 2006. pp. 397-406. [pdf](#). See also [YouTube](#) or [local video](#)
- Help answer **why** things happen in regular desktop applications
- Lots of complexity in powerful features that people generally like
- Ask “Why” about what recently happened

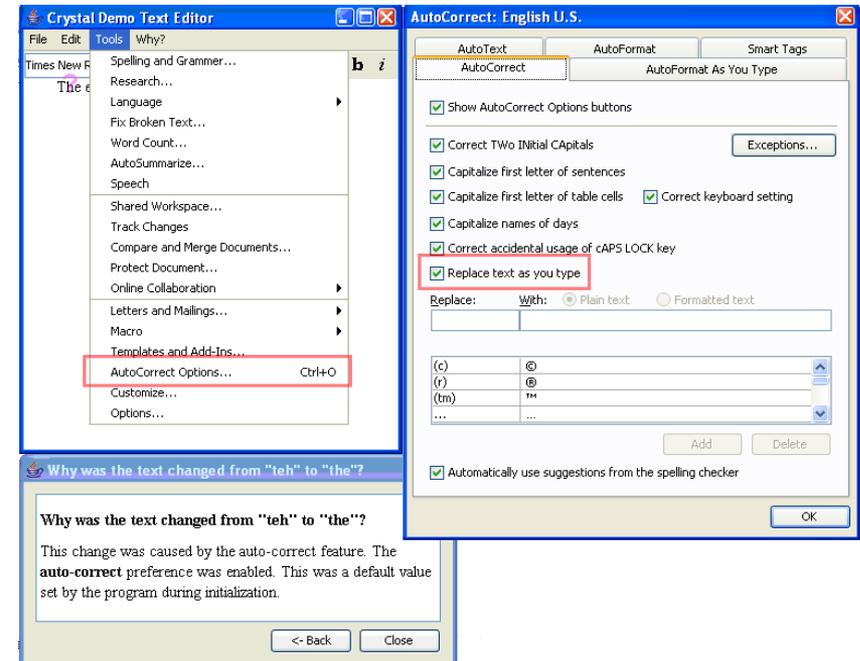


Crystal

- Or, ask Why about a location by clicking on objects, or whitespace



- Also can explain complexities like style inheritance, etc.





Crystal Implementation Overview

- (Full details in the paper)
- *Only a little more work than supporting Undo*
- “Command object” architecture for actions
 - Command objects stored on a list for undo
- Programmer adds back pointers from objects to the commands that changed them
- Add dependency information for mode variables
- Add special commands for actions *not* executed
- Add extra invisible objects for whitespace and deletions

- Correct Two Initial Capitals
- Capitalize first letter of sentences
- Capitalize first letter of table cells

Crystal Implementation, cont.

- Crystal framework then builds Why menus and answers automatically
- Crystal finds:
 - Objects under the mouse
 - Commands that affected those objects
 - User interface controls involved in those commands
- Programmer can annotate some commands to not include in menus
 - E.g., regular typing
 - Similar to heuristics for granularity of Undo