Structured Editors

CS 695 / SWE 699: Programming Tools
Fall 2023
Today

• Part 1 (Lecture)(~90 mins)

• 10 min break!

• Part 2 (Tech Talk)(20 mins)

• Part 3 (In-Class Activity)(40 mins)
Logistics

• HW1 due today

• HW 2 due in 2 weeks

• Anyone who has not yet signed up for a Tech Talk?
Overview

• Challenges in expressing and communicating computation

• Structured editors
What makes learning programming hard?

- What makes programming hard?
- Is the challenge thinking computationally?
- Or in understanding how to formally express computation in a programming language?

Slides partially adapted from Human Aspects of Software Development, Spring 2011, Lecture 11: How do people naturally think about computation? (Cyrus Omar)
Intuitions about programming language constructs

- Twelve fifth graders in a Pittsburgh public elementary school
- Equally divided amongst boys and girls
- No prior experience programming
- “The participants received no reward other than the opportunity to leave their normal classroom for half an hour and the opportunity to play a computer game for a few minutes.” 😊

Do this: Write a statement that summarizes how I (as the computer) should move Pacman in relation to the presence or absence of other things.

[Pane et al., 2001]
Intuitions about programming language constructs

**Programming Style**

- **54%** - production rules or event-based, beginning with *when, if* or *after.*
  - *When PacMan eats all the dots, he goes to the next level.*

- **18%** - global constraints
  - *PacMan cannot go through a wall*

- **16%** - declarations/other
  - *There are 4 monsters.*

- **12%** - imperative
  - *Play this sound. Display this string.*

Do this: Write a statement that summarizes how I (as the computer) should move Pacman in relation to the presence or absence of other things.

[Pane et al., 2001]
Intuitions about programming language constructs

**Modifying State**
- 61% - behaviors were built into the entity, e.g. OO
  - *Get the big dot and the ghost will turn colors...*
- 20% - direct modification of properties
  - *After eating a large dot, change the ghosts from original color to blue.*
- 18% - other

Do this: Write a statement that summarizes how I (as the computer) should move Pacman in relation to the presence or absence of other things.

[Pane et al., 2001]
Intuitions about programming language constructs

OR

- **63%** - boolean disjunction
  - To make PacMan go up or down, you push the up or down arrow key

- **20%** - clarifying or restating the prior item
  - When PacMan hits a ghost or a monster, he loses his life.

- **18%** - meaning otherwise

- **5%** - other

Do this: Write a statement that summarizes how I (as the computer) should move Pacman in relation to the presence or absence of other things.

[Pane et al., 2001]
Intuitions about programming language constructs

Insertion into a data structure

- **75%** - no mention of making room for new element
  - *Put Elton John in the records in alphabetical order*
- **16%** - make room for element before inserting it
  - *Use the cursor and push it down a little and then type Elton John in the free space*
- **6%** - make room for element after inserting it
- **4%** - other

[Pane et al., 2001]
Is natural language programming a solution?

A **difficult proposition** – natural language is complex and imprecise

- Computer and programmer do not have a shared context [Nardi, 1993]; programmers cannot use rules of cooperative conversation [Grice, 1975]
- Not obvious where the computer’s limits are

Novices **can use formal languages** if designed carefully [Bruckman and Edwards, 1999]

- Describing the instructee as a naïve alien increases precision of instructions [Galotti, 1985]
- Anthropomorphizing computers is counterproductive [du Boulay, 1989]
Goal: Gentle Slope Systems

Minimalist Learning Theory

• Choose an action-oriented approach
  • Provide an immediate opportunity to act, encourage self-directed exploration & innovation, prioritize user’s goals over delivery of information

• Anchor the learning tool in the task domain
  • Use real tasks as instruction, organize instruction around task steps

• Support error recognition & recovery
  • Prevent mistakes when possible, provide error information that offers not only detection but ‘on-the-spot’ diagnosis & recovery

• Support reading to do, study, locate
  • Make instructions brief & self-contained to support different levels of engagement

Problem frames

- Developers approaching messy problem interpret it with a *frame*
- Imposes boundaries on what learners will consider
Simplify typing code

• If key barrier is syntax, reduce challenge of working with syntax
• Reduce constructs in programming language
• Simplify constructs in programming language
• Eliminate possibility of syntax errors
Beginners All-Purpose Symbolic Instruction Code (BASIC, 1963)

- Support a subset of instructions & remove unnecessary syntax
- Offer rapid feedback through interpreted language
- Offer simplified statements w/ 3 parts: line number, operator, operands

```
FORTRAN:
  do 30 i = 1, 10
    m = m + I
  30 continue

BASIC:
  100 FOR I = 1 TO 10
  110 LET S = S + I
  120 NEXT I
```

Figure 2. A for loop to compute the sum of the numbers from 1 to 10 written in FORTRAN and BASIC.

J.G. Kemeny and T. Kurtz, Dartmouth College, 1963
LOGO (1967)

- Supports manipulating turtle to draw pictures
- Move forward 10 spaces
- Turn left 90 degrees
- Offers dialect of LISP with less punctuation
- Supports creating music, translating languages, and much more

Seymour Papert, MIT, 1967
Interacting with objects

- Enable users to create objects & rules on how objects behave

Figure 4. A view of the My Magic Castle courtyard. The user is creating the rule "Nicky should dance when it meets the horse."

Structured editors

Cornell Program Synthesizer/
Synthesizer Generator 1981

```plaintext
main

program <identifier>;
var
   <identifier> : <type>;
begin
   i { NOT DECLARED } := 1;
   while <exp> do
      <statement>
end.

Positioned at exp
```
The cursor, representing the current node of the tree, is displayed as dashed boxes. Unfilled or nodes, called meta-nodes, are displayed as $\text{CLASS}$ where \text{CLASS} defines the language constructs that may replace the node.

Screen A shows an existing program. In screen B, an IF node replaces $\text{stmt}$ as the cursor moves to the first new meta-node. Applying cursor-out increases the focus of attention as shown in screen C. In screen D, the cursor moves to the previous statement in the list. Deleting the PRINT statement yields screen E. Screen F shows the state after the list of statements is extended.

Figure 2-4: Sample ALOE Session
• Initially programmers conceive of a program as structure; then they transform their mental picture of structure into text; and finally a parser transforms the text back into structure. We believe that the user benefits greatly when they are relieved from the first transformation of structure into text. This has the added benefit of allowing us to eliminate, either totally or partially, the need for parsing as the user develops ALOE trees directly.

• Our editors further provide a good mechanism for replacing the traditional {edit, compile, link, debug) cycle with a more natural {edit, execute) cycle; indeed, the LOIW system just described is based on this simple tools cycle. We believe that this is just one example of situations where traditional mechanisms can be replaced by mechanisms that are more suitable to users.
Scratch 2005
Challenges addressed by blocks based editors
Learning programming vocabulary

- How to express computation in code?
- Learning 100-200 new words and understanding concepts behind them is overwhelming
- Much easier to select options from a palette
- Recognition easier than recall
Palettes vs. Autocomplete

• Both make it easier to find a command
• But autocomplete
  • Requires user to already remember part of what they're looking for
• Lacks hierarchic organization of similar functions
Filling in arguments

• Remembering order, type, and valid argument values is hard
• Block languages have
  • automatically generated holes showing what arguments are expected
  • useful default operands
  • drop down menus and specialized editors to create and change operands
  • extra text explaining operand meaning
Reading syntax

- Textual languages have lots of syntax that is unclear to newcomers
- Showing the structure of the code makes visible the chunks that experts eventually use to understand code
Following syntax

• Developers, particularly novices, make syntax errors all the time
  • Might be slip - intended to do something else
  • Or mistake - didn't know the right way to do it
• Structured editors make syntax error impossible
  • Each type has a distinct shape
  • Commands connect vertically with nubs and notches
  • Expressions are smooth
Tinkering

- In textual languages, hard to know what the output of an expression is
- In blocks languages with liveness, can execute fragments just by clicking on them
- (More on live programming later)
Intuitive words and concepts

• Textual languages rely on keywords that are incomprehensible to novices
  • e.g., for, !=

• Blocks based languages choose keywords that better leverage real world concepts and ideas
  • e.g., repeat, unequal
Example reuse

• Professional reuse code all the time, but adapting can be hard
• Blocks based languages can support this process, finding dependencies required to make a line of code run
Limitations of blocks based editors
Higher viscosity for small edits

• Writing \( \frac{a}{2} + \frac{b}{2} \) requires a number of steps to find and drag blocks for 3 arithmetic operators and fill in with variables and numbers

• Faster, for an experienced programmer, with a textual language

• Rearranging expression from \( \frac{a}{2} + \frac{b}{2} \) to \( \frac{a+b}{2} \) requires more steps to change the structure than similar textual structure
Other disadvantages

- Low density - blocks take up more space
- Search - hard to find code
- Source control - may not work without textual representation
Hybrid editors

- Can combine textual and blocks based editors
- Text-style entry of blocks - enter blocks as text, choosing blocks through autocomplete
- Can do drag and drop for bigger structure, and text for smaller
- Bidirectional mode switching
- Switch between blocks and text, edit in either
Examples
Many domains use blocks programming
Figure 9. The SPARQL playground is a blocks-based query execution tool that provides blocks for constructing queries of RDF data. Query results (bottom) are also provided as blocks, and they can be dragged to build into other queries.

```sparql
from http://dbpedia.org/sparql
where
dbr : The_Beatles is a
&
dbo : formerBandMember
order by & limit to first 5 rows

obj

dbpedia : Paul_McCartney

dbpedia : George_Harrison

dbpedia : John_Lennon

dbpedia : Ringo_Starr
```
Robot automation

When ▶️ is pressed:
- Open Hand.
- Move to Above Pickup �.levels
- Move to Pickup 🔄
- Close Hand.
- Move to Above Pickup 🔄
- Move to Handover A 🔄

Wait for each other.
- Close Hand.
- Wait for each other.
- Open Hand.
- Wait for each other.

Move to Above Place 🔄
- Move to Place 🔄
- Open Hand.
- Move to Above Place 🔄
Thomas Ball and Stefania Druga

From player to creator: Designing video games on gaming handhelds with Microsoft TileCode

Now on demand
10 min break
Tech Talk: Scratch
In-Class Activity

• In groups of 2, build pong in Scratch

• https://scratch.mit.edu/projects/editor/?tutorial=getStarted