An Introduction to PEBL: The Psychology Experiment Building Language

Shane T. Mueller, Ph.D.
Indiana University
http://pebl.sf.net

34th Annual Meeting of the Society for Computers in Psychology
Minneapolis, MN
Nov. 18, 2004

Motivations for PEBL

- Many solutions exist for experiment design
- None were right for me.
  - Everyone has solutions (6+ different ones used by people in my lab).
  - I use 3-4 different methods myself
  - I primarily use GNU/Linux, but my lab uses other platforms.
  - Many of the best solutions have restrictive licensing constraints that I would prefer not deal with.

PEBL Goals

- Design a system that is:
  - Free (redistributable without cost or licensing)
    - I have megabytes of documents/programs that will cost me hundreds of dollars to access again;
    - Sharing difficult (network effects work against user)
  - Source Code Availability
    - Transparency, modifiability, accessibility
  - Cross-platform
    - Further avoid lock-in and enable better sharing/flexibility
  - A simple scripting language
    - Avoid high level (though powerful) idioms

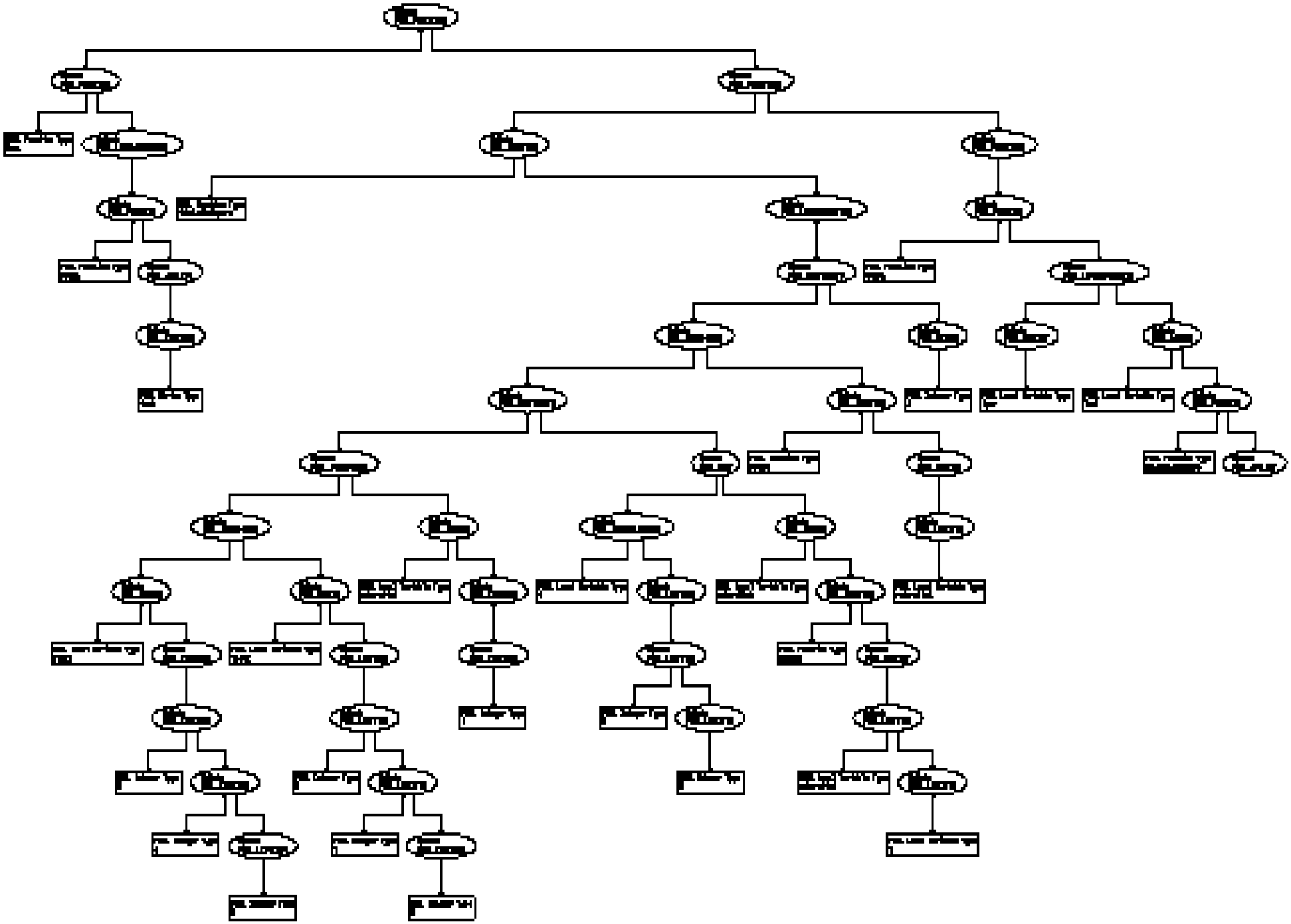
PEBL: Psychology Experiment Building Language

- Planning and development for the past 3 yrs.
- Still in early stages (You can have an impact)
- Initial goals: create general-purpose system that is tailored for creating basic psychology experiments
  - Present Text
  - Present Images
  - Present Sounds
  - Keyboard Input
  - Control of Timing
  - Data Recording

Technical Details

- PEBL written primarily in C++
- Parser/lexer constructed with flex/bison.
- Parser creates a p-node execution tree from a text program (not a text-based interpreter)
- Evaluator steps through p-node structure

```
define Start (lParameter )
{

  ##Make initial objects needed for display.
  gWindow <- MakeWindow()
  gPic <- MakeImage("pebl.bmp")

  gSmiley <- MakeImage("smiley-big.png")
  gFrowney <- MakeImage("frowney-big.png")
  AddObject(gPic, gWindow)
  AddObject(gSmiley, gWindow)
  AddObject(gFrowney, gWindow)

  Move(gSmiley, 300,50)
  Move(gFrowney, 300,50)

  lBG <- MakeColor("GREY")
  lFG <- MakeColor("dodgerblue3")
  lFont <- MakeFont("Vera.ttf",0,22,lFG,lBG,1)
  lText <- MakeLabel("Hello World", lFont)

  ##Initially place text here
  Move(lText, 300,200)
  ##Add text to the window.
  AddObject(lText,gWindow)
  ##Nothing appears until you draw!!!
  Draw()

  ##Set the positions of things
  x <- 0
  y <- 50

  ##The following loop enables simple animation.
  i <- 1
  while(i <= 350)
    {
      x <- x + 2
      y <- y + 1
      Move(lText,x, 25 * Sin(x / 25) + 300 - x/3)
      Move(gPic, x, y)

      if(Mod(i,50) == 0)
        {
```



Technical Details

- PEBL written primarily in C++
- Parser/lexer constructed with flex/bison.
- Parser creates a P-node execution tree from a text program (not a text-based interpreter)
- Evaluator steps through p-node structure
- Heavy use of the C++ STL
- Designed to allow different platforms
- Currently, supports libSDL front-end, a LGPL cross-platform game library.
- 15,000 lines of C++ (David A. Wheeler's sloccount)

PEBL Data Environment

- All data types are handled in a memory-managed variant data type.
- Complex objects (images, text, fonts, etc.) are managed with a counted-reference system.
- Global and local variables are available.
- Lists are currently the only sequence type.

PEBL's Focus

- Simple and forgiving syntax focusing on readability
- Avoid difficult but powerful programming idioms (Object-oriented, Pointers, Recursion)
- Use objects with few options
  - Avoid the numerous properties/options available (and useful) in general-purpose languages
- Statements separated by new lines
- Naming conventions enforced in syntax
  - Local/global status designated by initial letter
  - Function names begin with Capitals

Example Function

```
define Mean (listOfNums)
{
  # Initialize variables
    sum <- 0  #Variables begin with lowercase
    len <- 0

    loop(i,listOfNums) #Simple looping control
    {
        sum <- sum + i
        len <- len + 1
    }
   return sum/len
}
```

PEBL Function Library

- Philosophy: Provide easy access to anything commonly done in experiments
- 130+ Functions
- Core functions written in C++
- Increasing number written in PEBL itself
- Randomization, counterbalancing, etc.
  - `DesignFullCounterbalance()`
  - `DesignLatinSquare()`
- Response Collection
  - `WaitForKeyPress("X")`
  - `WaitForAnyKeyPress()`
  - `WaitForListKeyPress(["A","Z"])`

Performance: Speed of Variable Access

- Primitive variables created/accessed rapidly
- Use an STL map association structure to store variable values.

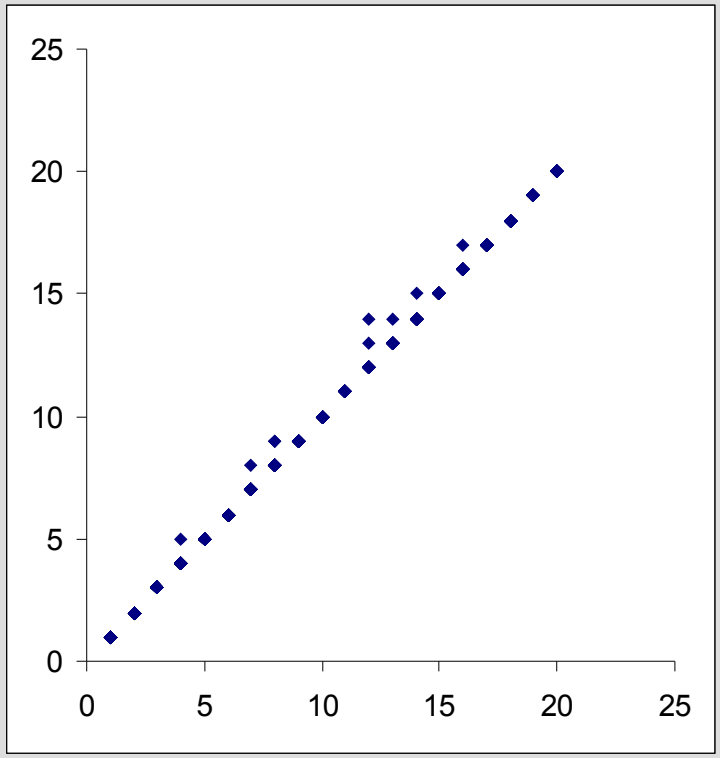
- 100,000 lookup/store operations in 810 ms
  - 123/ms
- Sufficiency of a map structure may diminish as number of stored variables increases (O (log(N)).

Performance: Evaluator Efficiency

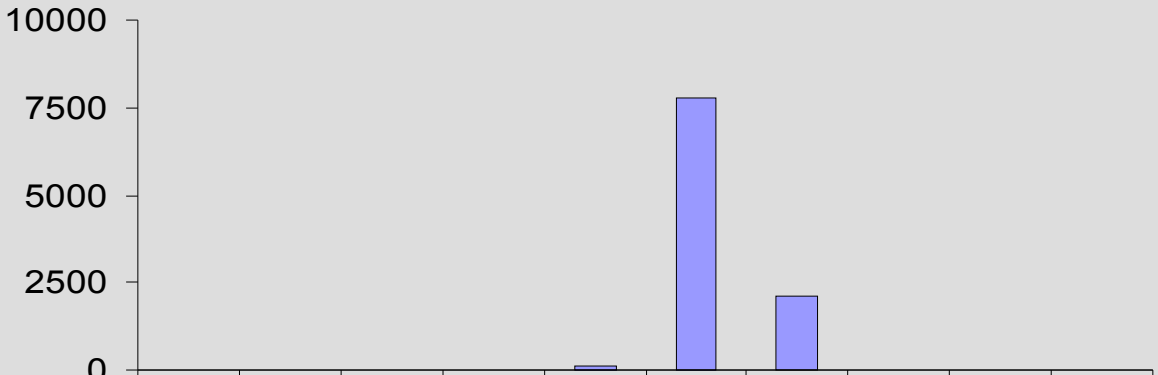
- 100,000 addition/multiplication ops in 150 ms
  - 667/ms
- 100,000 file write ops in 17.3 sec
  - 5.8/ms
- 100,000 list creations in 4.7 sec
  - 21/ms
- 100,000 list appends in 12.1 sec
  - 8.2/ms

Performance: Timing

- Delays/waits performed with busy wait
- Standard interrupt-based waiting (sleep()) would be better but normally is no better than 10ms accuracy
- On 10,000 programmed delays between 1 and 20 ms, 9 missed by 1-2 ms (could be partly measurement error)



Performance: Video Display



	61	62	63	64	65	66	67	68	69	70
Count	1	1	1	0	115	7777	2098	6	0	1

- Tested on 60 hz laptop
- 1000 iterations of a 4-frame animation
- With proper drivers/video cards, Draw() function blocks until next vertical refresh
- 4 consecutive draws should take 4 refresh cycles
- In 10,000 trials @ 60 hz (66.66 ms)

Performance: Summary

- Accuracy is probably good enough for most experiments people do.
- For experiments with high demands (presentation duration, response accuracy), current versions of PEBL may be insufficient.
- For these experiments, a real-time OS and special-purpose presentation and data acquisition hardware is probably necessary as well.

Limitations

- Timing and video display are good when you are careful and lucky
- Input is handled through a parallel thread in the library, imposing some delay in processing key events
- Currently available only on Microsoft Windows ™ and Linux (no Mac support)
- Underdeveloped function library in many areas (visual/images, sound)
- Keyboard is only input device currently supported
- No graphical drag-n-drop or menu-based interface designer or development environment

The Future

- OpenGL front end may offer better support for video timing
- Take advantage of real-time OS features on supported platforms
- Audio Subsystems to improve playback capabilities (PortAudio and/or Jack).
- Programming front-end for a cognitive modeling system (EPIC).
- Whatever potential users need and are willing to contribute.

Information

- PEBL Website: http://pebl.sf.net
- Current Release: 0.3
- Contact: stmuelle@indiana.edu or pebl-list@lists.sf.net