Engaging Students by Making Computer Science Concepts Come Alive

Susan H. Rodger
Duke University
November 21, 2014
Indiana University

Supported by NSF Grants DRL-1031351, DUE 1044191, CRA, and IBM Faculty Awards
Engaging Students in CS - Back in 1989

- New Assistant Professor at Rensselaer Polytechnic Institute

Rensselaer

- New Course: Combined automata theory with CS1 and CS2 (data structures)

- Student wanted feedback on all the answers in the book!
Different Types of Learners

• Learning Styles
  – Visual Learners
    • Learn through seeing
    • Learn best from visual displays
  – Auditory Learners
    • Learn through listening
    • Learn best through verbal lectures, discussions
  – Kinesthetic Learners
    • Learn through moving, doing and touching
    • Learn best through hands-on approach
How do you reach all three types?

• You must do all three!
  – Provide pictures, diagrams
  – Discuss what you are doing
  – Provide activities for trying it
Outline

• CS Concepts Come Alive with Software
  • Automata Theory with JFLAP
  • Learning Programming with Alice
  • Algorithm Animation

• Challenges in Designing Educational Software

• CS Concepts Come Alive in other ways
  • Manipulatives
  • Group Activities
  • Edible CS
Learner Engagement Taxonomy with visualization software

• Different forms of Learner engagement
  – No Viewing
  – Viewing
  – Responding
  – Changing
  – Constructing
  – Presenting

• ITiCSE Working Group Report 2002 (Naps et al.)
Outline

• **CS Concepts Come Alive with Software**
  – Automata Theory with JFLAP
  – Learning Programming with Alice
  – Algorithm Animation

• **Challenges in Designing Educational Software**

• **CS Concepts Come Alive in other ways**
  – Manipulatives
  – Group Activities
  – Edible CS
Automata Theory

• Foundations of Computer Science (theory)
• Formal Languages
  • Finite state machines, Turing machines, grammars

• Proofs

• Motivation
  • How does a compiler work?
  • How does it recognize the syntax errors in your program?
Formal Languages and Automata Theory

• Traditionally taught
  • Pencil and paper exercises
  • No immediate feedback

• More mathematical than most CS courses
• Less hands-on than most CS courses
• No programming? Unlike most other CS courses
Why Develop Tools for Automata?

<table>
<thead>
<tr>
<th>Textual</th>
<th>( {q_0, q_1, q_2}, {a, b}, \delta, q_0, {q_2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \delta = {(q_0, b, q_0), (q_0, a, q_1), (q_1, a, q_0), (q_1, b, q_2), (q_2, a, q_1)} )</td>
</tr>
<tr>
<td><strong>Tabular</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>\begin{tabular}{c</td>
</tr>
<tr>
<td><strong>Visual</strong></td>
<td>![Visual Diagram]</td>
</tr>
<tr>
<td><strong>Interactive</strong></td>
<td>![Interactive Diagram]</td>
</tr>
</tbody>
</table>
Overview of JFLAP

• **Java Formal Languages and Automata Package**

• Instructional tool to learn concepts of Formal Languages and Automata Theory

• Topics:
  – Regular Languages
  – Context-Free Languages
  – Recursively Enumerable Languages
  – Lsystems

• With JFLAP your creations come to life!
Thanks to Students - Worked on JFLAP and Automata Theory Tools

- NPDA - 1990, C++, Dan Caugherty
- JFLAP - 1996-1999, Java version
  Eric Gramond, Ted Hung, Magda and Octavian Procopiuc
- Pâté, JeLLRap, Lsys
  Anna Bilska, Jason Salemme, Lenore Ramm, Alex Karweit, Robyn Geer
- JFLAP 4.0 – 2003, Thomas Finley, Ryan Cavalcante
- JFLAP 6.0 – 2005-2008 Stephen Reading, Bart Bressler, Jinghui Lim, Chris Morgan, Jason Lee
- JFLAP 7.0 - 2009 Henry Qin, Jonathan Su
- JFLAP 8.0? – 2011-14 Julian Genkins, Ian McMahon, Peggy Li, Lawrence Lin, John Godbey

Over 20 years!
DFA Example

• Build a deterministic finite automaton (DFA) to recognize **binary numbers** with an **even number of 1s** that are an **even number**.

• Only use symbols 0 and 1

• Binary numbers: 0, 1, 10, 11, 100, 101, 110, 111, ...

• When is a binary number an even number?
  – Ends in 0

• Which strings should be accepted?

• 11010, 10010, 1111, 10100

  No, odd    Yes    No, ends    Yes
  no. of 1’s   Yes     In 1
Build with JFLAP
Simulation on 1101010
Accepts Input!

11010110
Add meaning to states!

- $q_0$ with transitions 1 and 0 to $q_1$ and $q_2$, respectively.
- $q_1$ with a transition 0 to $q_0$.
- $q_2$ with transitions 0 and 1 to $q_3$ and $q_4$, respectively.
- $q_3$ with a transition 0 to $q_4$.
- $q_4$ with a transition 0 to $q_2$. 

Note: $q_1$ is labeled 'only one 0', indicating a condition for state transitions.
Test Multiple Inputs

The diagram shows a finite automaton with states q0, q1, q2, q3, and q4. The transitions are as follows:

- From q0, on input 0, move to q2. States q0 and q2 are labeled with "odd number of 1's." On input 1, move to q3. State q3 is labeled with "even number of 1's, ends in 1." On input 0, return to q0.
- From q1, on input 0, move to q2. On input 1, move to q4. State q4 is labeled with "even number of 1's, ends in 1." On input 0, return to q1.

The table shows the input and corresponding results:

<table>
<thead>
<tr>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Reject</td>
</tr>
<tr>
<td>111</td>
<td>Reject</td>
</tr>
<tr>
<td>1010</td>
<td>Accept</td>
</tr>
<tr>
<td>10110</td>
<td>Reject</td>
</tr>
<tr>
<td>101</td>
<td>Reject</td>
</tr>
<tr>
<td>1100</td>
<td>Accept</td>
</tr>
<tr>
<td>110110</td>
<td>Accept</td>
</tr>
</tbody>
</table>
Another Example: Grammar

• Grammar – set of replacement rules to define a language

• Examples:
  – Grammar for English
    • defines English sentences
  – Grammar for Python programming language
    • defines syntactically correct programs
  – Grammar for a formal language (simpler)
Grammar for \(a^n b^n c^n\)

- Unrestricted grammar
- Generates strings with an equal number of a’s, b’s, c’s
- a’s first, then b’s, then c’s
- Example strings can derive:
  - abc
  - aabbcc
  - aaabbbccc
  - aaaabbbbbbcccc
  - aaaaaabbbbbcccccc
  - ...

---

<table>
<thead>
<tr>
<th>S</th>
<th>(\rightarrow AX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(\rightarrow aABbc)</td>
</tr>
<tr>
<td>A</td>
<td>(\rightarrow aBbc)</td>
</tr>
<tr>
<td>B</td>
<td>(\rightarrow \lambda)</td>
</tr>
<tr>
<td>B</td>
<td>(\rightarrow bB)</td>
</tr>
<tr>
<td>B</td>
<td>(\rightarrow D)</td>
</tr>
<tr>
<td>D</td>
<td>(\rightarrow EXc)</td>
</tr>
<tr>
<td>D</td>
<td>(\rightarrow bD)</td>
</tr>
<tr>
<td>D</td>
<td>(\rightarrow cD)</td>
</tr>
<tr>
<td>a</td>
<td>(\rightarrow aB)</td>
</tr>
<tr>
<td>b</td>
<td>(\rightarrow Eb)</td>
</tr>
<tr>
<td>c</td>
<td>(\rightarrow Ec)</td>
</tr>
</tbody>
</table>
Example Derivation for aabbccc

S → AX

rule: S -> AX
Example Derivation for aabbcc

S → AX
→ aAbcX

rule: S -> AX
rule: A -> aAbc
Example Derivation for aabbcc

S → AX  
→ aAbcX  rule: A → aAbc  
→ aABbccX  rule: A → aBbc

NOTE: We have generated the correct symbols, aabcbc, but they are in the wrong order!
Example Derivation for aabbcc

\[
\begin{align*}
S & \rightarrow AX & \text{rule: } S \rightarrow AX \\
& \rightarrow aA\text{bc}X & \text{rule: } A \rightarrow aA\text{bc} \\
& \rightarrow aaB\text{bc}cX & \text{rule: } A \rightarrow aB\text{bc} \\
& \rightarrow aabB\text{cbc}cX & \text{rule: } Bb \rightarrow bB
\end{align*}
\]
Example Derivation for aabbcc

\[ S \rightarrow AX \]  \hspace{1cm} \text{rule: } S \rightarrow AX

\[ \rightarrow aAbccX \]  \hspace{1cm} \text{rule: } A \rightarrow aAbcc

\[ \rightarrow aaBbccX \]  \hspace{1cm} \text{rule: } A \rightarrow aBbcc

\[ \rightarrow aabBbccX \]  \hspace{1cm} \text{rule: } Bb \rightarrow bB

\[ \rightarrow aabDbccX \]  \hspace{1cm} \text{rule: } Bc \rightarrow D

Note: the D absorbed the c!
Example Derivation for aabbcc

S → AX
    → aAAbcX
    → aaAbbcX
    → aabBbcbcX
    → aabDbcX
    → aabDbcX

rule:  S -> AX
rule:  A -> aAAbc
rule:  A -> aAbbc
rule:  Bb -> bB
rule:  Bc -> D
rule:  Db -> bD
Example Derivation for aabbcc

\[
\begin{align*}
S & \rightarrow AX \quad \text{rule: } S \rightarrow AX \\
\rightarrow aAAbcX & \quad \text{rule: } A \rightarrow aAAbc \\
\rightarrow aaBbcbcX & \quad \text{rule: } A \rightarrow aBbc \\
\rightarrow aabBbcbcX & \quad \text{rule: } Bb \rightarrow bB \\
\rightarrow aabBcbbcX & \quad \text{rule: } Bc \rightarrow D \\
\rightarrow aabbDcX & \quad \text{rule: } Db \rightarrow bD \\
\rightarrow aabbcDX & \quad \text{rule: } Dc \rightarrow cD
\end{align*}
\]
Example Derivation for aabbcc

\[ S \rightarrow AX \quad \text{rule: } S \rightarrow AX \]
\[ \rightarrow aAbcX \quad \text{rule: } A \rightarrow aAbc \]
\[ \rightarrow aaBbcbcX \quad \text{rule: } A \rightarrow aBbc \]
\[ \rightarrow aabBbcbcX \quad \text{rule: } Bb \rightarrow bB \]
\[ \rightarrow aabBbcbcX \quad \text{rule: } Bc \rightarrow D \]
\[ \rightarrow aabDbcX \quad \text{rule: } Db \rightarrow bD \]
\[ \rightarrow aabbbDcX \quad \text{rule: } Dc \rightarrow cD \]
\[ \rightarrow aabbcDX \quad \text{rule: } Dx \rightarrow Exc \]
\[ \rightarrow aabbcEXc \]

Eventually ... \rightarrow aabbcc

Note the c spit out on right end!
We could have done this derivation of aabbcc with JFLAP.

Now let’s see how JFLAP visualizes this derivation with a “parse tree”
Input: abbcc
String accepted! 51 nodes generated.

LHS | RHS
---|---
S   | AX
A   | aAbc
A   | aBbc
Bb  | bB
Bc  | D
Dc  | cD
Db  | bD
DX  | EXc
BX  | λ
cE  | Ec
bE  | Eb
aE  | aB
Input: aabbc
String accepted! 51 nodes generated.

<table>
<thead>
<tr>
<th>LHS</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>AX</td>
</tr>
<tr>
<td>A</td>
<td>aAbc</td>
</tr>
<tr>
<td>A</td>
<td>aBBc</td>
</tr>
<tr>
<td>Bb</td>
<td>bB</td>
</tr>
<tr>
<td>Bc</td>
<td>D</td>
</tr>
<tr>
<td>Dc</td>
<td>cD</td>
</tr>
<tr>
<td>Db</td>
<td>bD</td>
</tr>
<tr>
<td>DX</td>
<td>EExc</td>
</tr>
<tr>
<td>BX</td>
<td>(\lambda)</td>
</tr>
<tr>
<td>cE</td>
<td>Ec</td>
</tr>
<tr>
<td>bE</td>
<td>Eb</td>
</tr>
<tr>
<td>aE</td>
<td>aB</td>
</tr>
</tbody>
</table>

Derived AX from S.
Note all letters there, but wrong order: aabcbca
Absorb the “c”
Spit out the “c” at the right end
Absorb second “c”
Spit the “c” out at right end
Input string accepted! 51 nodes generated.

Derived λ from BX. Derivations complete.
What else can JFLAP do?

• Create other machines
  – Moore and Mealy
  – Pushdown Automaton
  – Turing machine

• Parsing of grammars
  – regular, context-free grammars
  – Unrestricted grammar

• Conversions for proofs
  – NFA to DFA to minimal DFA
  – NFA \leftrightarrow regular expression
  – NFA \leftrightarrow regular grammar
  – CFG \leftrightarrow NPDA
JFLAP - L-Systems

- L-Systems may be used to model biological systems and create fractals.
- Similar to Chomsky grammars, except all variables are replaced in each derivation step, not just one!
- Commonly, strings from successive derivations are interpreted as strings of render commands and are displayed graphically.
**Axiom:** $R \sim \## B$

<table>
<thead>
<tr>
<th>LHS</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B$</td>
<td>$[ \sim ## TL - B ++ B ]$</td>
</tr>
<tr>
<td>$L$</td>
<td>$[ \text{angle}=15 { -g ++ g % -- g } ]$</td>
</tr>
<tr>
<td>$R$</td>
<td>$! @@ R$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T g$</td>
</tr>
</tbody>
</table>

**Name** | **Parameter**
---|---
angle | 15
color | brown
polygonColor | forestGreen

$L$-System = $(A, \Sigma, R)$
Expansion contains 264 Symbols!
**Axiom:** $R \sim \# \# B$

<table>
<thead>
<tr>
<th>LHS</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B$</td>
<td>$[ \sim # # T \ - \ B \ + \ + \ B ]$</td>
</tr>
<tr>
<td>$L$</td>
<td>$[ \ \text{angle}=15 \ { \ - \ g \ + \ + \ g \ % \ - \ - \ g } ]$</td>
</tr>
<tr>
<td>$R$</td>
<td>$! \ @@ \ R$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T \ g$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>15</td>
</tr>
<tr>
<td>color</td>
<td>brown</td>
</tr>
<tr>
<td>polygonColor</td>
<td>forestGreen</td>
</tr>
</tbody>
</table>

$L:\text{-System} = (A, \iota, R)$

Table Text Size
Expansion contains 532 Symbols!
L-Systems

The same stochastic L-system, rendered 3 different times all at the 9th derivation.
Using JFLAP during Lecture

- Use JFLAP to build examples of automata or grammars
- Use JFLAP to demo proofs
- Load a JFLAP example and students work in pairs to determine what it does, or fix it if it is not correct.
JFLAP’s use Outside of Class

• Homework problems
  – Turn in JFLAP files
  – OR turn in on paper, check answers in JFLAP
• Recreate examples from class
• Work additional problems
  – Receive immediate feedback
Two-year JFLAP Study 2005-2007

Fourteen Faculty Adopter Participants

- small, large
- public, private
- includes minority institutions

- Duke
- UNC-Chapel Hill
- Emory
- Winston-Salem State University
- United States Naval Academy
- Rensselaer Polytechnic Institute
- UC Davis
- Virginia State University
- Norfolk State University
- University of Houston
- Fayetteville State University
- University of Richmond
- San Jose State University
- Rochester Institute of Technology
Conclusions From Study

• Results of Study showed
  – All the faculty used JFLAP in their courses, mostly for homework, some in lecture
  – Students had a high opinion of JFLAP
  – Majority of students felt access to JFLAP
    • Made learning course concepts easier
    • Made them feel more engaged
    • Made the course more enjoyable
  – Over half the students used JFLAP to study for exams
  – Over half the student thought time and effort using JFLAP helped them get a better grade.
JFLAP is free

www.jflap.org

JFLAP tutorial
Outline

• CS Concepts Come Alive with Software
  – Automata Theory with JFLAP
  – Learning Programming with Alice
  – Algorithm Animation

• Challenges in Designing Educational Software

• CS Concepts Come Alive in other ways
  – Manipulatives
  – Group Activities
  – Edible CS
Alice Programming Language

• Create interactive stories or games
• Learn programming in an easy way, drag-and-drop your code
• Problem solving with visual feedback
  • Objects are visual!
• Alice is free: www.alice.org
• Developed by Randy Pausch
Adventures in Alice Programming
www.cs.duke.edu/csed/alice/aliceInSchools

• 2-week Teacher workshops
  • Over 200 teachers, middle school, high school, some elementary
  • First week Teach Alice, Practice
  • Second week - Develop Lesson Plans
  • All disciplines: math, science, history, language arts, foreign language, art, music, business
• Summers 2008-2015, funding for lodging

• Main Sites:
  • Duke University, Durham, NC
  • Charleston/Columbia, SC
  • San Jose, CA (started 2014)
Curriculum Materials

www.cs.duke.edu/csed/alice/aliceInSchools

• Over 90 tutorials available for free
• Beginner, advanced, challenges, projects
• Paper handouts and video
• Over 200 Teacher lesson plans
  – Organized by discipline and grade level
Computer Science Concepts come alive with Alice - Examples

• Objects are visible
• Variables
• inheritance
• Lists
• Array

• Created Challenges – complete the program
Example: Objects are visible

Getting Started Tutorial teaches

• Placing objects
• Moving objects
• Setting up Camera tripods and moving between views
• Using built in methods and writing your own
  • Dragon flapWings
• Gluing objects together
• Adding sound, 2D pictures to enhance world
Getting Started Tutorial – 3 part
Variables – Timer and Score

To win this game, you must steer the boat through each ring and beat the clock. You receive one point for each ring, and there are 10 rings, so if your score is less than 10 at the end, you lose!
Variables – Scores/Timers
Game: Eragon

4 tasks to win the game
Example - Inheritance

- Start with a chicken object
- Rename it to TalentedChicken
  - Change its color
  - Resize it larger
  - Add new methods (jump, fly, scurry)
  - Add events for this chicken
- Save this new class TalentedChicken that inherits from the Chicken class
Example - List

The Alice Team Summer 2008
Example – Arrays
Shuffle, then Selection Sort

Sort by height
Harry Potter Challenge

• Mix of programming and math challenges

**Hailey Programmer and the Goblet of Java**

You will receive a password at the end of each level that will be used to unlock the next level. WRITE THESE DOWN!
If this is your first time playing, select Charms.
Harry Potter – Math/computing
Level 1 Charms - before
Harry Potter – Math/Computing
Level 1 Charms - after
Outline

• CS Concepts Come Alive with Software
  • Automata Theory with JFLAP
  • Learning Programming with Alice
  • Algorithm Animation

• Challenges in Designing Educational Software

• CS Concepts Come Alive in other ways
  • Manipulatives
  • Group Activities
  • Edible CS
Algorithm Animation Software/Aps/Videos

- AlgoViz.org – collection of algorithm visualizations
- Samba, Jsamba - Stasko (Georgia Tech)
- AnimalScript – Roessling (Darmstadt Univ of Tech, SIGCSE 2001)
- TRAKLA2 – Software Visualization Group – TKK Finland
- Lots of animations and systems on the web!
- Lots of videos of algorithm animations on the web!
Use of Algorithm Animation in CS 1/2

• Instructor
  – Make/Use animations for lecture
  – Stop/Pause – ask what will happen next
  – must be interactive

• Student
  – Create animations
  – Replay animations from lecture with same or new inputs
Lots of other software/programs for algorithm animation

• **Red Black Tree – animation on web page**

http://aleph0.clarku.edu/~achou/cs102/examples/bst_animation/RedBlackTree-Example.html

Student must have graduated. Link no longer works!
Another red-black tree animation

1. Search (top-down) and insert the new item \( u \) as in Binary Search Tree.

2. Return (bottom-up) and

2.1 If \( u \) is root, make it black and the algorithm ends or

2.2 If its parent \( t \) is black, the algorithm ends

2.3 If both \( u \) and its parent \( t \) are red, do one of the following:

2.3.1. [change colors] If \( t \) and its sibling \( v \) are red, change colors: change \( t \) and \( v \) black and their parent \( p \) red. Continue the algorithm in \( p \) if necessary.
Electronic Textbooks (ebooks) engage students

- OpenDSA (Shaffer, Virginia Tech)
  - Algorithm animations built in
- runestoneinteractive.org (Brad Miller,)
  - Several books (Python)
    - Python - try and run code built in
    - Quizzes
- Zyante.com – interactive textbooks
- Track student progress
- Requirements and design strategies for open source interactive computer science eBooks
  - ITiCSE 2013 Working Group (Korhonen, Naps, et al)
Outline

• CS Concepts Come Alive with Software
  • Automata Theory with JFLAP
  • Learning Programming with Alice
  • Algorithm Animation

• Challenges in Designing Educational Software

• CS Concepts Come Alive in other ways
  • Manipulatives
  • Group Activities
  • Edible CS
Make your tool as interactive as possible – but not too tedious!

- User shouldn’t type everything
- Sometimes select
Allow user to proceed on if they got it

- Complete the rest for them
- Complete parts for them
Avoid Too Many Pop up windows

- OLD JFLAP LR PARSE TOOL
Add Pause/Checkpoint questions

• Allow for pause to think about what comes next
• Undo/go back

• Pop up a quiz question to see if the user understands what he/she just did
  – JHAVE tool does this
  – Can integrate into ebooks
What can make the tool more useable?

- Annotations on states
- Multiple run window
  - Develop test data
  - Easier for grading
- General definitions
  - FA – recognize one or more symbols
  - NPDA – pop or push 0 or more symbols
- Batch processing
Naming your software

What is a “good” name for your tool?
JAWAA name is not unique

How popular is JAWAA?
JFLAP name is unique
Outline

• CS Concepts Come Alive with Software
  – Automata Theory with JFLAP
  – Learning Programming with Alice
  – Algorithm Animation

• Challenges in Designing Educational Software

• CS Concepts Come Alive in other ways
  – Manipulatives
  – Group Activities
  – Edible CS
Interaction in Class – Props
Passing “Parameters” in Class

• Pass by reference – throw frisbee

• Pass by value – throw copy of frisbee

• Pass by const reference – throw “protected” frisbee
Interaction in Class – Props
Linked List and Memory Heaps

ITiCSE 98 – Astrachan – “Concrete Teaching: Hooks and Props as Instructional Technology”
Interaction in Class – Props
Memory Heap
Paper Fetch:

If you’ve ever seen one of those wonders of nature movies with lizards that catch flies on their tongues, you’ll pick up this game much more quickly.

1. Lay five or six pieces of paper on the floor, in a circle with a diameter of, say, 6 or 7 feet. (The bigger the circle, the harder the game.) Get a partner with a similar fascination with lizards.
Example: Be a Robot

- 4 People
  - Controller (head)
  - Sensors (eyes)
  - Manipulators (2 hands)
- Blindfolded except eyes
- Controller knows what to build
- Limited communication

SIGCSE 96, Rodger, Walker
Cards

• Card Class – shuffling, dealing hands
• Poker hands – Full house, Flush, etc.
Notable Women in Computing Cards
bit.ly/NotableW

• Based on Wikipedia project – wrote guide on how to write a Wikipedia page on a Notable women in Computing
• Picked 54 Women - deck of cards
• Page on using cards to teach CS
• Can buy a deck
  – Have decks with me for $6/deck
  – Early register for SIGCSE 2015
  – Kickstarter now til Nov 8
Outline

• CS Concepts Come Alive with Software
  – Automata Theory with JFLAP
  – Learning Programming with Alice
  – Algorithm Animation

• Challenges in Designing Educational Software

• CS Concepts Come Alive in other ways
  – Manipulatives
  – Group Activities
  – Edible CS
Engaging students in a group activities/large course

- Problem Solving in groups
  - Clickers, Google forms – compare results
  - Flip Classroom, reading quizzes (turn off at start of class)

- Acting out stories, games
  - *Everything I needed to know about teaching...* - Pollard, Duvall (SIGCSE 2007)

- Acting out algorithms with the whole class
  - Make a binary tree with the whole class
  - Calculate the height of the tree
  - *Making Lemonade ... large lecture classes* – Wolfman (SIGCSE 2002)

- Acting out algorithms with a subset of students
  - Sorting algorithms – selection sort, insertionsort, etc
  - CS Unplugged activities
Middle School students sorting themselves with Bubblesort
Outline

• CS Concepts Come Alive with Software
  • Automata Theory with JFLAP
  • Learning Programming with Alice
  • Algorithm Animation

• Challenges in Designing Educational Software

• CS Concepts Come Alive in other ways
  • Manipulatives
  • Group Activities
  • Edible CS
About Me - Hobby – Baking Shape cakes

The Wiggles magazine Issue No. 42
How do you make those cakes?
What happens when your hobby and your career collide?

It is now time for engaging students with edible CS
Automata Theory
Interaction in Class – Props
Edible Turing Machine

• TM for \( f(x) = 2x \) where \( x \) is unary

• TM is not correct, can you fix it? Then eat it!

• States are blueberry muffins
Students building DFA with cookies and icing
CS 2 – Data Structures
Red-Black Tree (cookies)
CS 1
Animated Bouncing Smileys
CS 1
Sorting
Cookies
Discrete math
A graph and its Dual Graph
Discrete Math
The Pancake Problem
Alice Programming Language
More Alice Programming Language
Conclusions

• We have shown several ways to engage students with CS concepts and make those concepts come alive

• Questions?