Verification Games: Making software verification fun

Michael D. Ernst

University of Washington

Software engineering has been wildly successful

- Software pervades every aspect of our lives

  Try living a day without it!
- Software provides the value in our gadgets
- Huge economic impact
- Increasingly sophisticated functionality
  - We always want to do more!

Software engineering is challenging

Mathematics:

- Modeling
- Analysis

A non-ideal component :

People!

Both aspects are intellectually deep

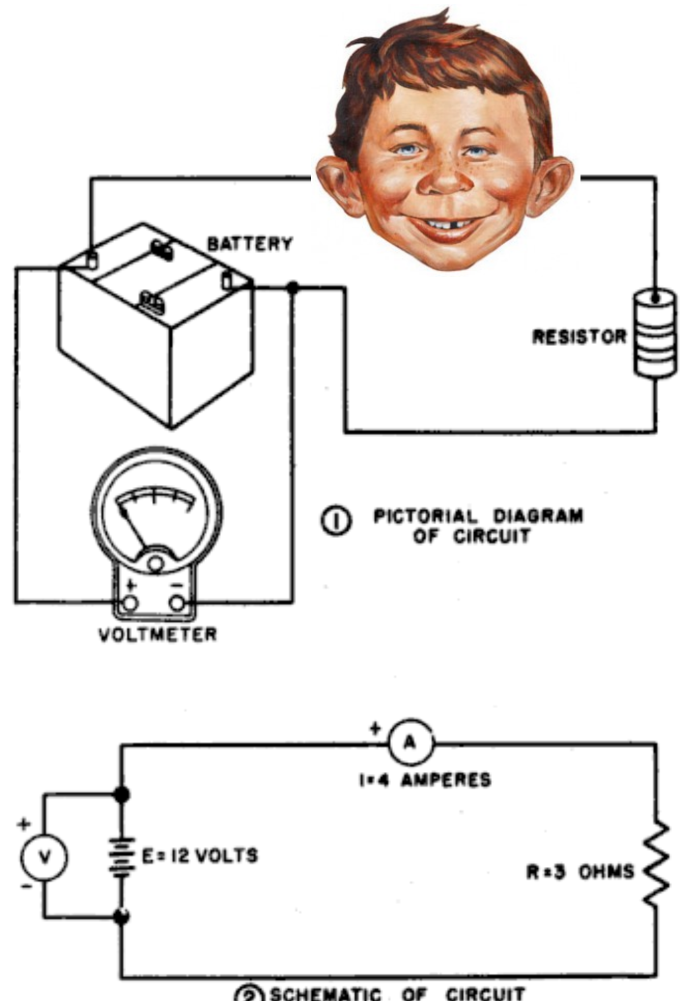


Figure 48. Diagram of a basic circuit.

- New slide??
- SE has been a huge success – maybe the most of any computing field
- (or just say it!)

- SE encompasses technical and psychological/ management [need better word!] boundaries

Hard problems in software engineering

- Choosing modularity and abstractions
- Task breakdown
- Dividing tasks between people and tools
- Transparent vs. powerful tools
- Optimizing intractable problems
- Cooperation, competition, and specialization
- The role of training
- Information overload
- Making SE fun

Angry Birds

Software verification

```
[nmote@monarch level]$ antf check-nullness
Searching for build.xml ...
Buildfile: /homes/gws/nmote/demo/java/Translation/build.xml

clean:
   [delete] Deleting directory /homes/gws/nmote/demo/java/Translation/bin

check-nullness:
    [mkdir] Created dir: /homes/gws/nmote/demo/java/Translation/bin
[jsr308.javac] Compiling 14 source files to /homes/gws/nmote/demo/java/Translation/bin
[jsr308.javac] javac 1.7.0-jsr308-1.1.4
```

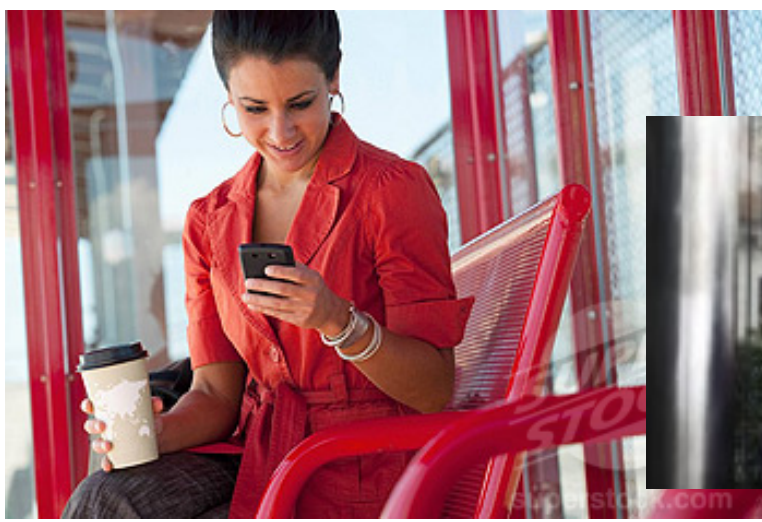
Which is more fun?

- Play games
- Prove your program correct

Crowd-sourced software verification

Goal: Verify software while you wait for the bus

- Make software verification **easy** and **fun**
- Make the game **accessible** to everyone
- Harness the power of the **crowd**



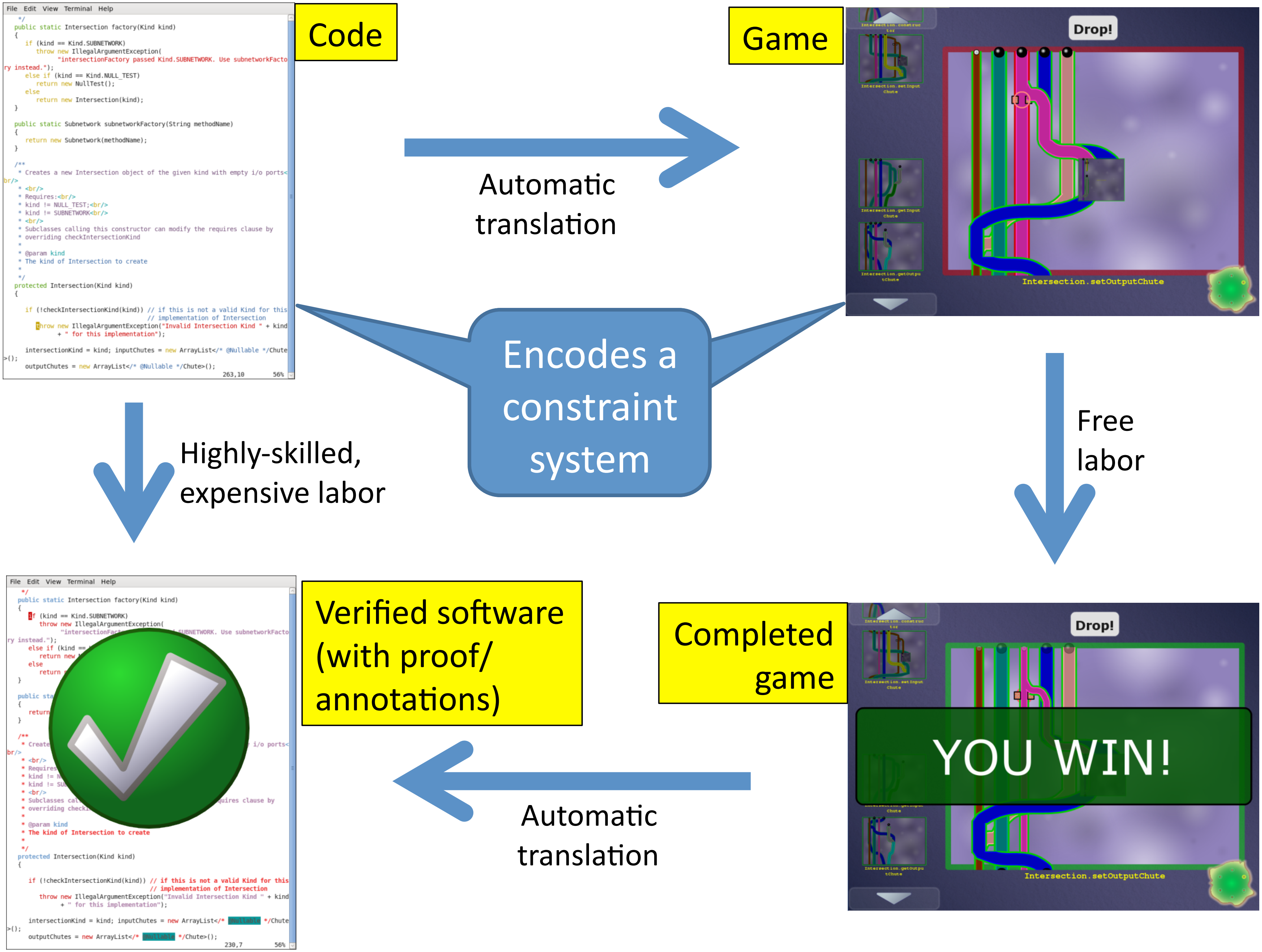

Programming is like a game

Fun puzzles that compel me to solve them:

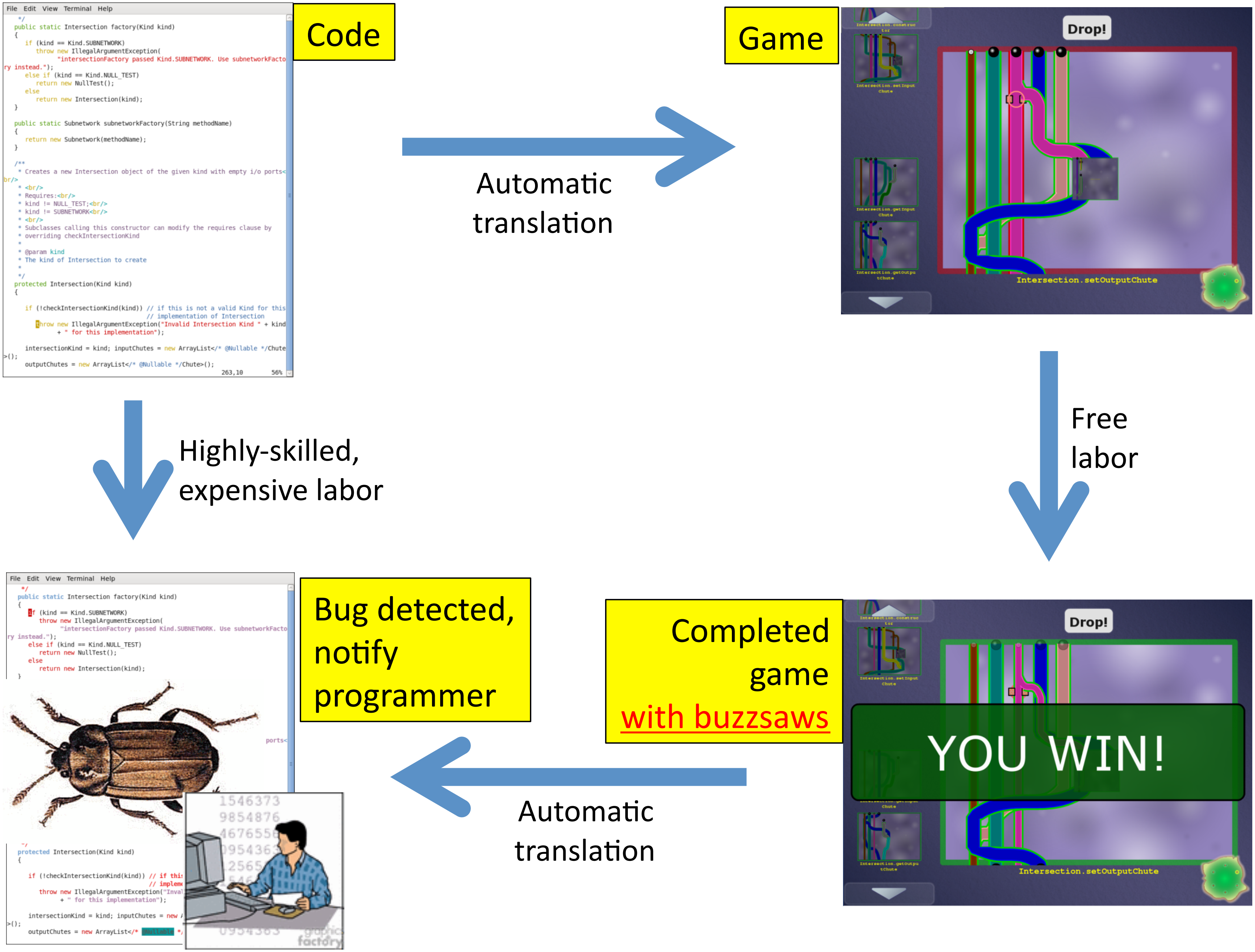
- minimize a test case
- fix a bug
- create a feature
- refactor

When is it not fun?

What is usability in software engineering?

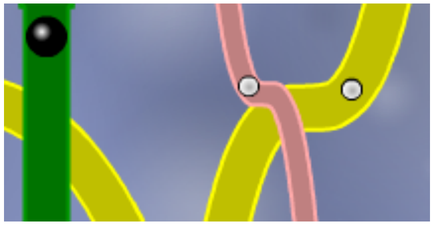

Code
Game
Automatic translation
Encodes a constraint system
Highly-skilled, expensive labor
Free labor
Verified software (with proof/ annotations)
Completed game
YOU WIN!
Drop!
Automatic translation


SAVE
BACK
SUBMIT
World 1 Map
SCORE
2727
DROP!
Connection-vulture--GET
Connection-vulture--SET
Connection-in--GET
Connection--init--Ljava-net-Socket-Ljava-lang-ThreadGroup-ILVulture---V
Connection-run---V
Connection
Connection-vulture--GET
Connection-vulture--SET
Connection-client--GET


Code
Automatic translation
Game
Drop!
Intersection.setOutputChute
Free labor
Highly-skilled, expensive labor
Completed game with buzzsaws
YOU WIN!
Automatic translation
Bug detected, notify programmer

Example: encryption

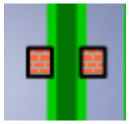
Goal: no cleartext is sent over the network

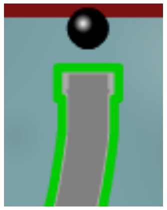


Pipe ↔ a variable

Pipe width ↔ narrow: encrypted, wide: cleartext

Ball ↔ a value





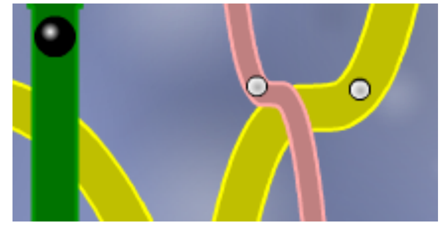
Ball size ↔ small: encrypted, large: cleartext

Pinch point ↔ network communication

Unmodifiable pipe/ball ↔ cleartext from user

Example: null pointer errors

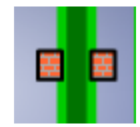
Goal: no dereference of `null`

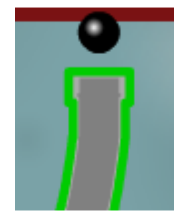


Pipe ↔ a variable

Pipe width ↔ narrow: non-null, wide: maybe null

Ball ↔ a value





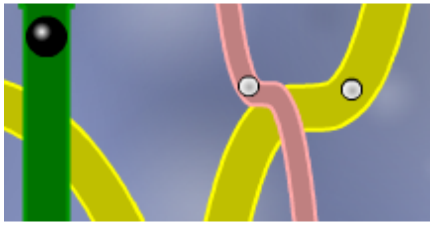
Ball size ↔ small: non-null, large: null

Pinch point ↔ dereference

Unmodifiable pipe/ball ↔ literal **`null`**

Program ↔ game correspondence

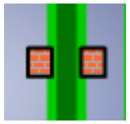
Intuition: dataflow

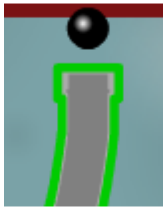


Pipe ↔ a variable

Pipe width ↔ a property of the variable (type)

Ball ↔ a value





Ball size ↔ a property of the value

Pinch point ↔ requirement

Unmodifiable pipe/ball ↔ requirement

Type flow vs. dataflow

- Multiple flows per variable
  - A variable's type may have multiple qualifiers

  @Immutable Map<@English String, @NonNegative Integer>
- Some variables are not represented at all
  - primitives (int, …) when analyzing null pointer errors
- No loops
  - If program is verifiable, solvable in polynomial time
  - Human leverage: high-level pattern matching, placement of buzzsaws/casts

More accurate intuition: type constraints

- Solving a game = type inference
- Computers do a poor job

Other examples

- SQL injection
- unintended side effects
- format string and regexp validation
- incorrect equality checks
- race conditions and deadlocks
- units of measurement
- aliasing
- 27 of the CWE/SANS Top 41 Most Dangerous Software Errors
- …

Type systems for verification

- Modular; local reasoning & understanding
- Equally powerful as any other verification technology (theorem proving, model checking, …)

- Less effective for correctness of numerical computations
- Not good for full functional correctness
- Not good for temporal properties (focus on data)

How do these properties help/hinder the game?

3-way collaboration: machines, players, verification experts

1. Machines: Inference and optimizations
   - Brute force is not feasible for large programs
   - Error messages from type inference systems are poor
2. Players do work that automated tools cannot
   - Use intuition & pattern-matching to place cheats
3. Verification experts do work that players cannot
   - Classify un-verifiable code as safe or insecure

Machine optimization

- Simplify the challenge to its essence
  - Related to the program/problem duality
- Optimization techniques:
  - Abstract interpretation
  - Type inference & constraint propagation
  - Heuristic solving
- In Pipe Jam:
  - Remove multiple pinch points in a row
  - Remove pipes that suffer no conflicts
  - Set pipes to known values, forbid changes to them

Elide irrelevant information

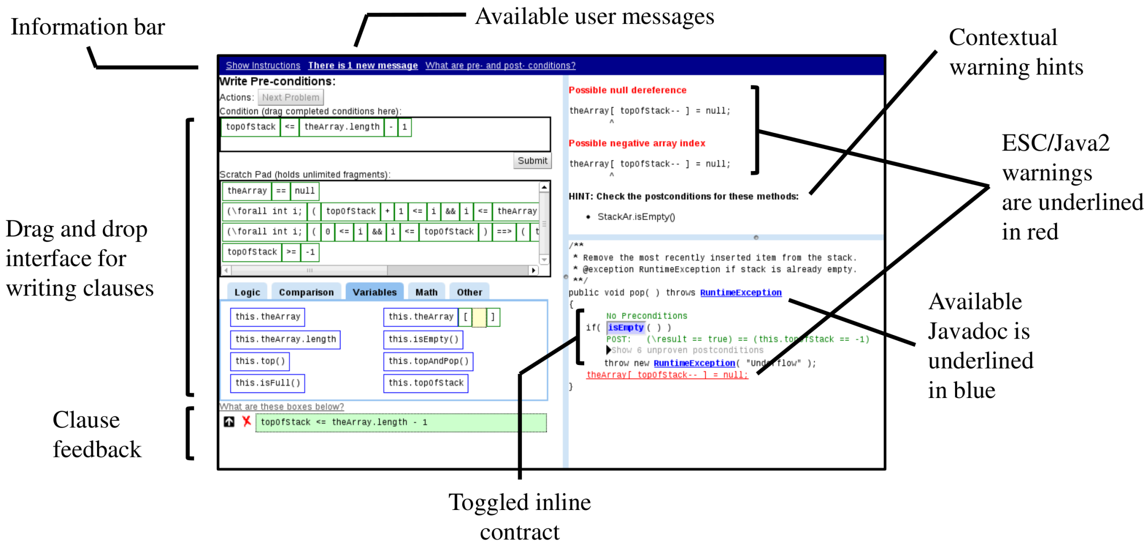
- Example: primitives (`int`), when proving lack of null pointer errors
- Also loses documentation, program context!
- Leave in some easy challenges so players feel good about progress

Information overload & relevance

- Too much detail: player/user gets distracted
- Too little detail: unable to produce useful result

- Example: optimization
- Example: hiding details

Avoiding the big picture



Novice users accomplished more when given less information but given guidance
[“Reducing the barriers to writing verified specifications”, Schiller & Ernst, OOPSLA 2012]

The gaming community

A potentially rich resource

- Angry Birds: 5 million hours of play time *per day*
- 200,000 cumulative years spent (as of 2011)

How do gaming and developer communities differ?

Collaboration and competition

Collaboraton:

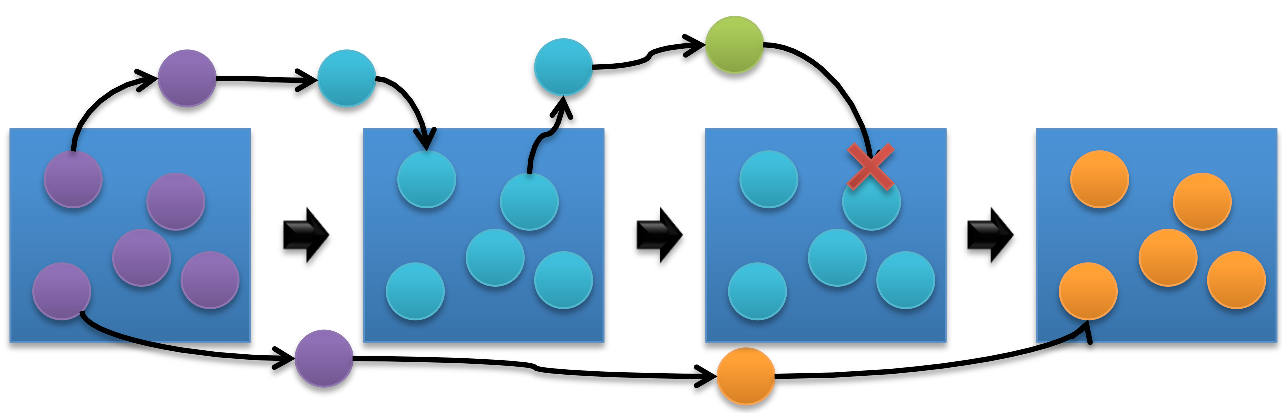
- Teams solve challenges
  - Team scoring
- Share solved levels, scripts
- Interaction: chats, forums, …

Competition:

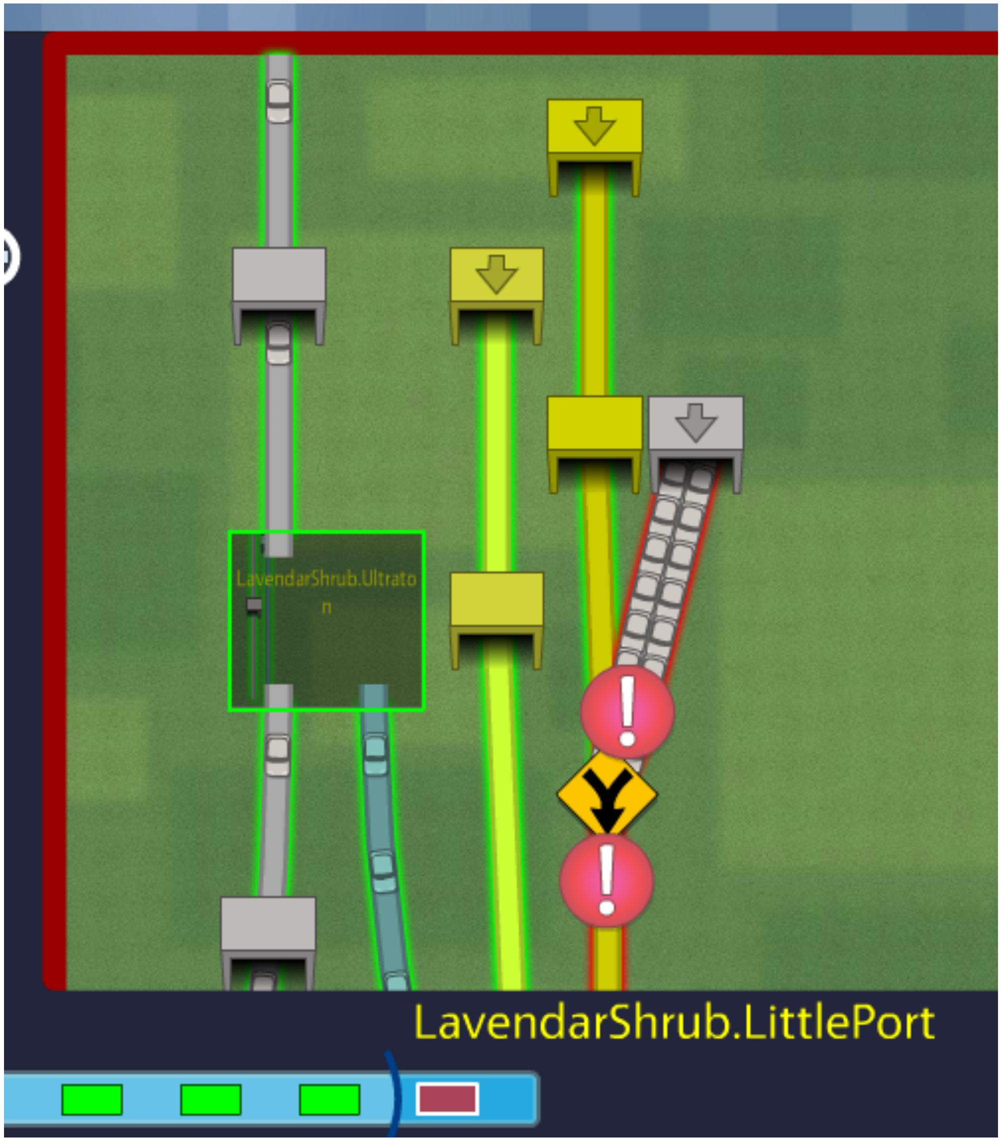
- Leaderboards, badges, challenges

Managing multiple solutions

- A player works on one level at a time
  - Score reflects effect on entire game world
  - Player can indicate need for changes on a different level
  - Player may accept a reduced score – avoid local maxima
- A player/team works in its own universe
  - Can save, restore, merge
  - Best solutions made available to other players



Demo: Traffic Jam



Problem decomposition

Program design methodologies:

- Procedural
- Object-oriented
- Functional
- Logic programming
- Design patterns

Lesson from software engineering:

- No one organization is best for all tasks
- Tradeoffs among competing desiderata

GridWorld

- Problem with Classic: action at a distance
  - Colored pipes are linked & have the same width
  - Represent different uses of the same variable
  - Game abstractions are same as the program's
  - Goal: bring information together

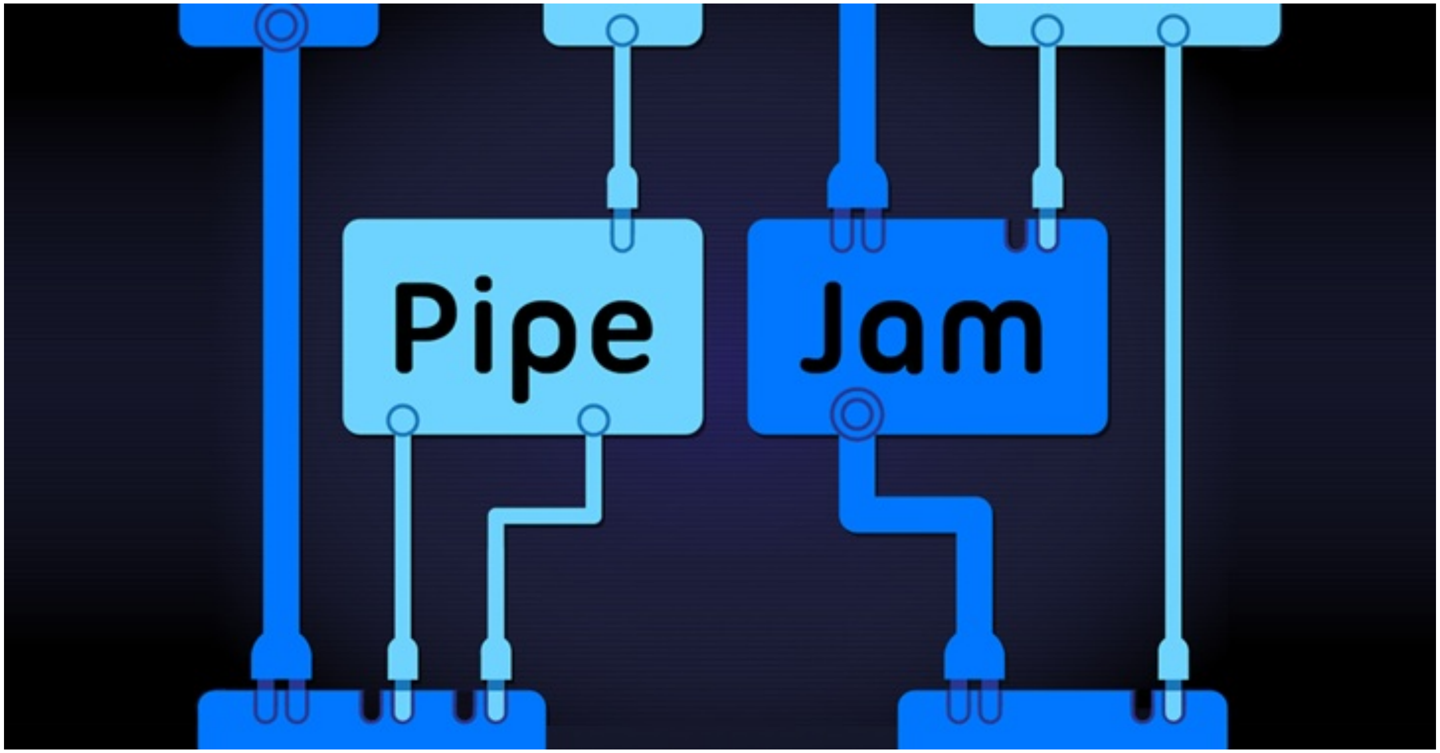
Problem: action at a distance

Pipe colors indicate non-local dependences: uses of the same variable must be consistent

Organizing a program's constraints

- Programmer-supplied decomposition
  - Classes, methods
  - Programmer probably had a good reason
  - But: variables & calls cross-cut these structures
- Alternate decomposition:
  - Bring together variables, split apart method bodies

Demo: Flow Jam



Just a new skin for the same game

- Pipes $\Rightarrow$ boxes
  - One box for arbitrarily many pipes of the same color
- Pipe connections $\Rightarrow$ lines
  - Didn't eliminate action at a distance, but made it explicit

Identical constraints and XML input file

Player is solving the same problem,
but it feels like a different game

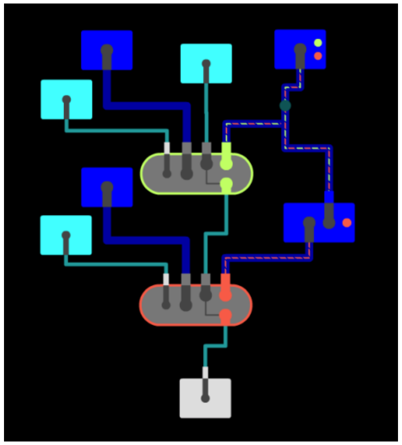
We plan A/B testing

Implications of Grid World's variable-oriented layout

- Fewer boards, but bigger ones
  - Lots of explicit links
  - Layout and navigation are more challenging
- More compact representation
  - No traveling balls/cars, no sub-boards
  - See more on the screen
- Two game-playing modalities: conflicts, layout

Type inference is challenging

- Example: prove that **`myMap.get(someKey)`** returns a non-null value
  (recall: **`Map.get`** returns null if the key isn't in the map)
  - myMap is declared as Map<KeyType, @NonNull valueType>
  - someKey is a key in myMap



- Example: polymorphism (Java generics)

Design goals for a (software engineering) game

- Address a hard problem
- Connect players to the real work they are doing
- Scale to (and be useful for) real problems

- Fun
- Use human skills
- Minimal distractions from underlying problem

- Allow and encourage collaboration

Designing a program analysis

(Examples: model checking, abstract interpretation)

Key problem: the abstraction

- Capture the essence of the problem
- Too much detail: infeasible analysis
- Too little detail: does not prove desired properties

Designer insight and iteration are crucial

Each new successful abstraction is a breakthrough

Designing a machine learner

Key issues:

- Learning algorithm (SVM, decision tree, neural network, genetic algorithm, …)
- Feature space (problem representation): the information fed to the algorithm

State of the art:

- Try lots of algorithms
- Try lots of feature spaces
- When one works, publish it

Designing a (software engineering) game

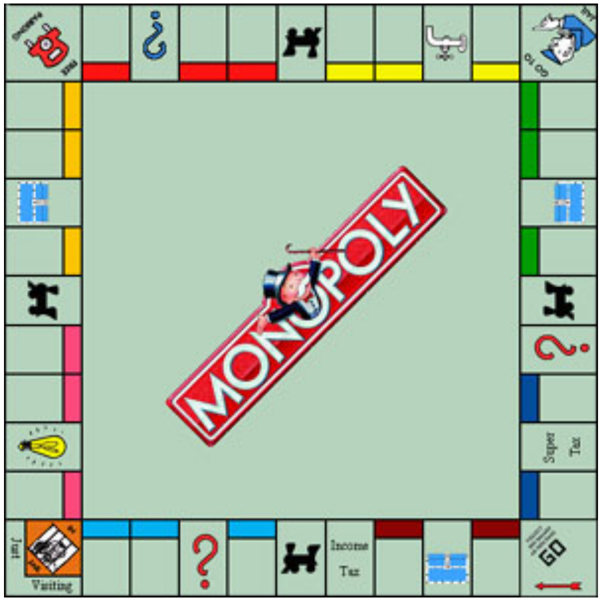
Goals:

- Address a hard problem
- Use human skills
- Fun

A challenging task with no simple rules

Modularity and abstraction make it even harder

Some successful games



ESP game (image labeling)

Duolingo (translation)

ReCAPTCHA (optical character recognition)

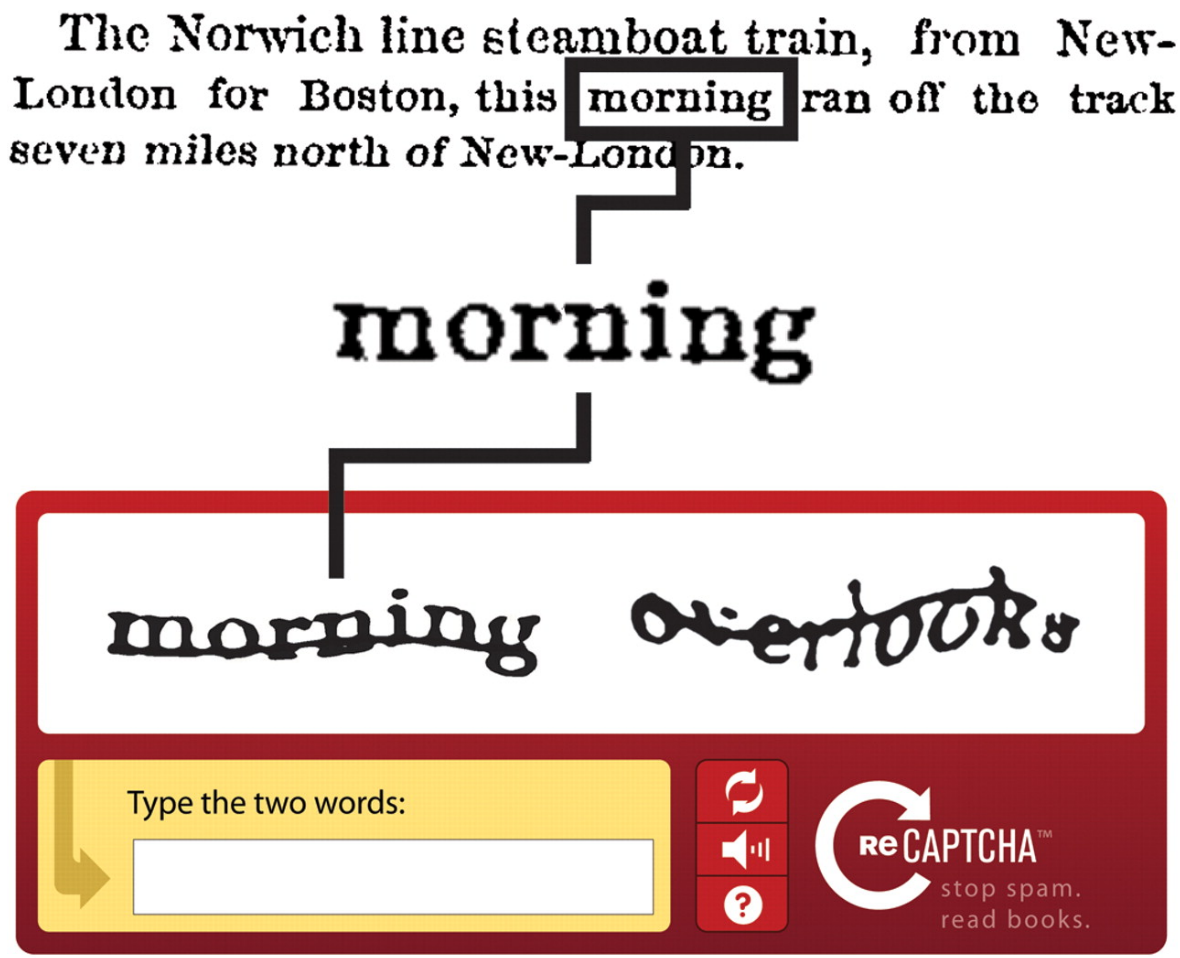
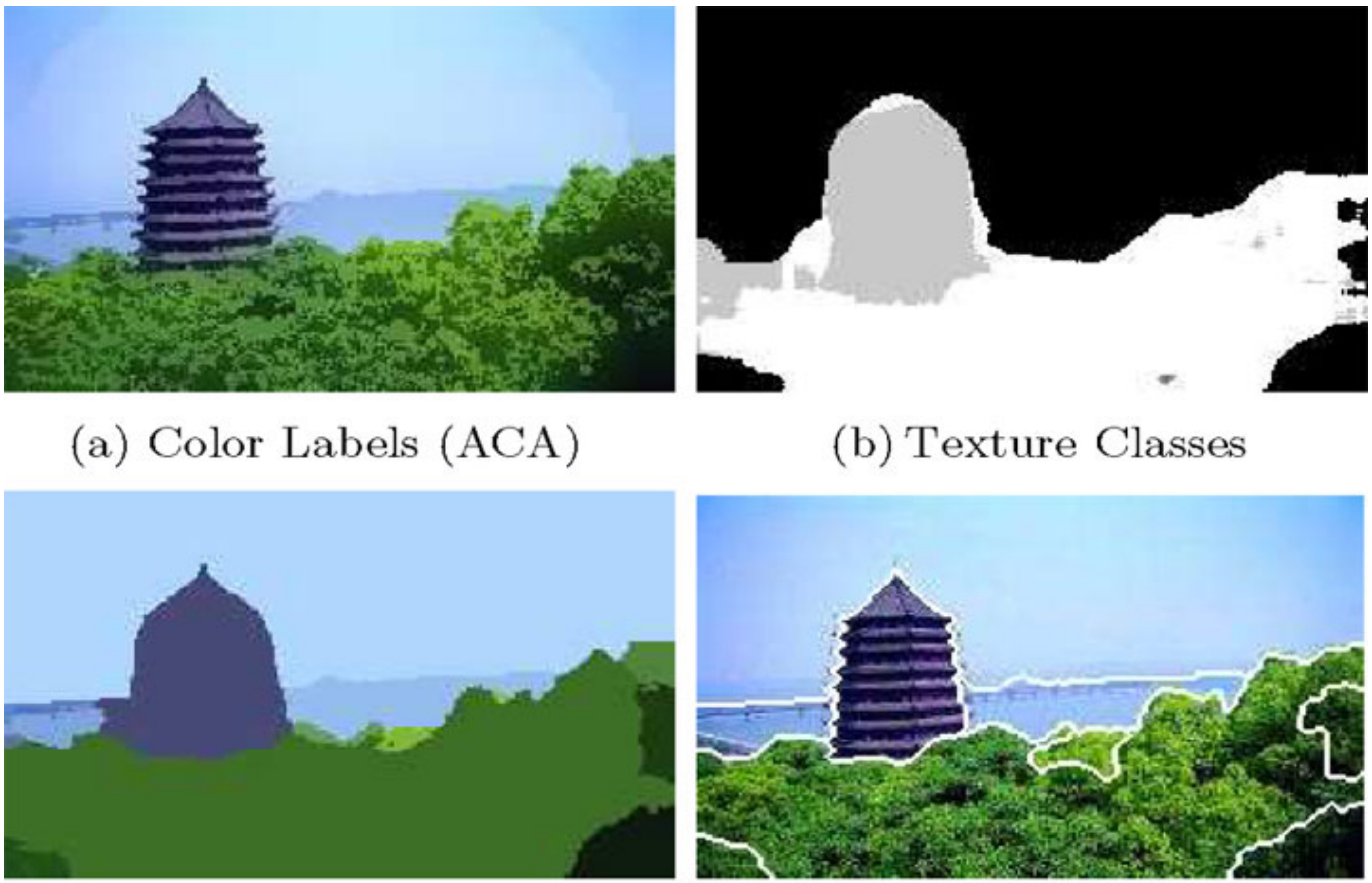


Image segmentation



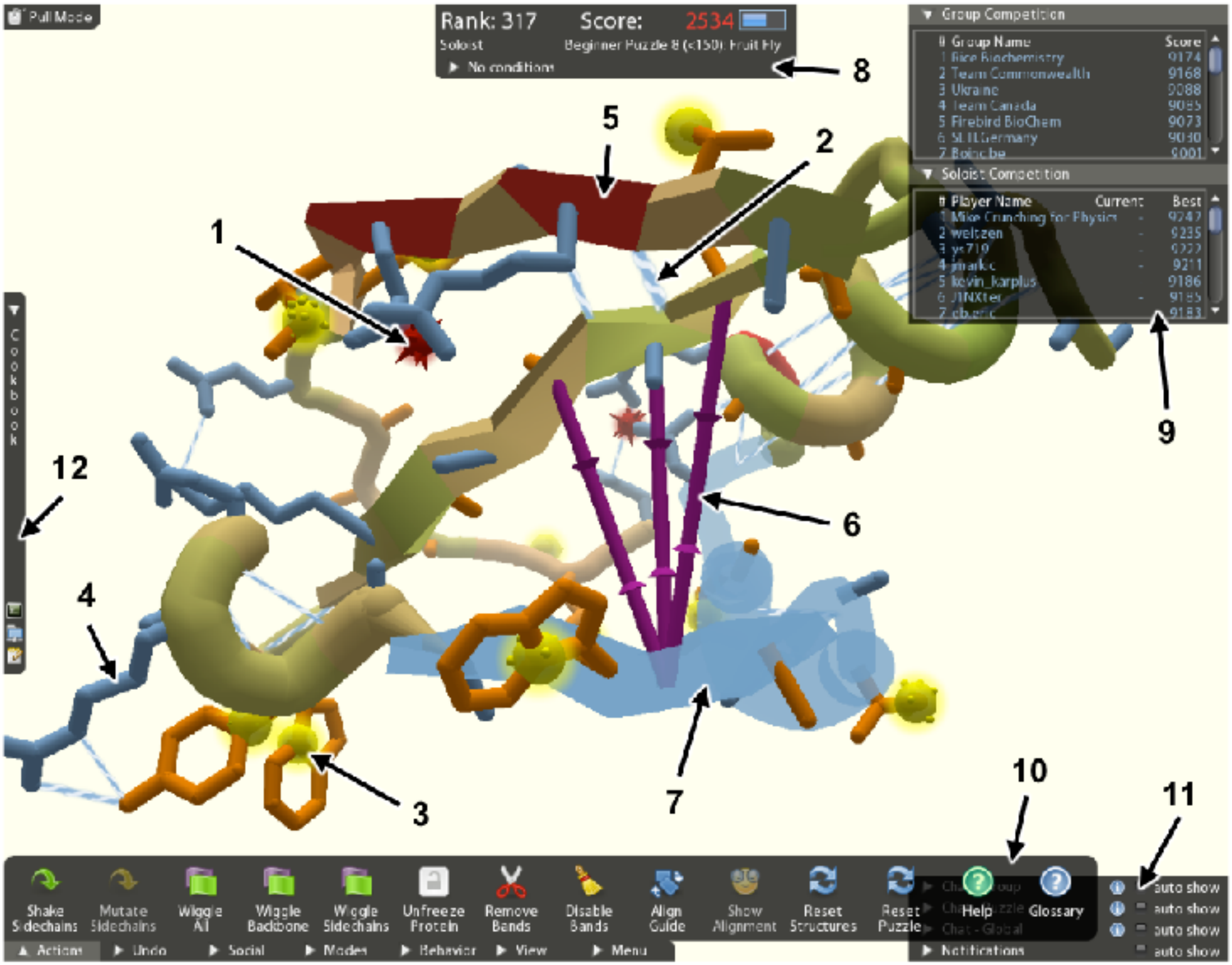
(a) Color Labels (ACA)

(b) Texture Classes

(c) Crude Segmentation

(d) Final Segmentation

FoldIt (proteomics)



FoldIt

- Proteomics game at UW
- Effectively created the genre of games that solve hard problems
- Three Nature papers in under 2 years
- Over 240,000 players, 200+ new per day

Comparison of games

Game	Abstraction?	Modularity?
Image labeling	✗	✓
Translation	✗	✓
OCR	✗	✓
Image segmentation	✗	✓
Protein folding	✗	✗
Type inference	✓	✓

Challenge: create more games that are abstract and modular

Abstraction and modularity in game design

Abstraction

- Is the player doing the goal task directly?
- In an abstracted game:
  - No need for expertise in the problem domain
  - No obvious connection to the real-world value

Modularity

- Each player solves part of the overall problem
- The system combines the contributions of different players
- Programs have natural modularity, created by the programmer

  In Pipe Jam:
  - World = program
  - Level = class
  - Board = method

Pipe Jam solves multiple problems

Anything that can be expressed as a type system:

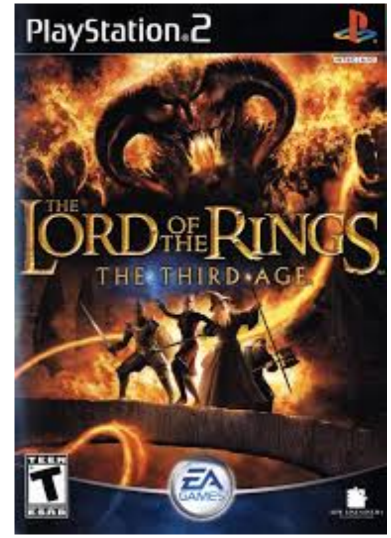
- null pointer errors
- encryption
- SQL injection
- unintended side effects
- format string and regexp validation
- incorrect equality checks
- race conditions and deadlocks
- units of measurement

For a new type system:

- Map type system into the Pipe Jam schema
- Convert a program to a game instance

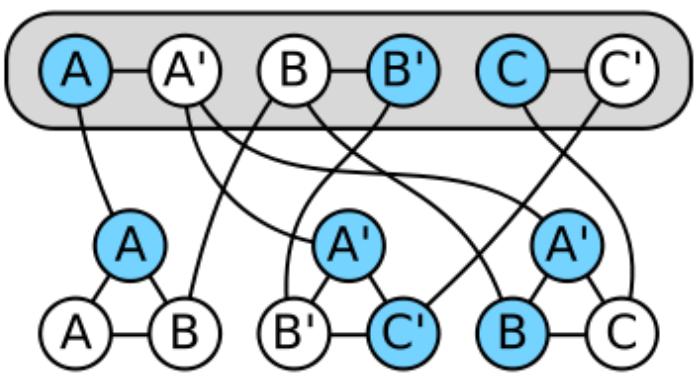
Pipe Jam also contains a layout game (different skill set)

One game to rule them all?



Problem reduction:
Many problems can be converted to SAT

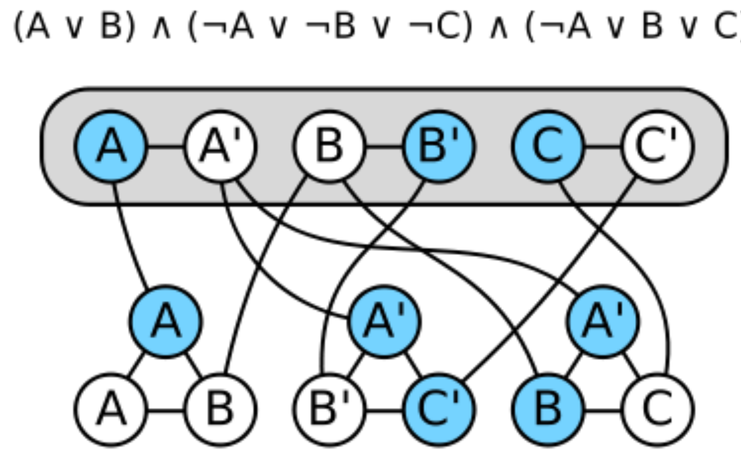
$(A \lor B) \land (\neg A \lor \neg B \lor \neg C) \land (\neg A \lor B \lor C)$



Can we gamify all those problems simultaneously?

Don’t think about SAT

… when you play the game



… when you design the game

Translation to SAT:

- Explodes problem size
- Destroys problem structure, no human intuition

Casual gamers vs. trained experts

With time, players develop unique skills

- A plumber might be a protein-folding savant

Our focus is not on casual gamers

Useful work comes from trained expert players

Human advantage

Do people outperform verification algorithms?

Inference is undecidable (human experts $\gg$ algorithms)

Hypothesis: no for correct, verifiable programs,

yes for incorrect or unverifiable programs

Location of buzzsaws is key to the whole approach

Game players only have to reduce overall verification cost, not fully verify the program

Player performs optimization

- Type error (Jam): -75 (or -1000)
- Wide inputs or narrow outputs: +25 (or +10)
- Constant offset to make positive

$$\max_A \alpha \sum_{c \in C} satisfied(c, A) + \sum_{a \in A} \beta_a desired(a) + \gamma$$

Scoring

Score is influenced by:

- Collisions (verifiability)
- Use of buzzsaws (trusted assumptions)
- Pipe widths, distinguishing input and output pipes (re-usability of modules)

Multiple solutions may be possible

Score is a proxy for quality of verification result

- Have we just rephrased the hardest question?
- Heuristics & search strategies for an optimization problem
- Discover algorithms that may outperform players

Other games being built

- Invariant detection (a la Daikon)
- Model checking
- Model merging
- Register allocation
- … your ideas here!

* These are not my ideas; many come from DARPA's Crowd-Sourced Formal Verification Program (Dr. Drew Dean, PM)

Play now at Verigames.com

Creating test cases

- Given a solved game,
  seek input balls that cause a conflict
- This can be converted to a test case

Other games being built:

- Model checking
- Model merging
- Register allocation

Pipe Jam status

- Prototype exists
  - Tested on modest programs (~100,000 lines)
  - Players say it is “kind of fun”
- Many challenges remain
  - Create tests (example failures, or counterexamples)
  - Scale to multiple players (parallelism, social aspects)
  - Make the game more fun

Gamification of SE (program verification)

Goal: cheaper verification ⇒ more verification

Pipe Jam and Flow Jam games…

- encodes correctness condition
- utilizes physical intuition & human insight
- is playable by anyone

Play at http://verigames.com

Credits

Collaborators on Verification Games project:

Jonathan Barone, François Boucher-Genesse, Brian Britigan, Dan Brown, Jonathan Burke, Matthew Burns, Craig Conner, Seth Cooper, Werner Dietl, Stephanie Dietzel, Kate Fisher, Barb Krug, Marianne Lee, Bryan Lu, David McArthur, Nathaniel Mote, Zoran Popović, Tim Pavlik, Tyler Rigsby, Eric Reed, Eric Spishak, Brian Walker, Konstantin Weitz

Funding: DARPA

Hard problems in software engineering

- Choosing modularity and abstractions
- Task breakdown
- Dividing tasks between people and tools
- Transparent vs. powerful tools
- Optimizing intractable problems
- Cooperation, competition, and specialization
- The role of training
- Information overload
- Making software enigneering fun

Games↔SE: A useful, if imperfect, analogy
Science benefits from:

- A games perspective on SE
- A SE perspective on games

May apply elsewhere

Disclaimer

This material is based on research sponsored by Defense Advanced Research Project Agency (DARPA) under agreement number FA8750-11-2-0221. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Defense Advanced Research Project Agency (DARPA) or the U.S. Government.