Lessons Learned in Game Development for Crowdsourced Software Formal Verification

Presented by Drew Dean

Contact Author: Sean Guarino

http://www.verigames.com

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions, and/or findings contained in this article are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Authors

Dr. Drew Dean, Dr. John Murray
SRI International

Mr. Sean Guarino, Mr. Leonard Eusebi
Human Effectiveness, Charles River Analytics Inc.

Mr. Andrew Keplinger
Left Brain Games

Dr. Tim Pavlik
Center for Game Science, University of Washington

Dr. Ronald Watro
Raytheon BBN

Mr. Aaron Cammarata
voidAlpha

Ms. Kelly McLaughlin
XPD Analytics

Dr. John Cheng, Mr. Thomas Maddern
Veracient LLC

Overview

- Problem Overview
- Case Studies
  - CircuitBot/Dynamakr
  - Flow Jam/Paradox
  - Ghost Map/Ghost Map: Hyperspace
  - Stormbound/Monster Proof
  - Xylem/Binary Fission
- Conclusions and Lessons Learned

Problem Overview

- Typical software has 1-5 bugs per thousand lines of code
- Formal verification can reduce this to 0.1 – 0.5 bugs per thousand lines of code
- However, formal verification is too expensive and time consuming
  - Typically 2x – 100x cost increase
  - Optimists will point to the times costs have decreased

Making Formal Verification Affordable

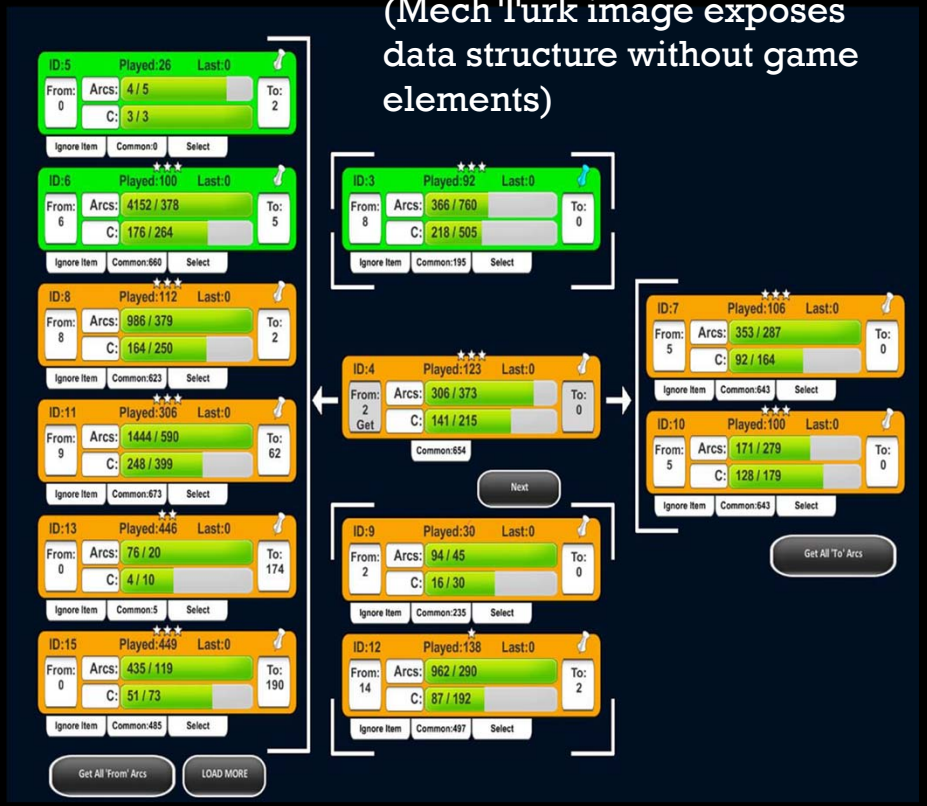
- How do we typically decrease costs?
  - Automation
- Alas, Rice's Theorem limits automation
  - All non-trivial properties of programs are undecidable in general
  - Much empirical support in practice
  - See Kathleen Fisher's talk on Thursday for progress in automation

Making Formal Verification Affordable

- Can we replace the expensive, highly trained computer scientists with the crowd?
  - Only if we make the tools far more approachable
  - Convert logic puzzles into games
  - Market …

Case Study: CircuitBot/Dynamaker Developing Points-to Graph

- Some types of automated verification require a points-to graph (i.e., graph of which pointers may hold which addresses at runtime)
- Determining reasonable approx-imations of the points-to graphs requires a high degree of graph intelligence (human or, perhaps, machine).
- Our auto-solver can run without human intervention. It is a research question whether it will compete with human experts in performance.

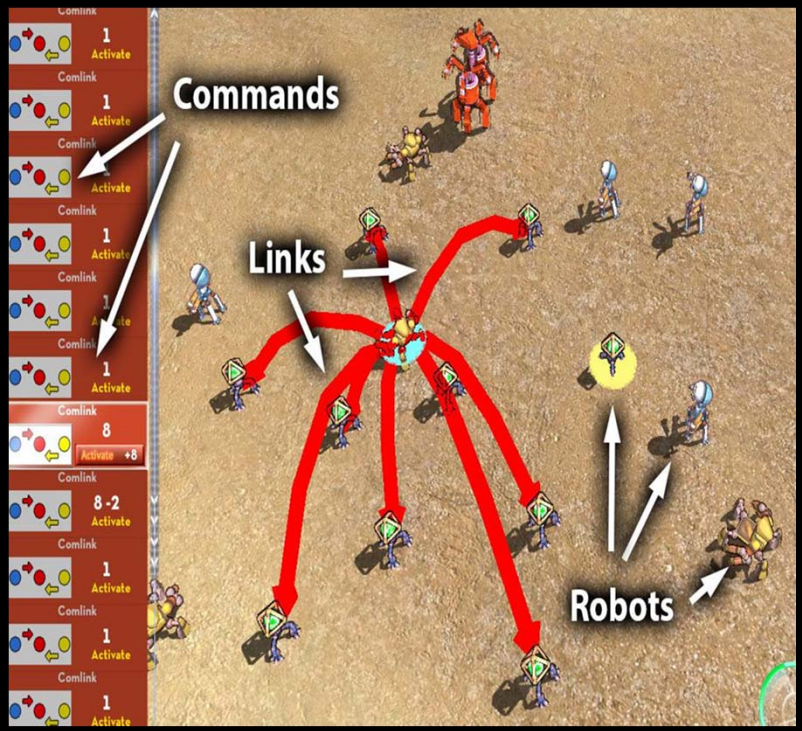


(Mech Turk image exposes data structure without game elements)

Team Authors:
Andrew Keplinger & Greg Izzo, Left Brain Games
Matthew Barry, Kestrel Technology
J. Nelson Rushton, Texas Tech University

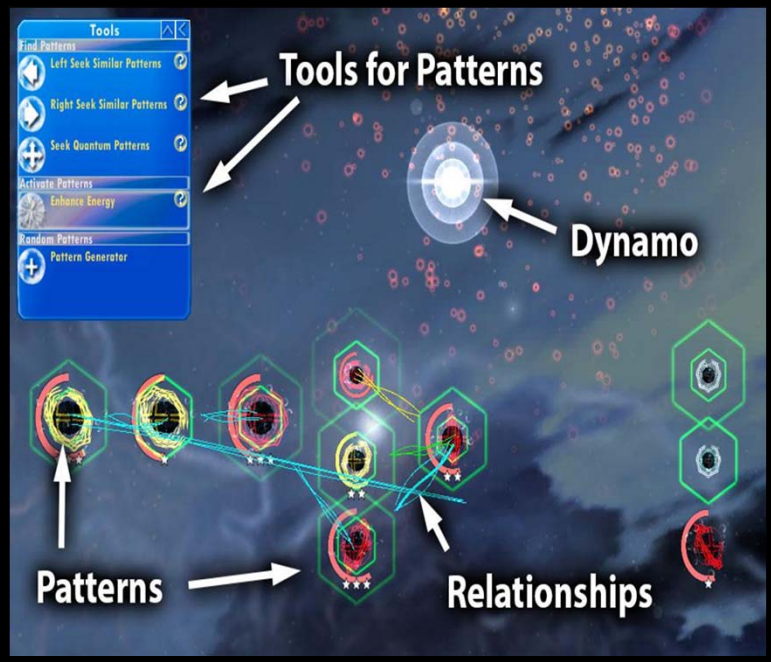
CircuitBot and Lessons Learned

- CircuitBot was developed to provide a framework for building the points-to graph
- Core game is designed to apply rules combined with existing information to produce relevant data
- Overarching exploration and strategy game provides motivation to replay core game
- Combination was effective at motivating certain types of players
- Rule development refinement could produce improved results
- Players could address much larger sets of rules if trivial content was filtered



Dynamakr

- Play multiple game instances simultaneously, illustrating how game results influence each other



- Find related games dynamically and determine play priorities for efficiency
- Auto-solver analyzers are embedded in the game, working alongside the player

Case Study: Flow Jam/Paradox Overview

- Input is a set of constraints over possible annotations
- Game level generated from constraints (links) and possible annotations (widgets) from code
- Widgets and links can be wide or narrow, links are the width of the widget they flow from
- Constraints include:
  - Jams on links
  - Bonuses on widgets
- Player assigns values to annotations to satisfy as many constraints as possible
- Output is desired annotation values

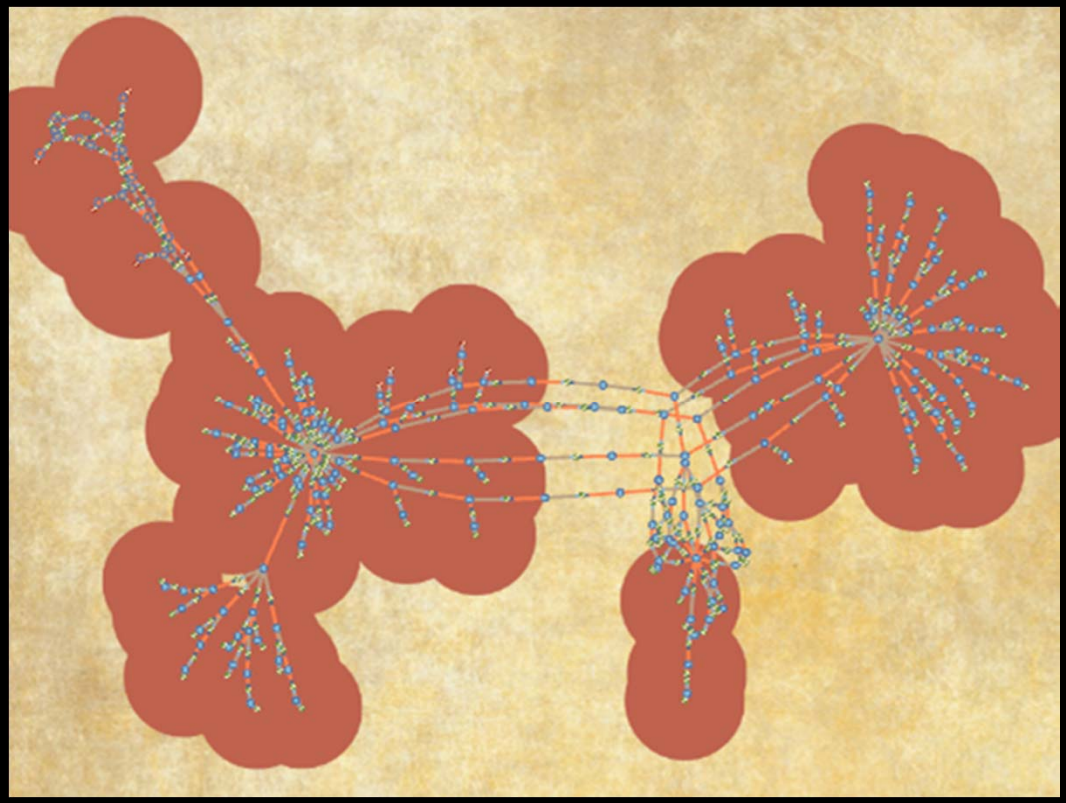
Team Authors:
Tim Pavlik, Craig Conner, Jonathan Burke, Matthew Burns, & Michael Ernst, University of Washington
Werner Dietl, University of Waterloo
Seth Cooper, Northeastern University

Flow Jam: Lessons Learned

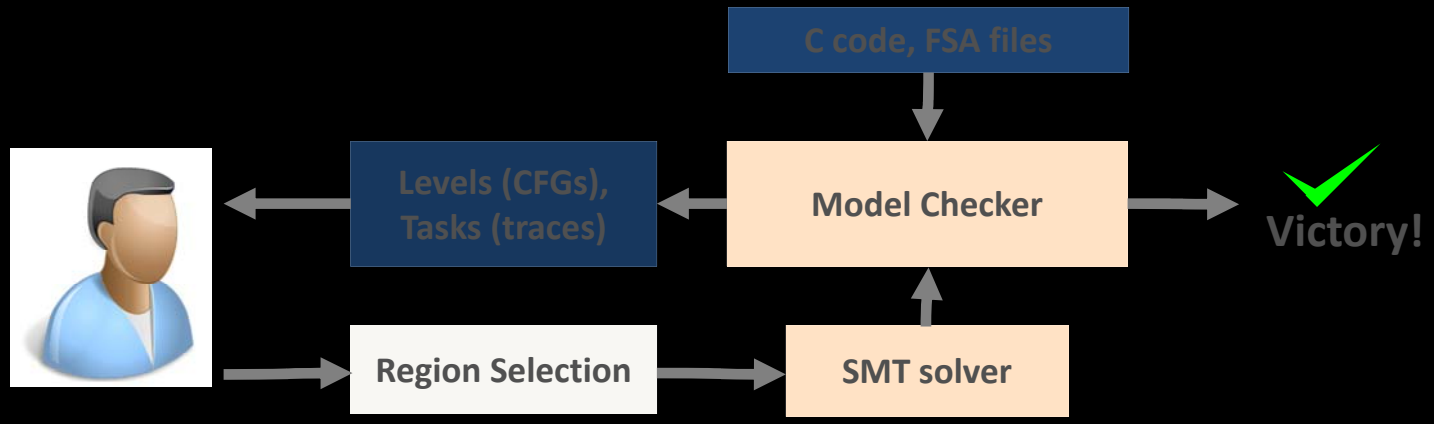
- For larger levels, toggling widgets individually was tedious
- Level layouts were difficult to understand
- Did not leverage human spatial reasoning

Paradox: Refined User Support

- Simplified graphical representation and improved graph layouts
- Allowed players to use auto-solve toolset for more rapid play
- Provided spatial painting mechanics to drive application of auto-solver



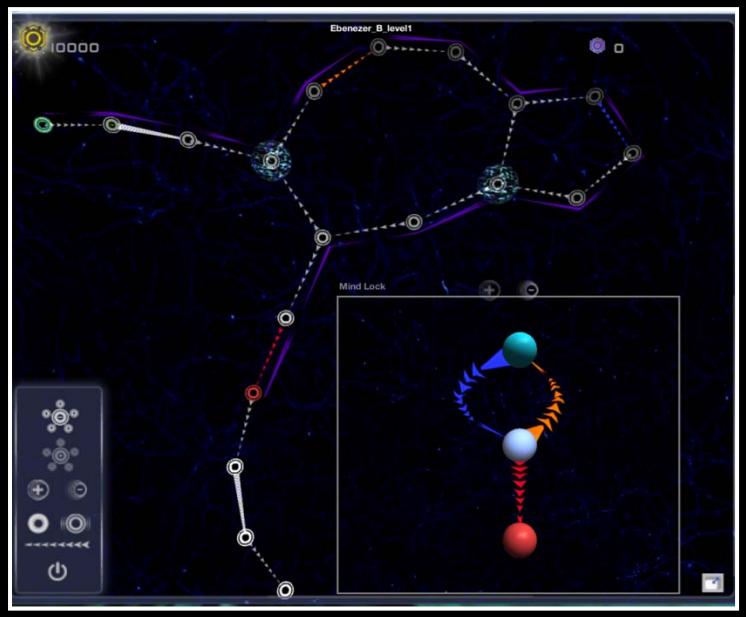
Case Study: Ghost Map



- Model checker finds counterexample traces
  - Violations of security property, encoded as FSA
- User selects region on trace to send to SMT
- Success means region unsatisfiable
  - Trace unrealizable; ergo false positive
- Modified graph sent back to model checker
  - Either done, or new trace

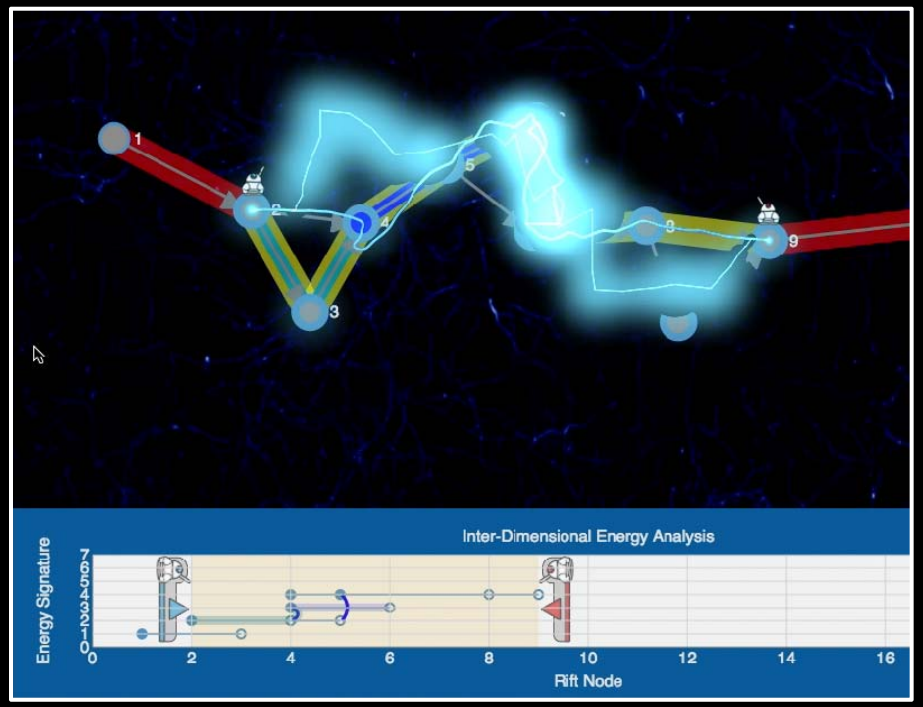
Team Authors:
Ronald Watro, Kerry Moffit, John Ostwald, Dan Wyschogrod, Andrei Lapets & Linsey Kennard, Raytheon BBN
Eric Church, BreakAway Games

Proof by Games – Ghost Map



- Player
  - AI on verge of consciousness
- CFG
  - Map of player's own mind
- FSA
  - "Mind Lock" preventing consciousness
- Player Goal
  - Modify own mind to thwart "Lock"
- Challenges
  - "Mind Locks" difficult for players to understand
  - FSAs do not sufficiently guide player actions

Proof by Games – Ghost Map: Hyperspace



- New Approach
  - Do not show FSA
  - Instead show program variables
  - Add "pure entertainment" element to enhance engagement
- Player
  - Space mercenary
- CFG
  - Map of Hyperspace
- Program Variables
  - "Energy Signatures"
- Variable Dependencies
  - "Energy Chains"
- Player Goal
  - Seal "Hyperspace Rifts" to prevent alien invasion
- Challenges
  - Balance "pure entertainment" action game with the real math game
  - Program enough information about variable usage – but not too much

Case Study: StormBound and Monster Proof Verification Approach

- Verifying memory safety (e.g. no buffer overruns) of C programs
- Through gameplay, players create assertions about code at particular program points
- Game-generated assertions assembled into a proof
- StormBound: Players view multiple snapshots of program variables at single program point and look for common patterns, e.g. "i < sizeof(ary)"
- Monster Proof: Players build proofs directly using well-defined rules, trying to prove a precondition that must be true if a function is memory safe

Team Authors:
Aaron Cammarata, VoidAlpha
Aaron Tomb, Galois Inc.

Comparing StormBound and Monster Proof

StormBound	Monster Proof
	
• Story-driven engagement • Levels solved by *pattern matching* • "Magepunk" universe, blend of brass/steam and glowing magical runes • Goal of "hiding the math": allow players to make assertions without any math or numbers in-game • Targeted a broader, casual audience • Used Unity Webplayer, embedded in a MeteorJS web page	• Resource-gathering and collection • Levels solved by *reasoning* • Cute cartoon monsters, emphasis on tongue-in-cheek humor • Goal of "showing the math": give players tons of context, and focus on efficiency and comprehension • Targets a focused puzzle-game audience • Used Famo.us for HTML/CSS Sprites, and MeteorJS for web page / server

Lessons Learned

- Challenging to give players a sense of progress and completion
  - Solutions to levels are unknown *a priori*
  - StormBound: players supplied (and rewarded for) as many level solutions as they liked without clear completion point
  - Monster Proof: clear when a solution is correct, with optional opportunity to improve on it
- Players need sufficient context to ensure solutions useful for verification
  - StormBound: players could (and mostly did) inadvertently create true but useless assertions that relate variables with no semantic relationship
  - Monster Proof: reduced complexity of levels as much as possible, then gave players all needed context needed to reason about the level
- Exposing the underlying problem helps motivate players ("citizen scientist" message)
  - StormBound: we "hid the math," but players asked to see it
  - Monster Proof: messaging clear about how game play impacts verification

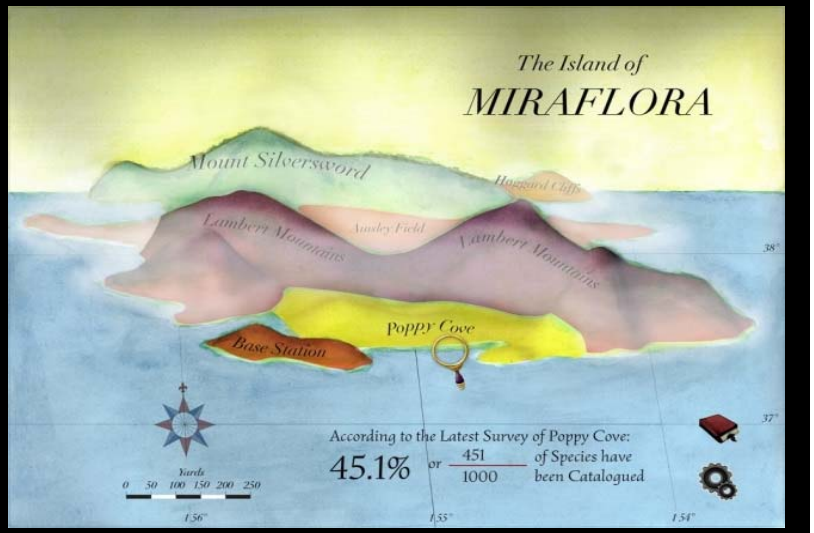
Case Study: Xylem/Binary Fission Overview

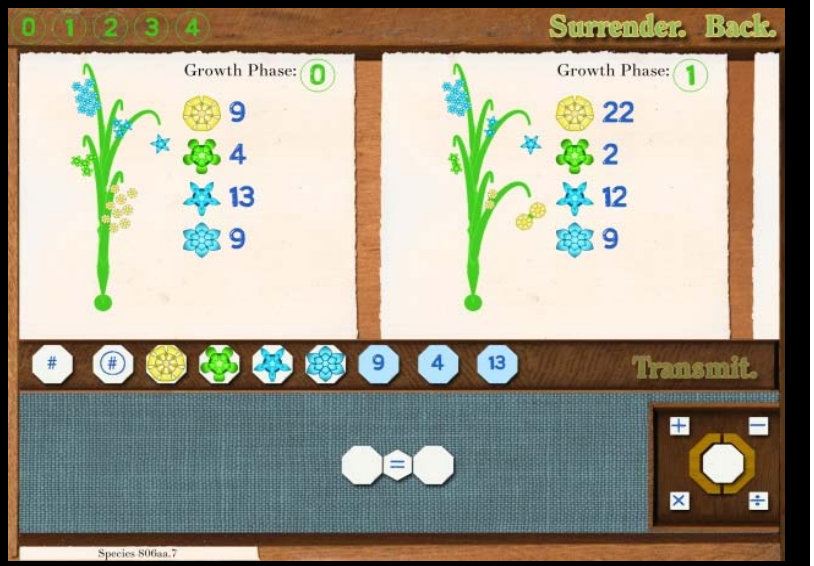
- Xylem presents problem as logical induction puzzle game
  - Players are botanists exploring a strange island
  - They observe and compare growth patterns of plants
  - Provide candidate invariants for the CSFV verification task in the process
- Binary Fission focuses on sorting colored atoms into two groups
  - Select from set of filters to perform sorting
  - Refine and merge invariant searches that have been performed by other games/automated systems

Team Authors:
John Murray, SRI International
Heather Logas & Jim Whitehead, University of California, Santa Cruz

Xylem: Lessons Learned

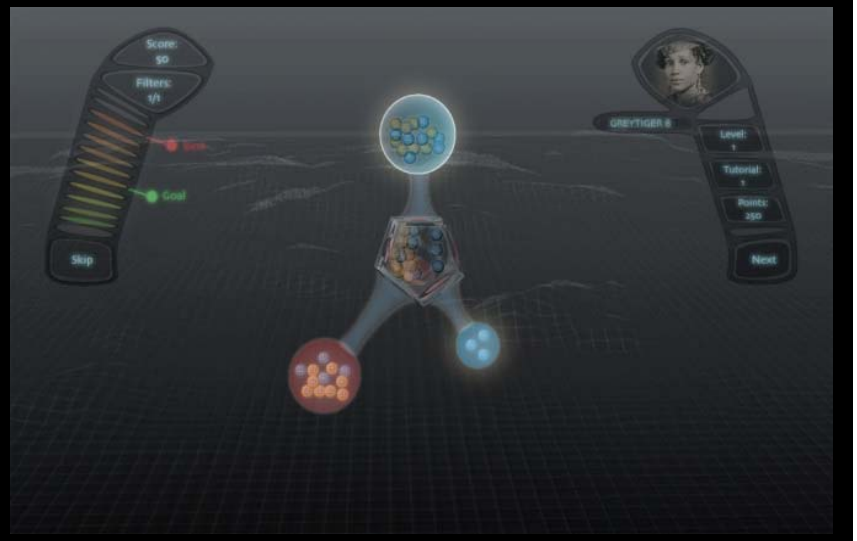
- Original goal focused on appealing to casual game audience by integrating puzzle-solving with evolving game narrative
- Casual gamers were not interested in mathematical gameplay
- Xylem complexity and lack of clarity led to limited players in casual gaming community



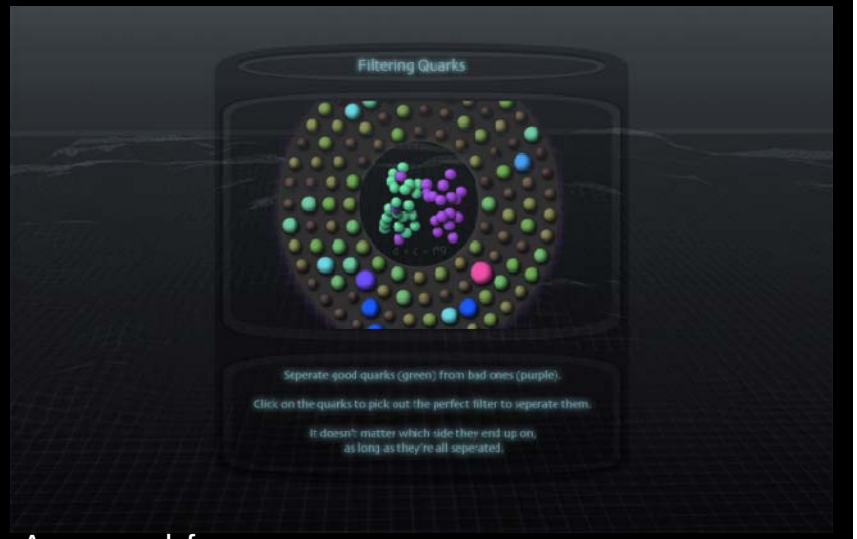


Binary Fission: Adapted to Support Citizen Science

- Binary Fission embraced citizen science approach by designing problems focused on a more mathematically inclined crowd



- Binary Fission also incorporated more cooperative game play, to foster community interaction in support of citizen science objectives



Conclusions/Lessons Learned

- *Know the player population*
  - Initial focus on engaging crowds with limited mathematical background – but these are not the high contributors
  - Problem better served by *citizen scientists* with mathematical expertise – later games were designed to this objective
- *Manage the complexity of the game design*
  - Use a progression of tools that teaches key concepts for contribution
  - Entertain players as new concepts are taught
- *Manage tradeoff between engagement and problem resolution*
  - Consider separating problem-solving process from fun and entertainment – incorporate engaging elements in downtime
  - Maximize the use of human intuition and insight
- *Use automation where possible*
  - Minimize busy work performed by the citizen scientists
  - Manage insertion of automation to ensure game players understand mechanics and impacts

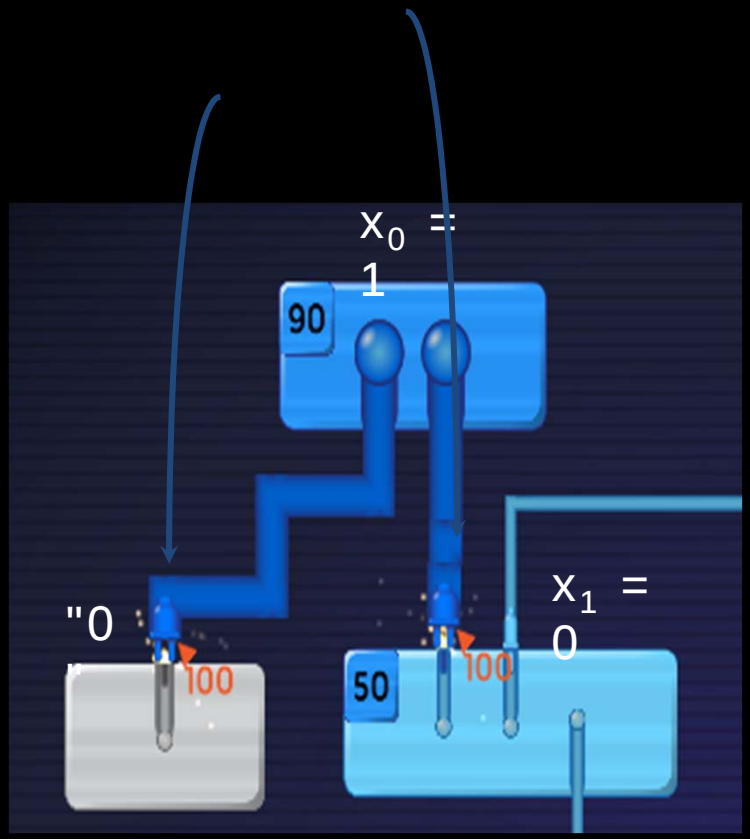
Contact Points and Website

Drew Dean
Program Director, SRI International
650-859-2444
ddean@csl.sri.com

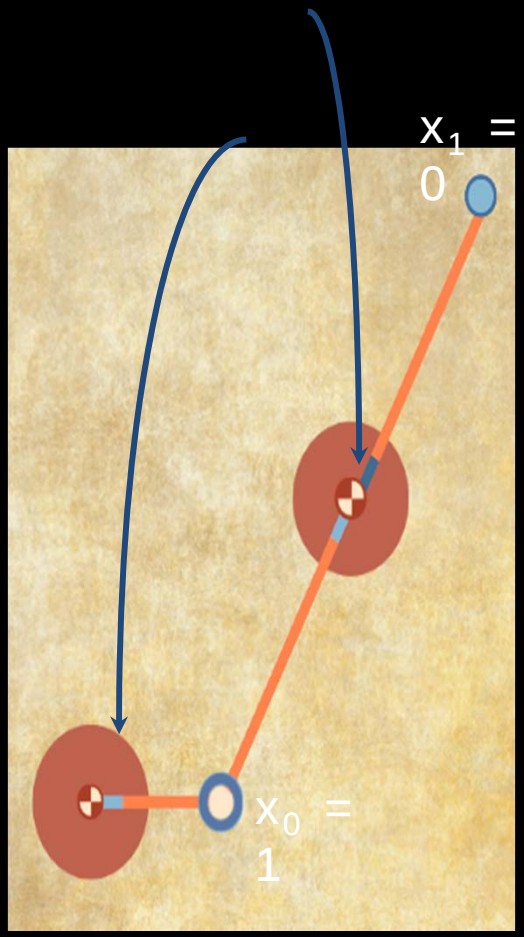
Sean Guarino
Principal Scientist, Charles River Analytics
617.491.3474 Ext. 561
sguarino@cra.com

CSFV games can be found at http://www.verigames.com

Paradox Backup: Constraints

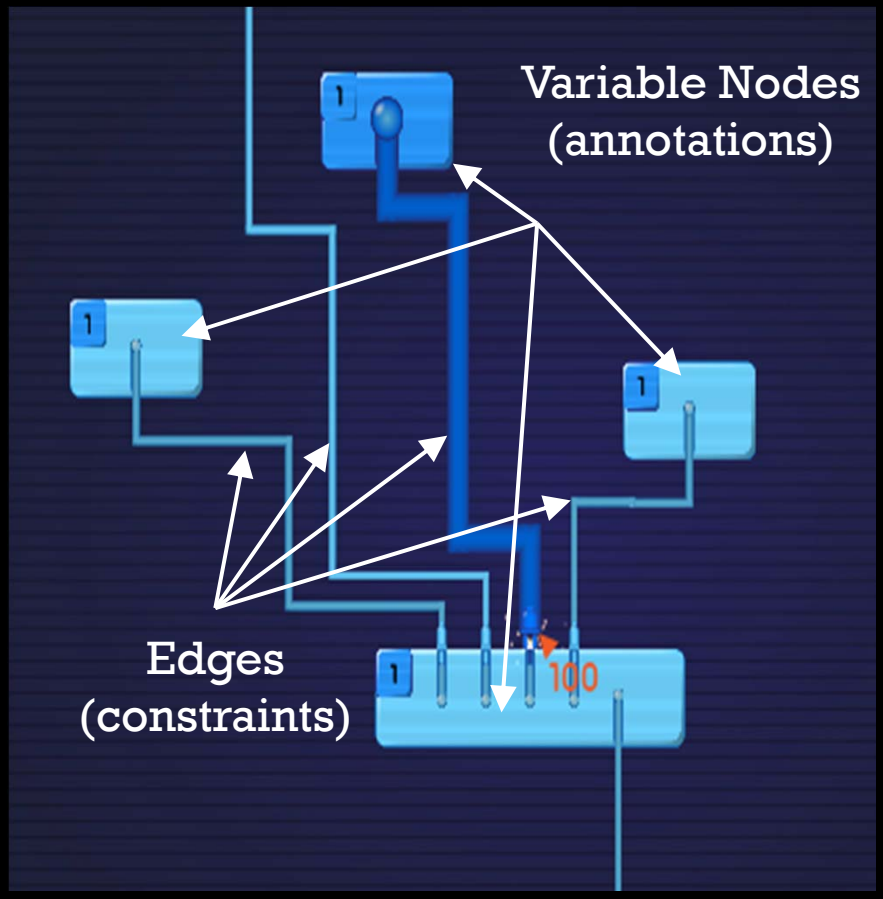


Flow Jam

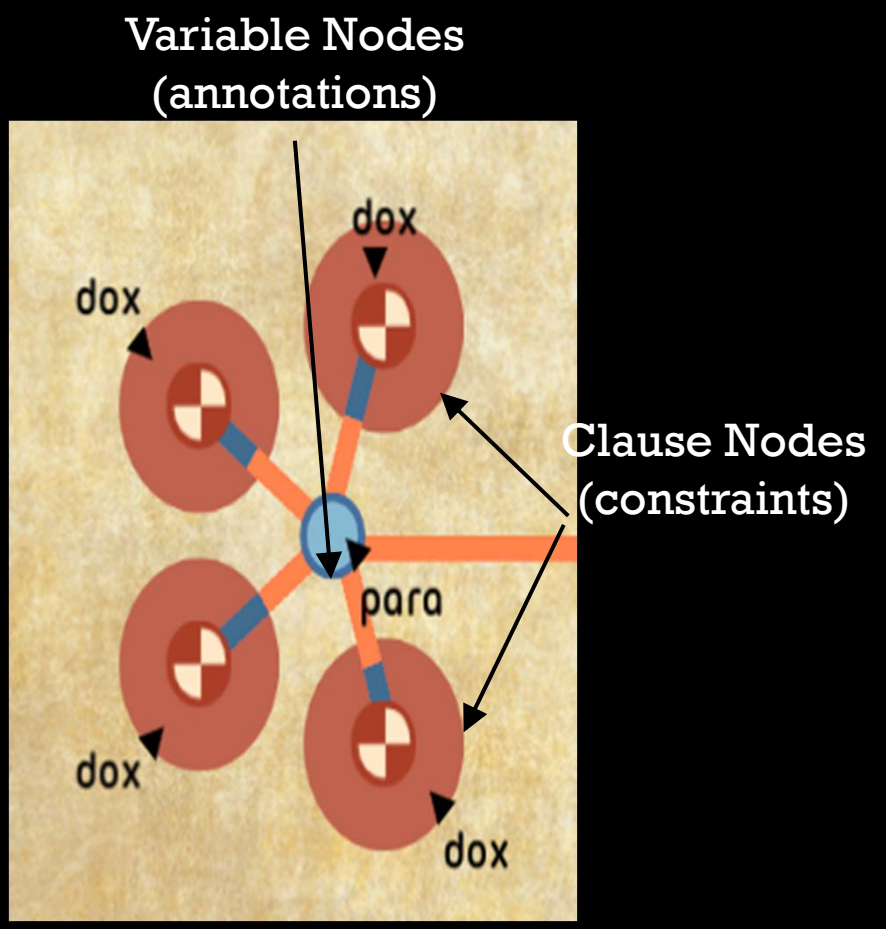


Paradox

Paradox Backup: Game Elements



Flow Jam



Paradox