

Shaping the Programming Experiences of Students:

Reflections on a Course on Game Design for Problem Solving

Steven L. Tanimoto
University of Washington

Paul G. Allen School of Computer Science and Engineering

PX/17.2



Outline

- Introduction
- A Course on Games for Problem Solving
- Problem Formulation
- Wicked Problems
- Towards Standard Formats
- Liveness in Problem Solving Environments
- Design vs Problem Solving
- Other Issues
- Peer Critiques in the Programming Lab
- Concluding Thoughts

Outline

- **Introduction**
- A Course on Games for Problem Solving
- Problem Formulation
- Wicked Problems
- Towards Standard Formats
- Liveness in Problem Solving Environments
- Design vs Problem Solving
- Other Issues
- Peer Critiques in the Programming Lab
- Concluding Thoughts

Instructor's Objectives

- Clarify an interdisciplinary approach to problem solving.
- Build on past work:
 - Collaborative Design (2009, 2010, with T. Robison, B. Johnson, and others)
 - CoSolve (with Sandra Fan et al. 2012).
 - Problem Formulation (with Rob Thompson, unpublished)
- Refine software that codifies a promising approach to formulating and solving problems.
- Engage students in learning important content.

The Course

- First offering: Spring 2017 ("the spring offering")
- Second offering: Summer 2017 ("the EFS offering")
 - The U.W. Early Fall Start program
 - Course flyer
 - Calendar
 - Student population and backgrounds
 - Classical Theory of Problem Solving

Shaping Student Experiences

- Students already have varied backgrounds in computer programming.
- Now we do programming “in context.” (serving a separate end than just fluency: game creation).
- Helps level the playing field, de-emphasizing the primacy of coding fluency.
- Offer new experiences to all...
- Agile, Scrum, Iterative design, User-centered design
- New libraries and coding frameworks.

Outline

- Introduction
- **A Course on Games for Problem Solving**
- Problem Formulation
- Wicked Problems
- Towards Standard Formats
- Liveness in Problem Solving Environments
- Design vs Problem Solving
- Other Issues
- Peer Critiques in the Programming Lab
- Concluding Thoughts

The Early Fall Start Program at U.W.

Get Ahead of the Pack

Are you ready for the University of Washington? Make a smooth transition to the UW by attending Early Fall Start, a special program for new students.



Make a Great Start

With Early Fall Start, you can enroll in a UW course about a month before autumn quarter begins and get comfortable with college life.

EFS Program (cont.)

The Biology of Human Consciousness

CSI: Seattle

Experiencing the Arts

Fetal Origins of Adult Diseases: A Public Health Perspective

Flight & Space Exploration

Food!

Four Great Moments in Classical China: History & Philosophy Through Literature & Film

Game Design for Problem-Solving With Python

Iconic Seattle: A Diverse Home for the Arts

Icons of Place: Exploration Through Poetic Thought

Indistinguishable From Magic: New Technologies, Science Fiction & Us

Kids, Crime & Race: Juvenile Justice in America

Laws of Sex: Exploring Legal Issues About Sex, Gender & Sexuality

Neurobiology of Critical Thinking: What Is Reality?

Numbers & Reasons

Out of This World: Writing About Science Fiction

The Political Economy of Everything

Sagas of the Vikings: Outlaws & Poet-Warriors

Secret Codes & Online Security

Sensory Worlds & Behavior of Animals

The Sound of Seattle

STEM(M) in the Ancient World: Computers, Robots, Neuroscience & Death Rays

Sustainable Energy Solutions for the 21st Century: Science, Technology & Policy

This Is Your Brain on Drugs

Travel Writing for Travelers (Not Tourists)

Unraveling the Dark Universe With the Large Hadron Collider

A World of Stories

Wicked Problems

Fate of the World

```

if m_remaining > 0 and m_remaining < c_remaining:
    n_at_arrival = p[M][1-side]*m
    c_at_arrival = p[C][1-side]*c
    if n_at_arrival > 0 and n_at_arrival < c_at_arrival:
        return True

def move(olds,n,c):
    '''Assuming it's legal to make the
    the new state resulting from m
    m missionaries and c cannibals.
    s = copy_state(olds) # start with
    side = s['boat']
  
```

python

Game Design for Problem Solving with Python

<p>Tuesday, August 22: Course Introduction lecture pdf Getting Started with Python lecture pdf</p> <p>Reading: LP intro. P0 out</p>	<p>Wednesday, August 23: Python's basic data objects; working with strings in Python lecture pdf</p> <p>Reading: LP basic data</p>	<p>Thursday, August 24: Working with numbers in Python; lists in Python lecture pdf</p> <p>Reading: LP more data P0 due at 5:00</p>	<p>Friday, August 25: Python's control structures; defining functions in Python lecture pdf Classical Theory of Problem Solving lecture pdf</p> <p>Reading: GD Game design intro; LP control structures P1 out</p>
<p>Tuesday, August 29: Python class definitions lecture pdf Classic Puzzles lecture pdf</p> <p>Reading: LP Python class definitions</p>	<p>Wednesday, August 30: Formulating Problems lecture pdf A game with graphics and introduction to Tkinter Reading: GD formulation P1 due; P2 out</p>	<p>Thursday, August 31: Wicked problems lecture pdf Running SOLUZION from IDLE; Demo: Formulating Tic-Tac-Toe Reading: GD wicked problems</p>	<p>Friday, September 1: Game structures lecture pdf Project ideas and planning Reading: GD game structures P2 due; P3 out</p>
<p>Tuesday, September 5: [Quiz 1 covering game structures] Introduction to software development with Scrum lecture pdf</p> <p>P3 Milestone A due</p>	<p>Wednesday, September 6: The Prisoner's Dilemma and related obstacles to problem solving lecture pdf P3A presentations of game ideas Reading: GD Prisoners Dilemma</p>	<p>Thursday, September 7: Featured guest: Maggie Ryan (CSE undergraduate program advisor) [Quiz 2 covering the prisoner's dilemma] Lab work, Scrum practice, more on Python functions</p>	<p>Friday, September 8: Learning curves lecture pdf Lab work Reading: GD learning curves P3 Milestone B due</p>
<p>Tuesday, September 12: [Quiz 3 on learning curves, theory of problem solving, and wicked problems] Wicked Problem Case Study: Homelessness lecture pdf</p> <p>P3 Milestone C due</p>	<p>Wednesday, September 13: Lab work, Scrum practice, and presentations of game ideas</p>	<p>Thursday, September 14: Iterative design of games; lab work P3 Milestone D due</p>	<p>Friday, September 15: Final presentations, demos, and game evaluations P3 Milestone E due</p>

last updated: September 7, 2017.

Python

Problem solving theory & formulation

Game design and theory

Software engineering

<p>Tuesday, August 22: Course Introduction lecture pdf Getting Started with Python lecture pdf</p> <p>Reading: LP intro. P0 out</p>	<p>Wednesday, August 23: Python's basic data objects; working with strings in Python lecture pdf</p> <p>Reading: LP basic data</p>	<p>Thursday, August 24: Working with numbers in Python; lists in Python lecture pdf</p> <p>Reading: LP more data P0 due at 5:00</p>	<p>Friday, August 25: Python's control structures; defining functions in Python lecture pdf Classical Theory of Problem Solving lecture pdf</p> <p>Reading: GD Game design intro; LP control structures P1 out</p>
<p>Tuesday, August 29: Python class definitions lecture pdf Classic Puzzles lecture pdf</p> <p>Reading: LP Python class definitions</p>	<p>Wednesday, August 30: Formulating Problems lecture pdf A game with graphics and introduction to Tkinter lecture pdf</p> <p>Reading: GD formulation P1 due; P2 out</p>	<p>Thursday, August 31: wicked problems lecture pdf Running SOLUZION from IDLE; Demo: Formulating Tic-Tac-Toe lecture pdf</p> <p>Reading: GD wicked problems</p>	<p>Friday, September 1: Game structures lecture pdf Project ideas and planning lecture pdf</p> <p>Reading: GD game structures P2 due; P3 out</p>
<p>Tuesday, September 5: [Quiz 1 covering game structures] Introduction to software development with Scrum lecture pdf</p> <p>P3 Milestone A due</p>	<p>Wednesday, September 6: The Prisoner's Dilemma and related obstacles to problem solving lecture pdf P3A presentations of game ideas lecture pdf</p> <p>Reading: GD Prisoners Dilemma</p>	<p>Thursday, September 7: Featured guest: Maggie Ryan (CSE undergraduate program advisor) [Quiz 2 covering the prisoner's dilemma] Lab work, Scrum practice, more on Python functions lecture pdf</p>	<p>Friday, September 8: Learning curves lecture pdf Lab work lecture pdf</p> <p>Reading: GD learning curves P3 Milestone B due</p>
<p>Tuesday, September 12: [Quiz 3 on learning curves, theory of problem solving, and wicked problems] Wicked Problem Case Study: Homelessness lecture pdf</p> <p>P3 Milestone C due</p>	<p>Wednesday, September 13: Lab work, Scrum practice, and presentations of game ideas lecture pdf</p>	<p>Thursday, September 14: Iterative design of games; lab work lecture pdf P3 Milestone D due</p>	<p>Friday, September 15: Final presentations, demos, and game evaluations lecture pdf P3 Milestone E due</p>

last updated: September 7, 2017.

Student Population

- 25 incoming freshmen
- 5 female, 20 male
- 16 international students (from China, India, Japan, Korea, Lebanon, Singapore)
- 9 domestic (US)

Prior Programming Experience

Amount of Experience	Number (and percentage)	
None	3	(12%)
Up to about 1 month	6	(24%)
1 to 6 months	3	(12%)
More than 6 months	13	(52%)

Diversity of Programming Exp.

Question:

If you did have some experience with computer programming, please describe that experience ... was it a class, a code camp, etc? Was it self-taught? What sort of program did you create? (a game, an animation, and interactive experience, a computation?). What language(s) did you use (for example Java)?

I learned C++ by myself. The programs I created were console programs. Most of them did calculations. Some of them printed things in a screen. I used complex loops and functions. I learned about classes but never used them. I made a tictactoe game where the user has to input the position of their letter by typing it. I gave the program an anti-error loop; if the user inputs an unacceptable number, the program doesn't crash.

I took AP Computer Science my senior year of High School, where I learned Java. Also, over the summer I have begun to familiarize myself with Unity and C#.

I learned Java and Python via MOOC. Currently I'm trying to create plugins for Minecraft.

Java

I studied Java for all four years of high school and made a couple of basic games like Space Invaders and Brick Breaker as side projects by myself. I taught myself basics of Python and some Swift

I have been programing in C++ for around 4 years. I have done some Java programing to prepare for AP Computer Science. I also taken some courses on HTML and CSS. I am familiar on a pretty basic level with Python.

I have studied Computer Science at School as my 5th subject in my High School. I have studied C++ including arrays, loops, lists etc. The programs created by me hosts calculators, data management software, sorting, searching etc.

I learned Java, C, and python for 1 month each in the camp.

Server maintenance in an existing system using Java and SQL. App creation for Amazon Alexa in Node.js

I studied IB Diploma Computer Science, in which I studied Java and focused on producing database programs.

Code camp, game, java

Students' Hopes/Motivations

(optional) What do you hope to get out of taking this EFS course? ...

I'm hoping I could tell for sure if CS is the best major for me. I did get a lot of satisfaction from making programs that ran successfully but it was also very frustrating at times. I want to be able to determine if the CS path is best suited for my mental capabilities.

To get a better idea for how people think when trying to make and develop games.

Collaboration skills. Looking at different aspects of a problem and learning how to approach effectively.

To be fluent in python and learn how to solve real world problems.

I hope this course will ehlp me further my (probably fairly rudimentary)knowledge of computer science.

I hope to be confident in my Python skills.

I have always been interested in learning more about computers ,so by attending this EFS Course I will be able to learn one more type of computer language which is Python. I am also very keen in playing video games from my childhood and have always had a will to help how these amazing things work so effortlessly.

Capability to make a program or game

Game design skills and basic python skills

I wish to learn more about coding in python and better structuring development projects.

solid basis for coding and making games

(optional reasons for taking this EFS course)

No response

Get more insightful understanding of programming, and meet friends who have mutual interest.

A deeper understanding of object oriented programming

Summary of Students' Hopes

- Improve Python skills
- Feel for CS as a possible major
- Learn the thinking of game designers
- Collaboration skills
- Meet friends with similar interests
- Learn to solve real-world problems

Classical Theory of Problem Solving

- A key course component
- Origins of the theory: early A.I.
- Essential terms in the theory
- Problem-space diagram

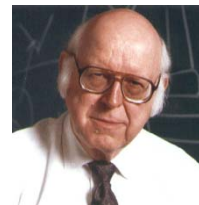
The Classical Theory of Problem Solving

- Began with the General Problem Solver (GPS) G.W. Ernst, Alan Newell & Herbert Simon.
- GPS worked by successively applying operators to try to reduce the distance from a current state to a goal state.

The CTPS is well described in several AI textbooks, e.g.,

- Judea Pearl: “Heuristics: Intelligent Search Strategies for Computer Problem Solving”
- S. Russell & P. Norvig. “Artificial Intelligence: A Modern Approach”

Newell



Simon



Pearl



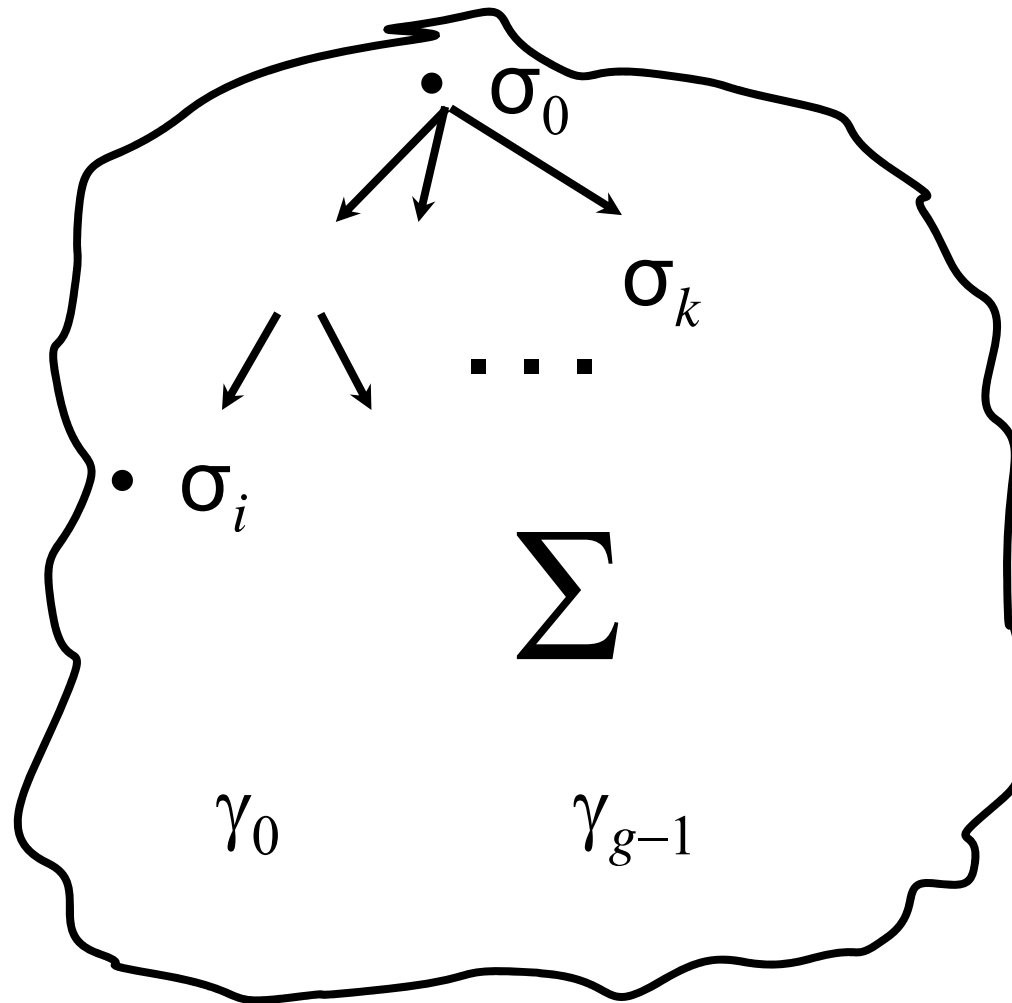
Definition

- A problem is a triple: (σ_0, Φ, Γ) where σ_0 is an initial state, Φ is a set of operators, and Γ is a set of goal states.
- Each $\phi_i \in \Phi$ has a precondition, and a state-transformation function.
- These implicitly define Σ , the set of all states reachable from σ_0 by applying members Φ zero or more times.

initial state

other
reachable
states

goal state(s)




State
Space

Outline

- Introduction
- A Course on Games for Problem Solving
- **Problem Formulation**
- Wicked Problems
- Towards Standard Formats
- Liveness in Problem Solving Environments
- Design vs Problem Solving
- Other Issues
- Peer Critiques in the Programming Lab
- Concluding Thoughts

Example: Towers of Hanoi

- $P = (\sigma_0, \Phi, \Gamma)$

- σ_0 : 

- $\Phi : \{ \text{Move1_2, Move1_3, Move2_3, Move2_1, Move3_1, Move3_2} \}$

- $\Gamma = \{\gamma\}$:



Outline

- Introduction
- A Course on Games for Problem Solving
- Problem Formulation
- **Wicked Problems**
- Towards Standard Formats
- Liveness in Problem Solving Environments
- Design vs Problem Solving
- Other Issues
- Peer Critiques in the Programming Lab
- Concluding Thoughts

Formulation of Wicked Problems

- Some wicked problems
- The Rittel & Webber characteristics of wicked problems
- The formulation process



“Getting a Handle on
Wicked Problems”

A Wicked Problem:

Slowing global warming



More Wicked Problems

- Stopping the spread of antibiotic-resistant diseases
- Halting nuclear proliferation
- Ending homelessness in King County, WA
- Avoiding species extinction
- Providing all citizens with health care
- Colonizing Mars

Rittel-Webber Characteristics 1-5 of 10

1. There is no definitive formulation of a wicked problem
2. Wicked problems have no stopping rule
3. Solutions to wicked problems are not true-or-false, but good-or-bad
4. There is no immediate and no ultimate test of a solution to a wicked problem
5. Every solution to a wicked problem is a “one-shot operation”; because there is no opportunity to learn by trial-and-error, every attempt counts significantly

Rittel-Webber Characteristics 6-10 of 10

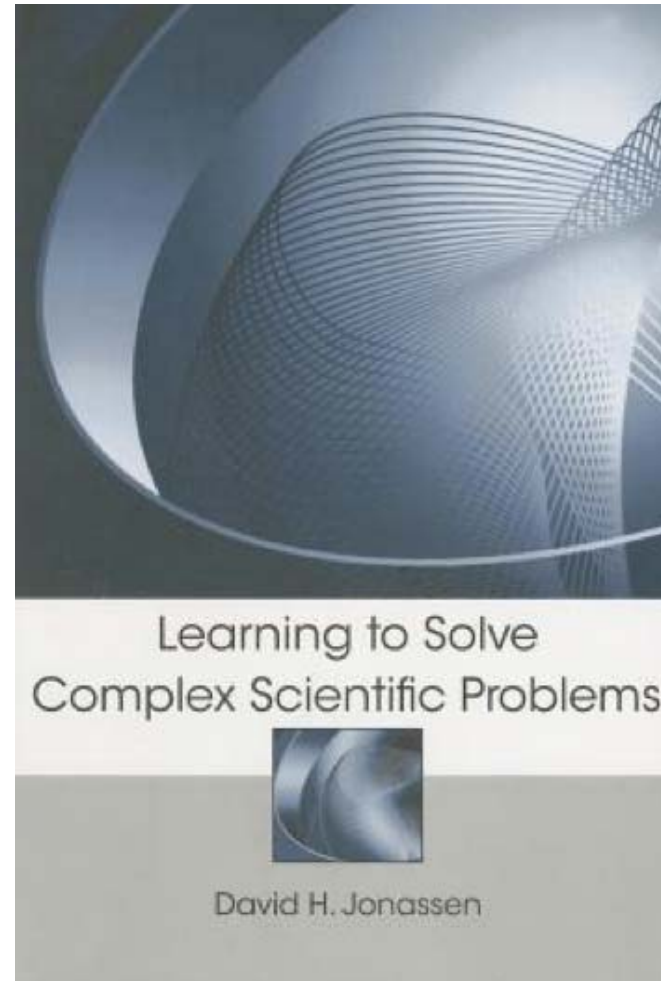
6. Wicked problems do not have an enumerable (or an exhaustively describable) set of potential solutions, **nor is there a well-described set of permissible operations** that may be incorporated into the plan
7. Every wicked problem is essentially unique
8. Every wicked problem can be considered to be a symptom of another problem
9. The existence of a discrepancy representing a wicked problem can be explained in numerous ways. The choice of explanation determines the nature of the problem's resolution
10. **The planner has no right to be wrong**

The Formulation Process

adapted from:

"Learning to Solve Complex Scientific Problems"

edited by David Jonassen



Steps in Problem Formulation

- Describing a need
- Identifying resources
- Restriction and simplification
- Designing a state representation
- Designing a set of operators
- Listing constraints and desiderata
- Specifying in code the state representation, operators, constraints, evaluation criteria, and goal criterion.
- Specifying in code a state visualization method.
- If appropriate, providing for multiple roles within teams of solvers.

Steps in Problem Formulation

- Describing a need
 - Identifying resources
 - Restriction and simplification
 - Designing a state representation
 - Designing a set of operators
 - Listing constraints and desiderata
 - Specifying in code the state representation, operators, constraints, evaluation criteria, and goal criterion.
 - Specifying in code a state visualization method.
 - If appropriate, providing for multiple roles within teams of solvers.
- (Preformulation)

Steps in Problem Formulation

- Describing a need
 - Identifying resources
 - Restriction and simplification
 - Designing a state representation
 - Designing a set of operators
 - Listing constraints and desiderata
 - Specifying in code the state representation, operators, constraints, evaluation criteria, and goal criterion.
 - Specifying in code a state visualization method.
 - If appropriate, providing for multiple roles within teams of solvers.
- (Preformulation)
- (Posing)

Steps in Problem Formulation

- Describing a need
 - Identifying resources
 - Restriction and simplification
 - Designing a state representation
 - Designing a set of operators
 - Listing constraints and desiderata
 - Specifying in code the state representation, operators, constraints, evaluation criteria, and goal criterion.
 - Specifying in code a state visualization method.
 - If appropriate, providing for multiple roles within teams of solvers.
- (Preformulation)
- (Posing)
- (Coding the formulation)

Outline

- Introduction
- A Course on Games for Problem Solving
- Problem Formulation
- Wicked Problems
- **Towards Standard Formats**
- Liveness in Problem Solving Environments
- Design vs Problem Solving
- Other Issues
- Peer Critiques in the Programming Lab
- Concluding Thoughts

Why Common Problem Formats?

- Problem formulations can share libraries, esp. for state visualization.
- Solving infrastructure and game engines can be shared among problems.
 - Breadth-First & Heuristic Search implementations.
 - Autoplayers,
 - General player interfaces.
- Debugging and testing tools can also be shared: operator validation, problem-state class unit tests.

Formulation File Excerpt

```
''' TowersOfHanoi . py
A QUI ET2 Sol ving Tool problem formulati on.
QUI ET = Quetzal User Intelligence Enhanci ng Technol ogy.
'''

#<METADATA>
QUI ET_VERSION = "0. 2"
PROBLEM_NAME = "Towers of Hanoi "
PROBLEM_VERSION = "0. 2"
PROBLEM_AUTHORS = [' S. Tanimoto' ]
PROBLEM_CREATION_DATE = "11-OCT-2017"
PROBLEM_DESC=\
''' This formulati on of the Towers of Hanoi problem uses generic
Python 3 constructs and has been tested with Python 3. 6.
It is designed to work according to the QUI ET2 tools interface.
'''

#</METADATA>
```

Formulation File Excerpt (cont.)

```
#<COMMON_CODE>
class State:
    def __init__(self, d):
        self.d = d

    def __eq__(self, s2):
        for p in ['peg1', 'peg2', 'peg3']:
            if self.d[p] != s2.d[p]: return False
        return True

    def __hash__(self):
        return (self.__str__()).__hash__()

    def copy(self):
        # Performs an appropriately deep copy of a state,
        # for use by operators in creating new states.
        news = State({})
        for peg in ['peg1', 'peg2', 'peg3']:
            news.d[peg]=self.d[peg][: ]
        return news
```

Formulation File Excerpt (cont.)

```
def can_move(self, From, To):
    '''Tests whether it's legal to move a disk in state s
        from the From peg to the To peg.'''
    try:
        pf=self.d[From] # peg disk goes from
        pt=self.d[To]    # peg disk goes to
        if pf==[]: return False # no disk to move.
        df=pf[-1] # get topmost disk at From peg..
        if pt==[]: return True # no disk to worry about at To peg.
        dt=pt[-1] # get topmost disk at To peg.
        if df<dt: return True # Disk is smaller than one it goes on.
        return False # Disk too big for one it goes on.
    except (Exception) as e:
        print(e)
```


Formulation File Excerpt (cont.)

```
def goal_test(s):  
    '''If the first two pegs are empty, then s is a goal state.'''  
    return s.d['peg1']==[] and s.d['peg2']==[]
```

Formulation File Excerpt (cont.)

```
#<INITIAL_STATE>
INITIAL_DICT = \
    {'peg1': list(range(N_disks, 0, -1)), 'peg2': [], 'peg3': [] }
CREATE_INITIAL_STATE = lambda: State(INITIAL_DICT)
#DUMMY_STATE = {'peg1': [], 'peg2': [], 'peg3': [] }
#</INITIAL_STATE>

def goal_test(s):
    '''If the first two pegs are empty, then s is a goal state.'''
    return s.d['peg1']==[] and s.d['peg2']==[]
```

Formulation File Excerpt (cont.)

```
#<OPERATORS>
peg_combinations = [('peg'+str(a),'peg'+str(b)) for (a,b) in
                    [(1,2),(1,3),(2,1),(2,3),(3,1),(3,2)]]
OPERATORS = [Operator("Move disk from "+p+" to "+q,
                      lambda s,p1=p,q1=q: s.can_move(p1,q1),

                      # The default value construct is needed
                      # here to capture the values of p&q separately
                      # in each iteration of the list comp. iteration.

                      lambda s,p1=p,q1=q: s.move(p1,q1) )
              for (p,q) in peg_combinations]
#</OPERATORS>
```

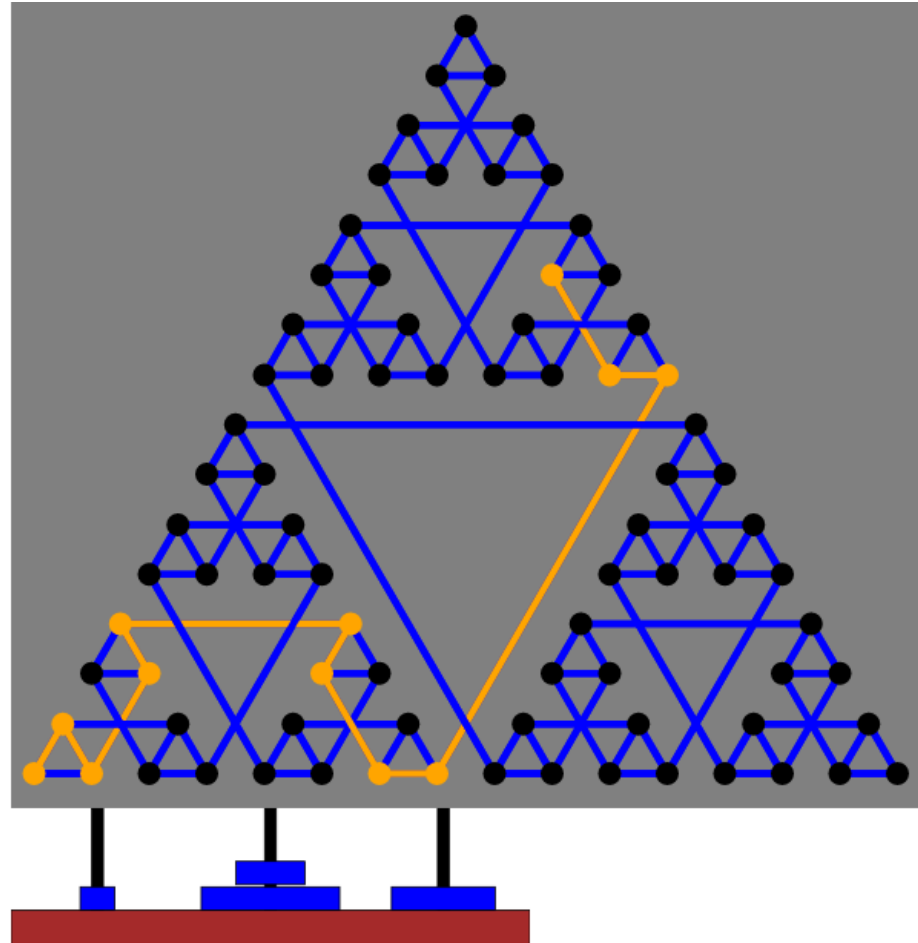
Outline

- Introduction
- A Course on Games for Problem Solving
- Problem Formulation
- Wicked Problems
- Towards Standard Formats
- **Liveness in Problem Solving Environments**
- Design vs Problem Solving
- Other Issues
- Peer Critiques in the Programming Lab
- Concluding Thoughts

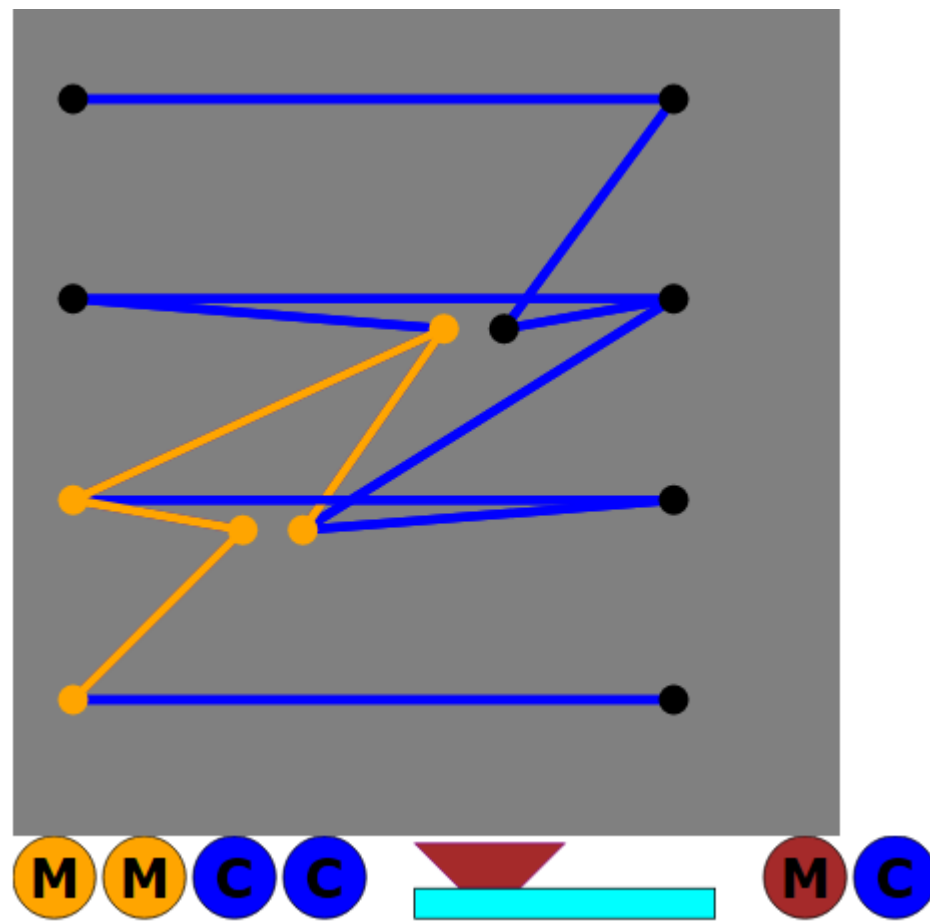
Live Solving

- A process in which a human solver manipulates the state of a problem quickly with the help of a computer.
- There could also be a performance component of the experience, not unlike live coding to produce music.

Live Solving Towers of Hanoi



Live Solving Missionaries and Cannibals



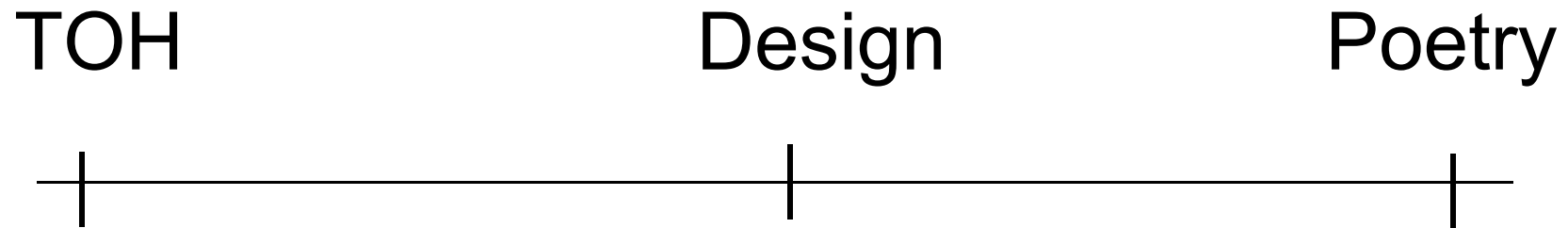
Why Live Solving?

- Reinforces understanding of the theory
- Shows all problems are like mazes
- Supports discussion of a problem's characteristics
- Can transform any well-formulated problem into a puzzle-like experience.

Outline

- Introduction
- A Course on Games for Problem Solving
- Problem Formulation
- Wicked Problems
- Towards Standard Formats
- Liveness in Problem Solving Environments
- **Design vs Problem Solving**
- Other Issues
- Peer Critiques in the Programming Lab
- Concluding Thoughts

A Continuum of Problem Types*



TOH: complete agreement on solution

Design: reasonably wide agreement

Poetry: somewhat less agreement

*Scott Klemmer, personal communication.

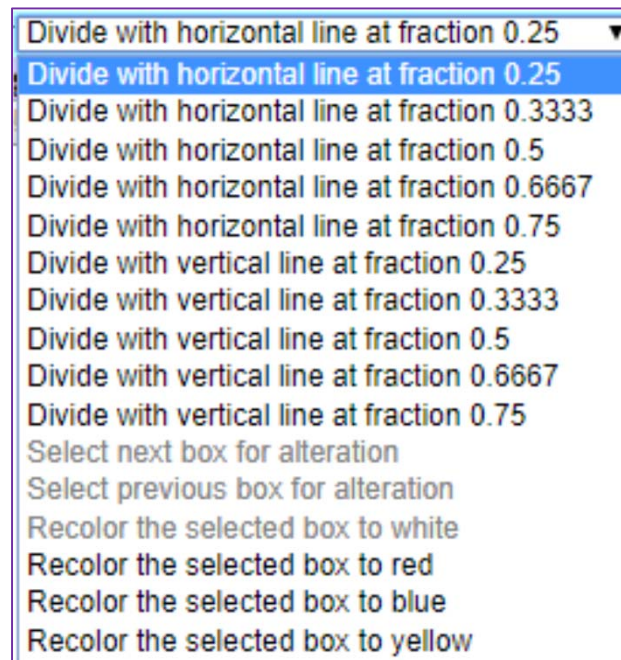
H. Simon, in the *Sciences of the Artificial*, makes the case that the classical theory of problem solving is applicable to nearly the full spectrum of human creative endeavors.

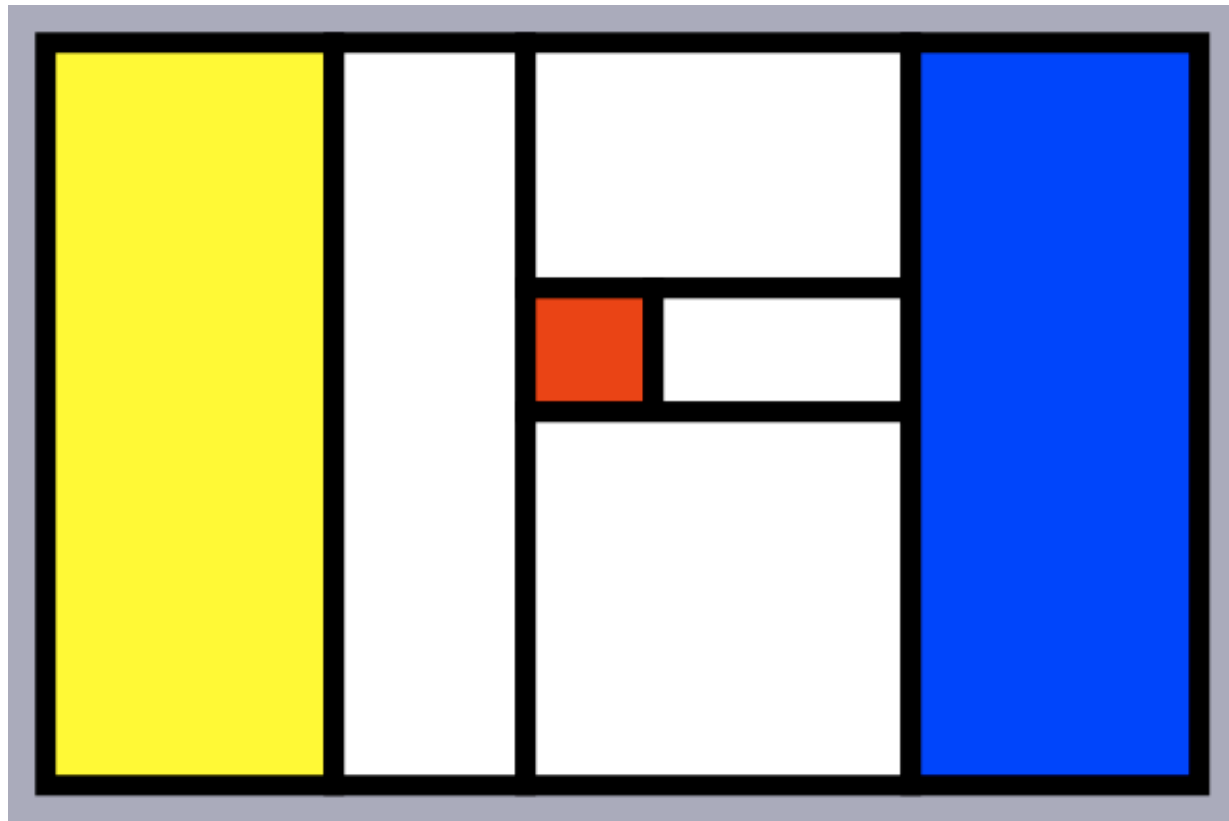
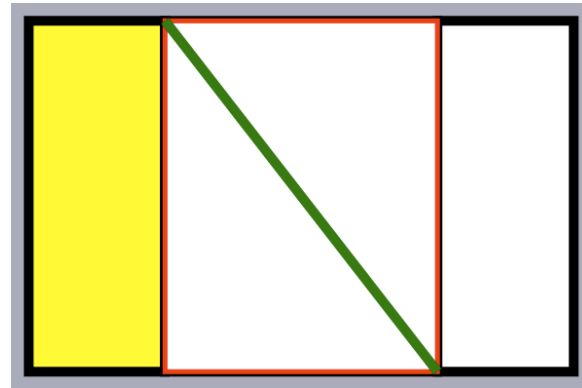
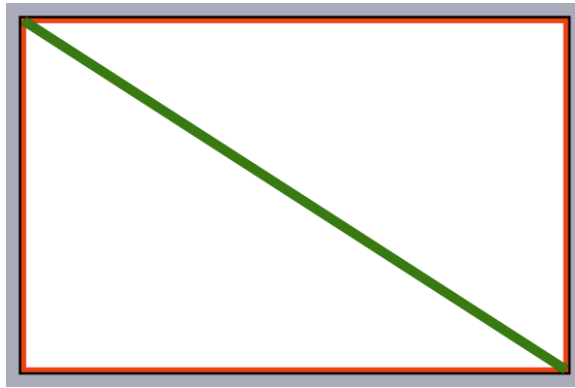
Design as Problem Solving

- Consider Recursive Mondrianization:
- Mondrianization
- Mondrianization

Design as Problem Solving

- Consider Recursive Mondrianization:
- Mondrianization
- Mondrianization



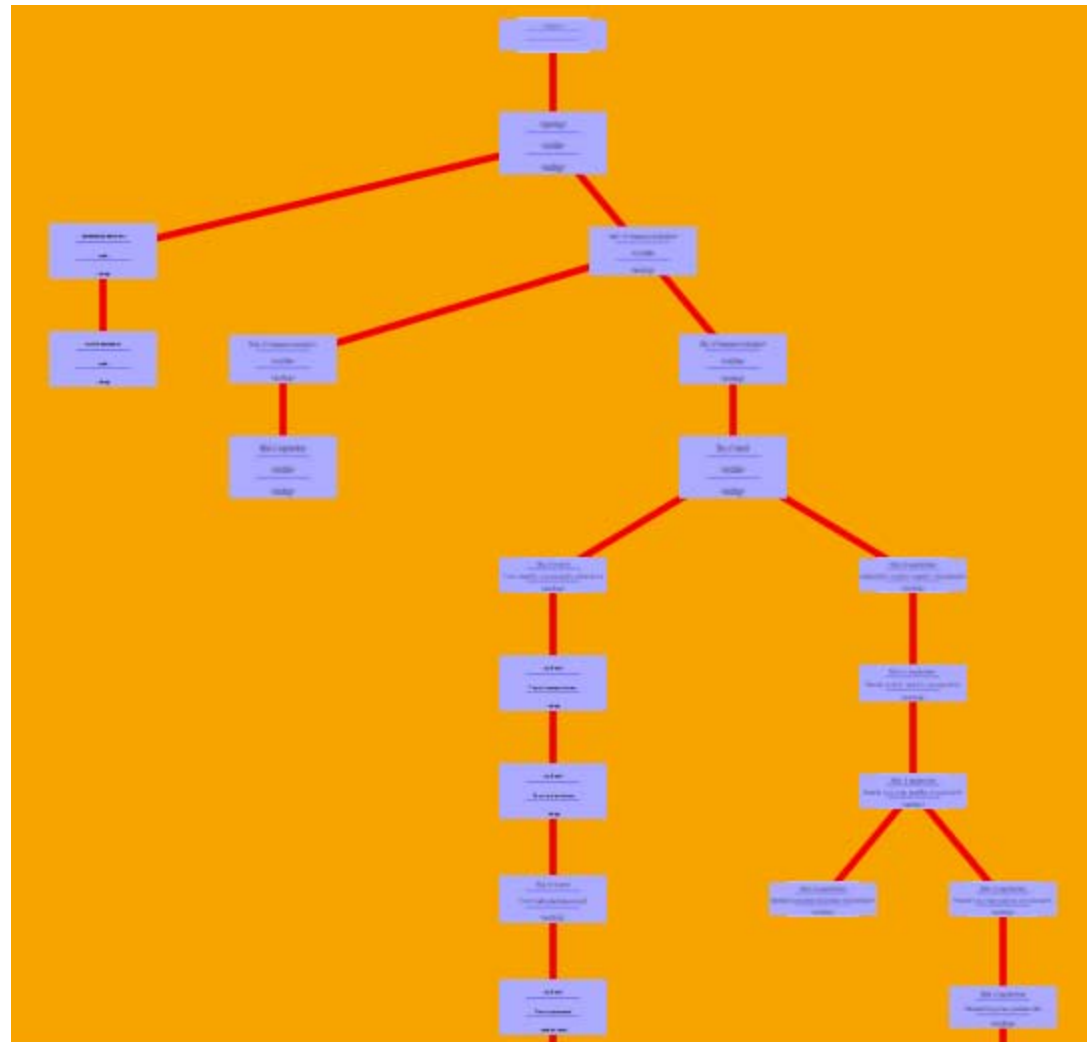


Poetry as Recursive Selection

<haiku>

Poetry as Recursive Selection

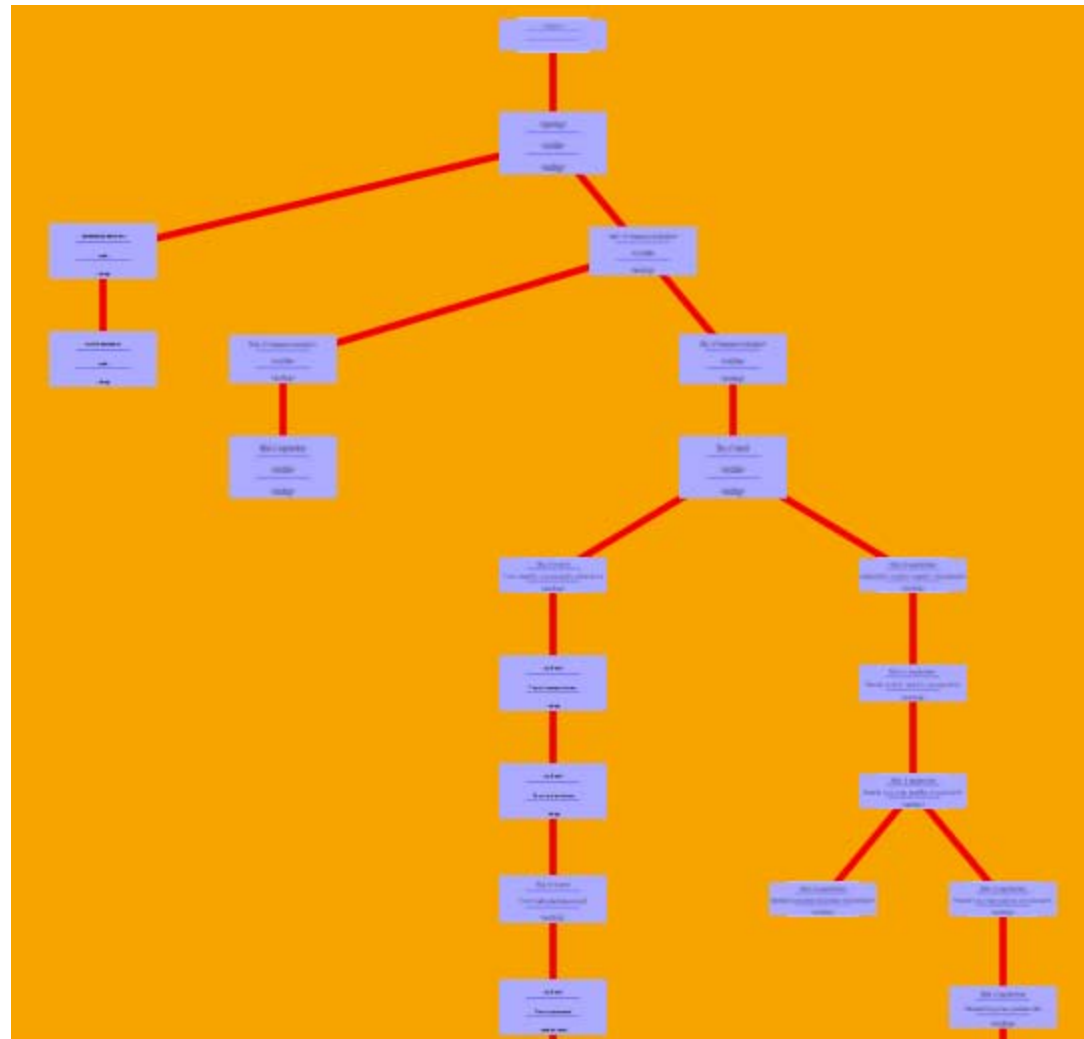
<haiku>



Poetry as Recursive Selection

<haiku>

<season-indicator> -> september
<season-indicator> -> october
<season-indicator> -> november
<season-indicator> -> december
<season-indicator> -> frosty
<season-indicator> -> sweltering
<season-indicator> -> leaves falling
<season-indicator> -> snowflake
<quantifier> -> the
<quantifier> -> many a
<quantifier> -> one
<opening> -> <season-indicator> <color> <water>
<opening> -> <air> of <season-indicator>
<middle> -> <movement> <direction> <quantifier> <earth>
<middle> -> from <earth> <movement> <direction>
<middle> -> <direction> <color> <earth> <movement>
<ending> -> <intensifier> <sound>
<ending> -> <sound> and <sound>
<n> -> [NL]
<haiku> -> <opening> [NL] <middle> [NL] <ending>
<haiku> -> <opening> [NL] <middle> [NL] <ending>



Poetry as Recursive Selection

October brown brook
From pasture tumbles downhill
Remarkable Crash

Outline

- Introduction
- A Course on Games for Problem Solving
- Problem Formulation
- Wicked Problems
- Towards Standard Formats
- Liveness in Problem Solving Environments
- Design vs Problem Solving
- **Other Issues**
- Peer Critiques in the Programming Lab
- Concluding Thoughts

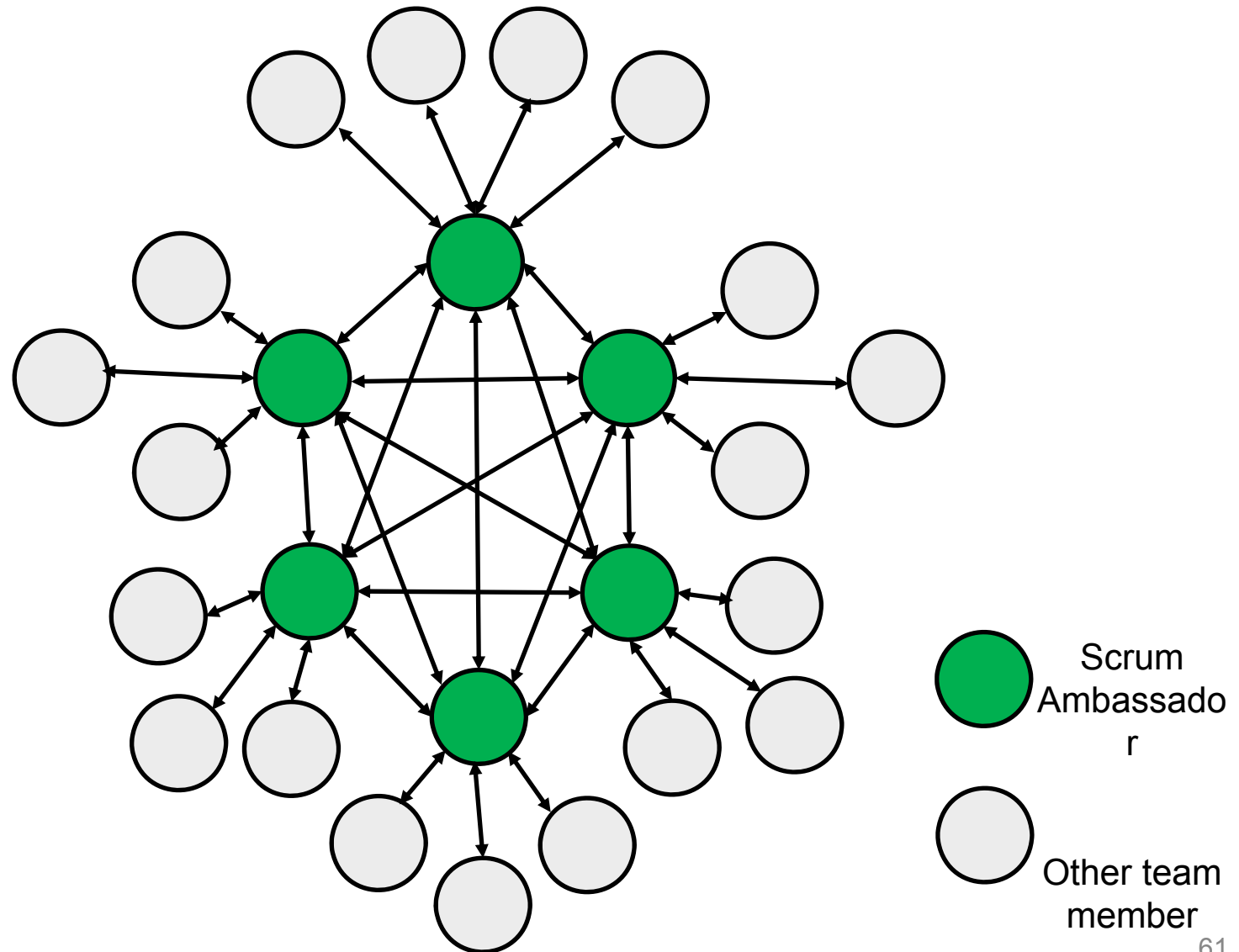
Other Issues

- Tkinter vs. Web Programming
 - CSE 190C vs CSE 190D
- Meta design: Game Design as a Problem
 - When a game embeds a problem formulation, designing the game is solving a metaproblem.

Outline

- Introduction
- A Course on Games for Problem Solving
- Problem Formulation
- Wicked Problems
- Towards Standard Formats
- Liveness in Problem Solving Environments
- Design vs Problem Solving
- Other Issues
- **Peer Critiques in the Programming Lab**
- Concluding Thoughts

Flow Diagram for Peer Evals.



Peer Evaluation Schedule

- Milestone A: Initial oral presentation of ideas; oral questions and comments
- Milestone **B**: Short formative written peer reviews (using a 1/2 page form ...)
- Milestone C: no peer feedback
- Milestone **D**: More formal 2-page peer evaluations of semi-final games ...
- Milestone E: Summative peer ratings on multiple scales

Milestone B review form

Review by: _____ Game Title: _____

Group: (circle 1): INFO, MENTAL, HEALTH, CLIMATE, VIOLENCE, BIO.

Initial State: Suggestions for additional state variables or for removing variables.

State visualization: What graphical rep. of states might add to what's here?

Are the operators clear? Are they sufficient? Do they work?

What might be a couple of additional operators that would add something new or cool to the game?

What scoring or progress indicators are there and how could they be made more compelling?

Other suggestions?

Milestone D peer review questions

Review by: _____

Game Title: _____

1. Clarity of Goal or Objective: Suggestions for how to make it clearer what the goal of the game is.
2. State visualization: How clear is the state visualization? Should any textual explanation be added?
3. Are the operators clear? Are they sufficient? Do they work?
4. Credibility of the Model: Is the underlying model rich enough to be a credible basis for this game? How many state variables are there, and to the operators really relate to them in the real-world wicked problem? If not, what would be a reasonable storyline for why the fictional operators or variables are OK for the game? (E.g., the focus is on other variables or some other important aspect of the player's understanding of the problem.)
5. Time frame: If this game uses a simulation of a process over time, is the time frame clearly presented? Is it credible? (Could the effect of the operators actually manifest themselves over the short(?) period represented by a turn or simulation step? If not, what should the game designers do?
6. What might be a couple of additional operators that would add something new or cool to the game?
7. Expected level of engagement: If you were playing this game, how engaged would you feel? What can the designers do to increase your feeling of engagement?
8. What scoring or progress indicators are there and how could they be made more compelling?
9. What would you be learning about the wicked problem when you play this game? What more would you WANT to be learning, but are not? What could the game designers do about that?
10. Other suggestions?

Sample Response to Feedback

- (separate document).
- The Final Project Reports largely consist of explanations of how the team responded to the suggestions of the class in the Milestone D peer reviews.

Parallels to Writers' Workshops

- Critiques may be more thorough when a highly structured process is followed.
- Peers see each work from the perspective of someone in a similar role.
- Rules related to critiquing the work rather than the authors are not needed, because the review form focuses on the work.

- UTOPIA THEORY -



"Yeah, I see him too...But nobody wants to talk about it!"

A FLY ON THE WALL

OMG! The
things I've
heard!



©nakedpastor

edhayward.

Outline

- Introduction
- A Course on Games for Problem Solving
- Problem Formulation
- Wicked Problems
- Towards Standard Formats
- Liveness in Problem Solving Environments
- Design vs Problem Solving
- Other Issues
- Peer Critiques in the Programming Lab
- **Concluding Thoughts**

Concluding Thoughts

- Observations ...
- Beliefs ...
- Future Work ...

Observations

- Students are learning programming in a diversity of contexts (from the background questionnaire).
- The classroom context provides a social context for programming, with some similarities to a corporate context.
- Agile methodology needed some adaptation for use in short, introductory programming contexts. (shorter, less formal sprints; whole class as project owner).

Beliefs

- Programming as a means to an end (game creation) can help smooth over this variability in student programming backgrounds.
- Live solving offers a reconceptualization of what it means to solve, but reinforces an understanding of the classical theory.

Future Work

- Preparation for a 2nd EFS offering:
 - Make a more complete set of readings.
 - Offer more software tools for game testing and evaluation:
 - Autoplayer, State-space Analyzer, Generic Livesolver
 - Identify a good online peer-review/rating support tool that can aggregate ratings and keep constructive comments organized (by reviewer and by issue or feature).
- Preparation for an advanced course
 - Integrate into the CSE curriculum as a senior capstone design course.
 - Organize a set of tools to better support design of collaborative games that work on the web.
- Possible book on how to formulate wicked problems for human/computer solving.

Acknowledgments

- ❖ **Collaborators:** Sandra Fan, Brian Johnson & Tyler Robison
- ❖ **CoSolve developers:** Rob Thompson, Laura Dong, Chris Brenan, Yizhou Wang, Christopher Clark
- ❖ **Game design study:** Rolfe Schmidt, Yun-En Liu, Tyler Robison, Sandra Fan, Brian Johnson
- ❖ **Problem Template creators:**
Jordan Atwood, Charliz Burks, Cezanne Camacho, Michael Duong, Katherine Hulsman, Galen Knapp, Richard Rice, Yifan Zhang
- ❖ **National Science Foundation** under grant 0613550

Thank you