

# What is PCG in games?

## Procedural Content Generation

Artificial Intelligence for  
Interactive Media and Games

Professor Charles Rich  
Computer Science Department  
rich@wpi.edu

[Based on lecture by Julian Togelius, IT University of Copenhagen]

CS/IMGD 4100 (C-18)

- Procedural Generation: with no or limited human intervention, algorithmically
- of Content: *not* NPC behaviour, *not* the game engine, things that affect gameplay
- in Games: computer games, board games... any kind of games

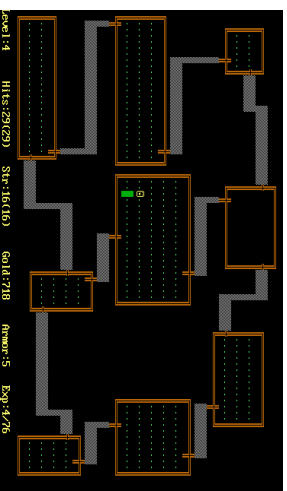
Tuesday, September 3, 2010

## Game content, e.g.

- Levels, tracks, maps, terrains, dungeons, puzzles, buildings, trees, grass, fire, plots, descriptions, scenarios, dialogue, quests, characters, rules, boards, parameters, camera viewpoint, dynamics, weapons, clothing, vehicles, personalities...

## History: Runtime random level generation

- Rogue-2D



1980

Tuesday, September 3, 2010

## History: Runtime random level generation

- Tribal Trouble



2005

## Civilization IV



2005

Tuesday, September 3, 2010

## History: Runtime random level generation

- Dwarf Fortress-3D

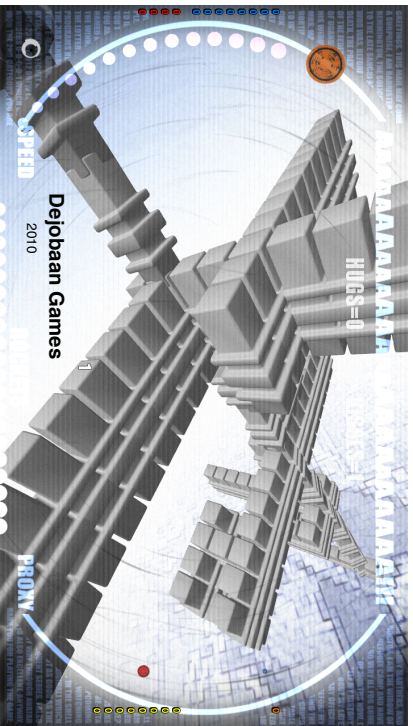


2007

## Diablo



2008



Dejobaan Games  
2010

## Sudoku

9		1						5
8		5	9		2			1
			4					
			7		8			
				2	6			
								9
2			3					6
			2			9		
		1	9		4	5	7	

## The future...

- Can we drastically cut game development costs by creating content automatically from designers' intentions?
- Can we create games that adapt their game worlds to the preferences of the player?
- Can we create endless games?
- Can the computer circumvent or augment limited human creativity and create new types of games?

# A taxonomy of PCG

In general,

## PCG > randomness

- Online/Offline
- Necessary/Optional
- Random seeds/Parameter vectors
- Stochastic/Deterministic
- Constructive/Generate-and-test

Friday, September 24, 2010

Friday, September 3, 2010

## Online/Offline

- Online: as the game is being played
- Offline: during development of the game

## Necessary/Optional

- Necessary content: content the player needs to pass in order to progress
- Optional content: can be discarded, or bypassed, or exchanged for something else

Friday, September 3, 2010

Friday, September 3, 2010

## Stochastic/ Deterministic

- Deterministic: given the same starting conditions, always creates the same content
- Stochastic: the above is not the case

## Random seeds/ Parameter vectors

- a.k.a. dimensions of control
- Can we specify the shape of the content in some meaningful way?

Friday, September 3, 2010

Friday, September 3, 2010

# Constructive/ Generate-and-test

- Constructive: generate the content once and be done with it
- Generate-and-test: generate, test for quality, and re-generate until the content is good enough

Friday, September 3, 2010

# The Search-based Paradigm

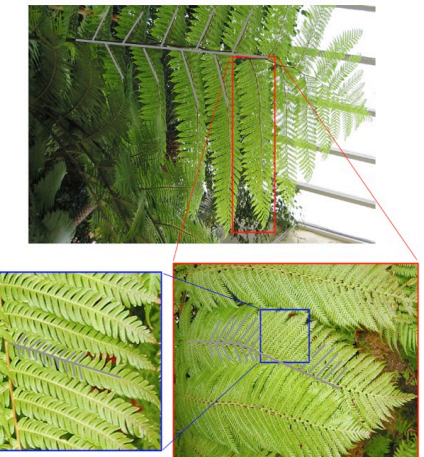
- A special case of generate-and-test:
- The test function returns a numeric fitness value (not just accept/reject)
- The fitness value guides the generation of new candidate content items
- Usually implemented through **evolutionary computation**

Friday, September 3, 2010

## Plants?

- Core feature of the natural world... therefore of many games
- Need for believability
  - Infinitely detailed
  - Similar and recognizable, but not identical
- Need for compact representation
- Need for automatic large-scale generation

Friday, September 17, 2010



## Self-similarity

Friday, September 17, 2010

## SpeedTree



Friday, September 17, 2010

## Self-similarity

- Nature has obviously thought out some clever way of representing complex organisms using a compact description...
- ...permitting individual variation...
- ...why is this relevant for us?

Friday, September 17, 2010

# L-systems

- Introduced by Aristid Lindenmeyer 1968, to model plant development
- Creates strings (text) from an *alphabet* based on a *grammar* and an *axiom*
- Closely related to Chomsky grammars (but productions carried out in parallel, not sequentially)

Friday, September 17, 2010

# An example L-system

- Alphabet: {a, b}
- Production rules (grammar):  
a → ab  
b → a
- Axiom: b

Example of a derivation in a DOL-System

```
b
 |
a
 |
ab
 |
aba
 |
abaa
 |
abaaaba
```

Friday, September 17, 2010

# Types of L-systems

- **Context-free**: production rules refer only to an individual symbol
- **Context-sensitive**: productions can depend on the symbol's neighbours
- **Deterministic**: there is exactly one production for each symbol
- **Stochastic**: several productions for a symbol

Friday, September 17, 2010

# A graphical interpretation of L-systems

- Invented/popularized by Prusinkiewicz 1986
- Core idea: interpret generated strings as instructions for a **turtle** in turtle graphics
- Read the string from left to right, changing the state of the turtle (x, y, heading)

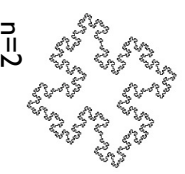
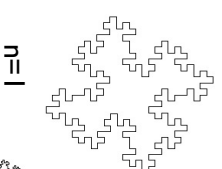
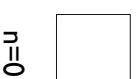
Friday, September 17, 2010

# Example graphical L-system

- Alphabet: {F, f, +, -}
- F: move the turtle forward (drawing a line)
- f: move the turtle forward (don't draw)
- +/-: turn right/left (by some angle)

# Graphical L-system

- axiom: F++F++F
- grammar:  
F → F+F-F-FF+F+F-F
- Turning angle: 90°



Friday, September 17, 2010

Friday, September 17, 2010

# Bracketed L-systems

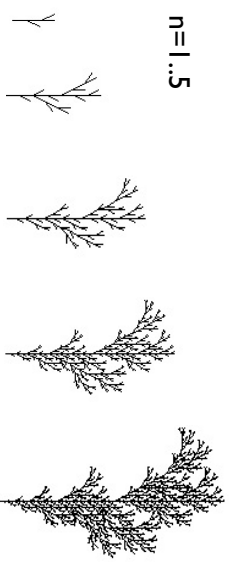
- Alphabet: {F, f, +, -, [, ]}
- **[**: push the current state (x, y, heading of the turtle) onto a pushdown stack
- **]**: pop the current state of the turtle and move the turtle there *without drawing*
- Enables branching structures!

Friday, September 17, 2010

# Bracketed L-systems

- Axiom: F
- Grammar:  $F \rightarrow F[F][F][F][F]$
- Turning angle:  $30^\circ$

$n = 1..5$



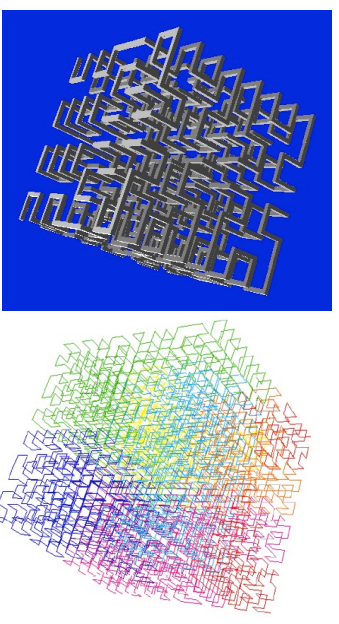
Friday, September 17, 2010

# 3D graphics

- Turtle graphics L-system interpretation can be extended to 3D space:
- Represent state as x, y, z and pitch, roll, yaw
- +, -: turn (yaw) left/right
- &, ^: pitch down/up
- \, /: roll left/right (counterclockwise/clockwise)

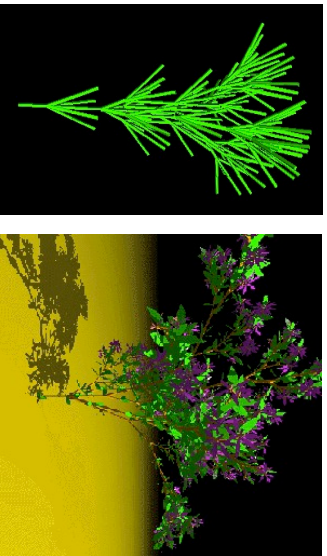
Friday, September 17, 2010

# 3D interpretation of L-systems



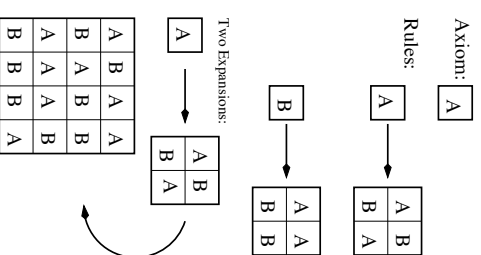
Friday, September 17, 2010

# 3D interpretation of bracketed L-systems



Friday, September 17, 2010

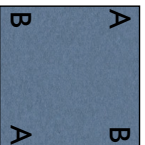
# 2D L-systems



Friday, September 17, 2010

## Terrain interpretation of 2D L-systems

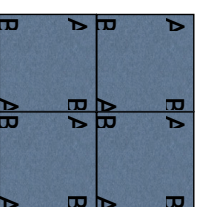
- Each group of four letters is interpreted as instructions for lowering or raising the corners of a square
- e.g.  $A=+0.5, B=-0.5$



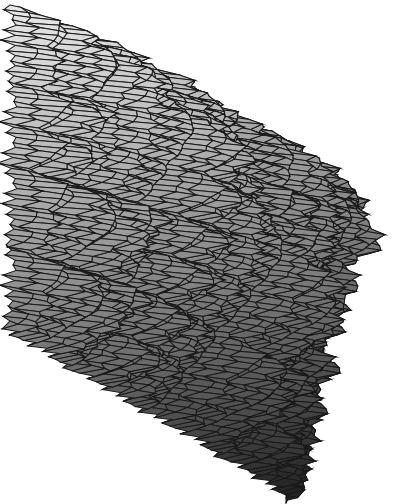
Friday, September 17, 2010

## Terrain interpretation of 2D L-systems

- In next iteration, the 2D L-system is rewritten once, and each square is divided into two
- “Doubling the resolution”



Friday, September 17, 2010



Friday, September 17, 2010

## Evolving L-systems

- How can we combine L-systems with evolutionary computation?

## Evolutionary computation?

- Keep a **population** of candidates
- Measure the **fitness** of each candidate
- Remove the worst candidates
- Replace with copies of the best (least bad) candidates
- **Mutate/crossover** the copies

Friday, September 17, 2010

## Evolving L-systems

- Evolving the axiom
- Evolving the grammar:
  - change the shape of one or more production rules, or
  - add/remove/replace productions
  - counter limits
- Evolving the interpretation:
  - Evolve production probabilities
  - Evolve other aspects (e.g. turning angles)

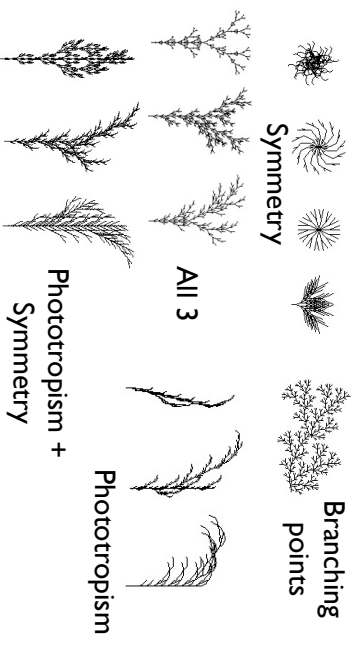
Friday, September 17, 2010

## Fitness functions

- Phototropism
- Bilateral symmetry
- Proportion of branching points

Friday, September 17, 2010

## Evolved L-systems



Friday, September 17, 2010

## Multiobjective Exploration of the StarCraft Map Space

Julian Togelius, Mike Preuss,  
Nicola Beume, Simon Wessing,  
Johan Hagelbäck and Georgios N. Yannakakis

2010 IEEE Symposium on Computational Intelligence and Games

Friday, September 24, 2010

- ### Why generate maps?
- Give players an unlimited supply of new, unpredictable maps
  - Negates rote learning advantages
  - Dynamically adapt the game to individual players' strengths...
  - ...or to groups of players!
  - Help designers generate more novel and balanced maps
  - Help them with the "boring stuff"

Friday, September 24, 2010

- Classic real-time strategy game
- Korea's unofficial national sport
- Two or three player competitive matches
- Three distinct races



Friday, September 24, 2010

## StarCraft

## Traditional (constructive) map generation

- Place features on maps according to some heuristic
- e.g. fractals, growing islands, cellular automata
- Hard or impossible to optimize for gameplay properties
- Restrictions on possible content necessary in order to ensure valid maps

Friday, September 24, 2010

# Our approach:

- Direct/indirect map representations
- An ensemble of fitness functions
- Multiobjective evolution

Frida, September 24, 2010

# Our approach

- Define desirable traits of RTS maps
- Operationalize these traits as fitness functions
- Define a search space for maps
- Search for maps that satisfy the fitness functions as well as possible, using multiobjective evolution
- (visualize trade-offs as Pareto fronts)

Frida, September 24, 2010

## Desirable traits of an RTS map

- Playability
- Fairness
- Skill differentiation
- Interestingness

Frida, September 24, 2010

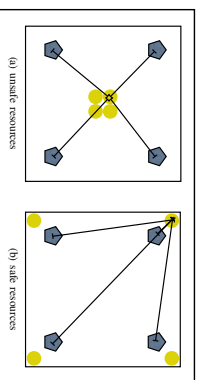
## Playability fitness functions

- Base space: minimum amount of space around bases
- Base distance: minimum distance between bases (via  $A^*$ )

Frida, September 24, 2010

## Fairness fitness functions

- Distance from base to closest resource
- Resource ownership
- Resource safety
- Resource fairness



Frida, September 24, 2010

## Skill differentiation fitness functions

(also contribute to interestingness)

- Choke points  
(narrowest width of shortest path)
- Path overlapping

Frida, September 24, 2010

# Dual map representation

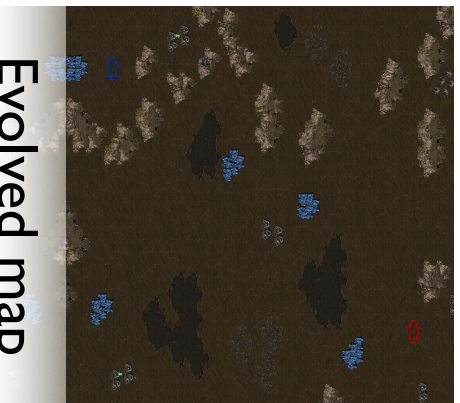
- Indirect representation: a vector of real numbers in  $\{0..1\}$
- Direct representation: a 64x64 grid corresponding to a StarCraft map, including impassable areas, bases, resource sites
- **Genotype to phenotype** mapping: before fitness calculation

Frida, September 24, 2010

# Genotype to phenotype

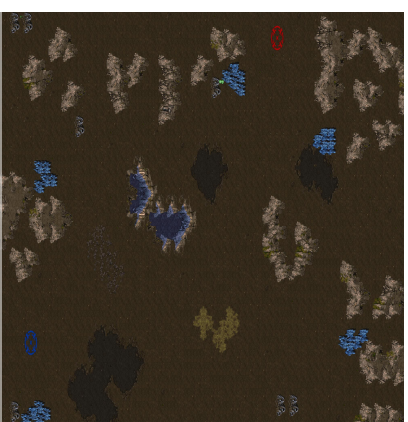
- Two or three bases, five mineral sources and five gas wells: ( $\phi$ ,  $\theta$ ) coordinates
- Rock formations represented indirectly using "turtle graphics". Each formation has:
  - ( $x$ ,  $y$ ) starting position
  - probability of turning left/right
  - probability of gaps ("fitting the pen")

Frida, September 24, 2010



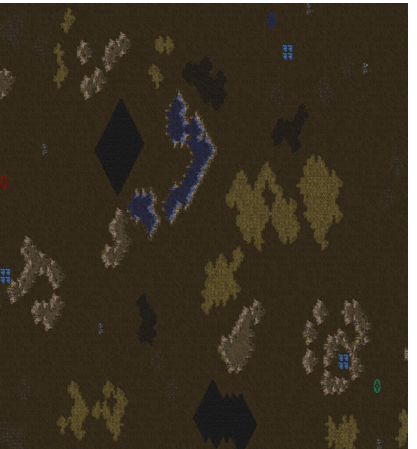
**Evolved map**  
Resource fairness vs. choke points

Frida, September 24, 2010



**Another evolved map**  
Resource fairness vs. choke points

Frida, September 24, 2010



**Three-player map**

Frida, September 24, 2010



**Another three-player map**

Frida, September 24, 2010

## Agent-based methods

- Use a number of “artificial agents” that construct the landscape by acting on it
- Agents of different types do different jobs
- Could be more controllable than diamond-square
- Could give rise to different types of landscapes

Friday, September 24, 2010

## Controlled Procedural Terrain Generation Using Software Agents

Jonathon Doran and Ian Parberry

Published in IEEE TCAIG, 2010

Friday, September 24, 2010

## D&P's five agent types

- Coastline agents
- Smoothing agents
- Beach agents
- Mountain agents
- River agents

Friday, September 24, 2010

## Rules for agents

- Each agent has a set number of “tokens” to spend on actions
- Each agent is allowed to see the current elevation around it, and allowed to modify it
- Agents don't interact *directly*

Friday, September 24, 2010

## In the beginning...

...there was a vast ocean.

Then came the first coastline agent.

Friday, September 24, 2010

## Coastline agents

- Multiply until they cover the whole coast - about 1000 necessary for this size maps
- Move out to position themselves right at the border of land and sea
- Generate a repulsor and an attractor point
- Score all neighbouring points according to distance to repulsor and attractor points
- Move to the best-scoring points, adding land as they go along

Friday, September 24, 2010

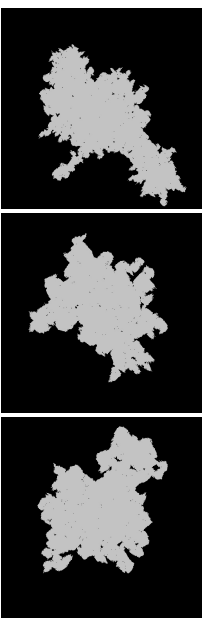
```

COASTLINE-GENERATE(agent)
1  if tokens(agent) ≥ limit
2  then
3      create 2 child agents
4      for each child
5      do
6          child ← a random seed point on parent's border
7          child ← 1/2 of the parent's tokens
8          child ← a random direction
9          COASTLINE-GENERATE(child)
10     else
11         for each token
12         do
13             point ← random border point
14             for each point p adjacent to point
15             do
16                 score p
17             fill in the point with the highest score

```

Fridy, September 24, 2010

## Coastline agents

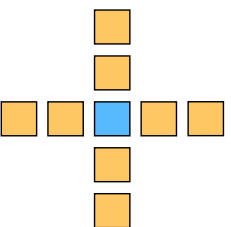


Varying action sizes

Fridy, September 24, 2010

## Smoothing agents

- Take random walks on the map
- Change the elevation of each visited point to (almost) the mean of its extended von Neumann neighbourhood



```

SMOOTH(starting-point)
1  location ← starting-point
2  for each token
3  do
4      heightlocation ← weighted average of neighborhood
5      location ← random neighboring point

```

Fridy, September 24, 2010

Fridy, September 24, 2010

## Beach agents

- Select random position along the coast, where coast is not too steep
- Flatten an area around this point (leaving small variations)
- Move randomly a short direction away from the coast, flattening the area

## Beach agents

```

BEACH-GENERATE(starting-point)
1  location ← starting-point
2  for each token
3  do
4      if heightlocation ≥ limit
5      then
6          location ← random shoreline point
7          flatten area around location
8          smooth area around location
9          inland ← random point a short distance inland from location
10         for i ← 0 to size(walk)
11         do
12             flatten area around inland
13             smooth area around inland
14             inland ← random neighboring point
15         location ← random neighboring point of location

```

Fridy, September 24, 2010

Fridy, September 24, 2010

# Beach agents



Varying beach width

Friday, September 24, 2010

# Mountain agents

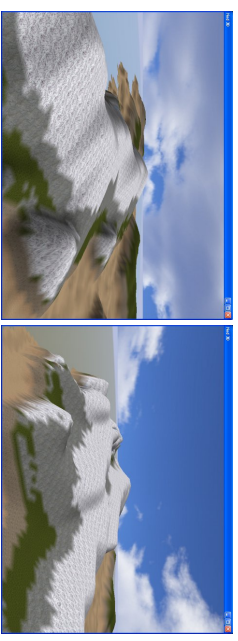
- Start at random positions and directions
- Move forward, continuously elevating a wedge, creating a ridge
- Turn randomly without 45 degrees from the initial course
- Periodically offshoot “foothills” perpendicular to movement direction

Friday, September 24, 2010

# Mountain agents

```
MOUNTAIN-GENERATE(starting_point)
1  location ← starting-point
2  direction ← random direction
3  for each token
4    do
5    elevate wedge perpendicular to direction
6    smooth area around location
7    location ← next point in direction
8    every n-th token
9    do
10   direction ← original-direction ± 45-degrees
```

Friday, September 24, 2010



Narrow versus wide features

Friday, September 24, 2010

# River agents

- Move from a random point on the coast towards a random point on a mountain ridge
- “Wiggle” along the path
- Stop when reaching too high altitudes
- Retrace the path down to the ocean, deepening a wedge along the path

Friday, September 24, 2010

# River agents

```
RIVER-GENERATE()
1  coast ← random point on coastline
2  mountain ← random point at base of a mountain
3  point ← coast
4  while point not at mountain
5    do
6    add point to path
7    point ← next point closer to mountain
8  while point not at coast
9    do
10   flatten wedge perpendicular to downhill direction
11   smooth area around point
12   point ← next point in path
```

Friday, September 24, 2010

# River agents



A dry river, and the outflow of three rivers

Friday, September 24, 2010

# In what order?

- Doran and Parberry suggest
- Coastline
- Landform
- Erosion
- But the “Implementation” suggests random order

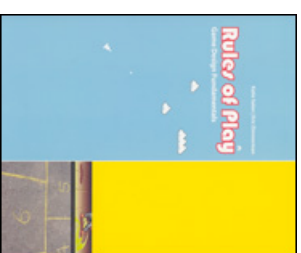
Friday, September 24, 2010

# Further questions

- Parameters... what parameters?
- What features of landscapes do we want to be able to specify?
- How can the human and the algorithm interact productively?

# Salen and Zimmermann define games:

*“A game is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome.”*



Friday, September 24, 2010

Friday, October 8, 2010

# Can we create game rules automatically?

- If so, which types of rules?
- For which types of games?
- How would we represent them?
- How would we judge how good a set of rules is?
- And why would we do this?

# Challenges

- How to represent game mechanics
- Representation should be complete
- Most games should make sense (?)
- High locality (?)
- Human-readable/editable (?)
- How to search the space
- How to evaluate the games

Friday, October 8, 2010

Friday, October 8, 2010

# Automatic generation of recombination games

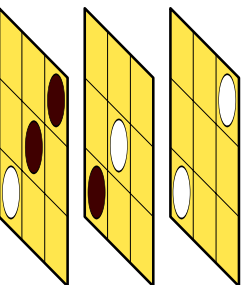
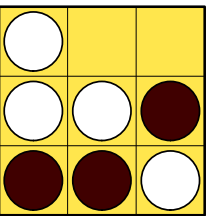
Cameron Browne  
 PhD Thesis, 2008  
 IEEE TCIAIG, 2010

- Finite: produce a well-defined outcome.
- Discrete: turn-based.
- Deterministic: chance plays no part.
- Perfect information: no hidden information.
- Two-player.

“Combinatorial games”

## The Ludi Game Description Language

- In practice limited to board games
- *Ludeme*: Fundamental units of independently transferable game information (“game meme”)
- (tiling square)
- (size 3 3)

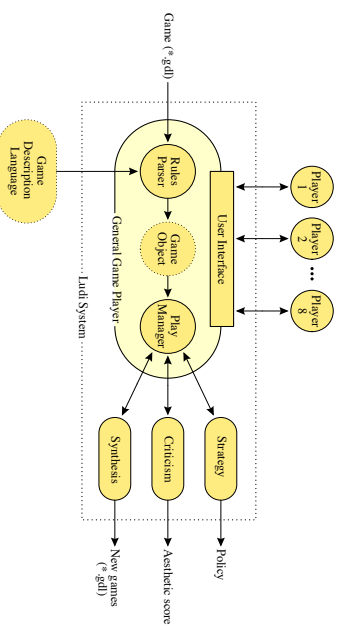


(size 3 3) vs (size 3 3 3)

## Tic-Tac-Toe

```
(game Tic-Tac-Toe
 (players White Black)
 (board
  (tiling square i-nbors)
  (size 3 3)
 )
 (end (All win (in-a-row 3))))
```

## The Ludi system



# Evaluating a game

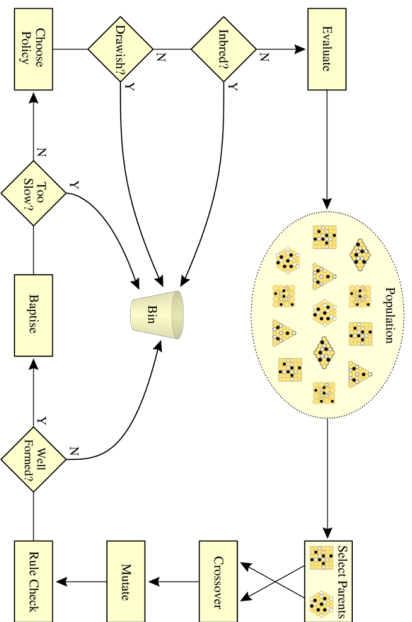
- Play the game (both player use same algorithm, with optimized board evaluation)
- Measure various *aesthetic criteria*: aspects of how the game is played, of the ruleset, and of the outcomes
- Combine the scores into a fitness value somehow

Friday, October 8, 2010

# Aesthetic criteria

- 16 Intrinsic: based on rules and equipment
- 11 Viability: based on game outcomes
- e.g. completion, duration
- 30 Quality: based on trends in play
- e.g. drama, uncertainty

Friday, October 8, 2010



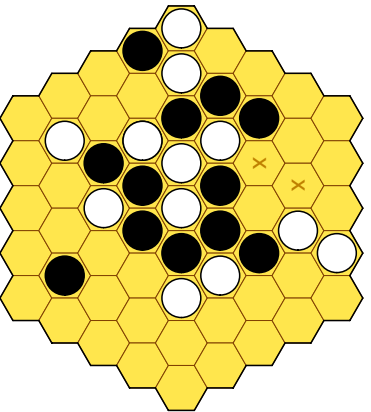
Friday, October 8, 2010

```
(game Yavalath
 (players White Black)
 (board (tiling hex) (shape hex) (size 5))
 (end
 (All win (in-a-row 4))
 (All lose (and (in-a-row 3) (not (in-a-row 4))))
 )
 )
```

Friday, October 8, 2010

# Yavalath

## Yavalath



Friday, October 8, 2010

## PCG and authorship

- How can we combine a human designer's authorial control and expressive ability with PCG capabilities?
- Dimensions of control
- Ease of use
- Multi-level editing / two-way flow of control

Friday, October 26, 2010



# Open issues

- Preserving manual changes
- Balance control and consistency
- Iterative modeling workflow and edit history (recreate previous actions?)

the death of level designer

seriously ?

---

Friday, October 28, 2010

## Where PCG will move?

- Traditional level design will adopt more PCG functions
- Games that do PCG will do much better in the marketplace
- PCG will continue to eat away at the bottom end
- Middleware developers will get on board with PCG