

# Game rules and PCG

Mark Nelson

# Encoding and generating game mechanics

- Encoding: a computer-readable representation of a game
- Generating: producing new games (or game variants)
- Game mechanics: a game's rules, or core logic

# Is generating rules PCG?

- Narrow view of PCG defines it as generating game *content*
  - Levels, items, terrain, ...
- Generating game rules becomes *automated game design*
  - Not things *in* the game, but the core of the game itself

# Challenges

- How do we represent game mechanics?
- Games *in general* can be almost anything
- Computer games *in general* can be almost any kind of software, almost as general as just code, an opaque representation

# Properties of a useful encoding

- Has elements that are common across games
- Usually within a genre, at least for now
- A large proportion of games in the encoding should be valid
- High locality/editability: make game variants by changing parts of an existing game

# More properties of a useful encoding

- Human editable?
  - Modular?
  - Breadth of coverage?
  - Analyzable?
- 
- Often tailored to choice of how the space will be searched, and how games will be chosen from it

# Two domains

- Board games
- 2d graphical-logic games

# Board games

- The domain many existing systems focus on
- Have a nice formal structure
- Discrete, turn-based
- There is an established culture of inventing variants



# Encoding board games

- What would an encoding need to be able to write down the rules for a game like chess?
- [Brainstorm an encoding for chess-like games.]

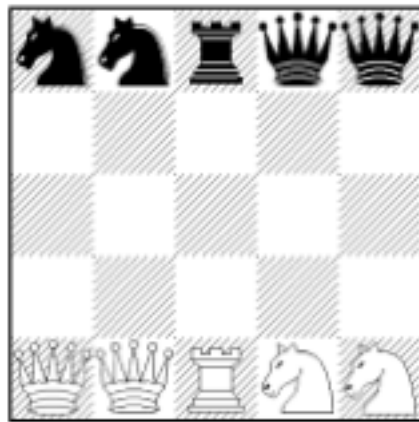
# METAGAME

- Early system to generate game rules (1992)
- Has a game grammar that defines a space of games more general than chess, but still roughly chess-like
- A specific game is a choice of rules from the grammar

# METAGAME's encoding

- Ontology: board, spaces, pieces with movement, capture, promotion rules
- Goals: stalemate, eradicate, or arrival
- Symmetric by construction
- "...a tradeoff between generality, where we prefer classes which can describe the widest variety of games, and structure, where we prefer a class where the individual games appear to have similar underlying structure"

# METAGAME – Turncoat chess



→ **Firefly**

→ **Termite**

→ **Slug**

- <http://www.barneypell.com/papers/metagame-scl-UCAM-CL-TR-277.pdf> (pp. 21-22)

# METAGAME's generator

- Probabilistic grammar: sample games from the grammar
- Works because:
  - The space of games is dense in playable games
  - Rough balance by symmetric construction
- Other features:
  - Cross-branch references to reuse logic
  - Some high-level parameters

# METAGAME's control knobs

- Rule complexity
- Decision complexity
- Indirectly: Search complexity
- Indirectly: Locality

# Optimizing for balance

- METAGAME aims for rough balance by construction
- Idea: Explicitly test for balance by self-play simulation
- Need a general game-playing engine that can play any game in the space. Then, a game is considered balanced if win rates in self-play are near 50/50.

# Optimizing for balance

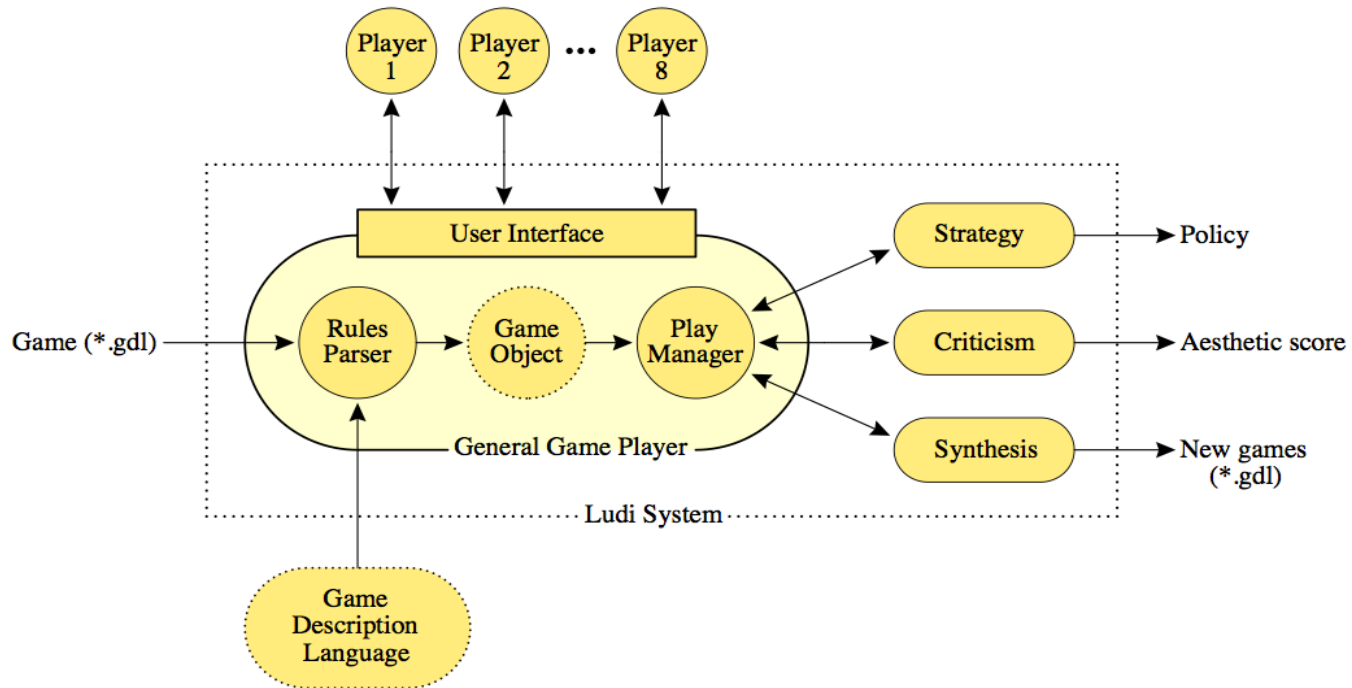
- Hom & Marks (2007)
- Significantly smaller space of games than METAGAME
- Self-play with the Zillions of Games engine
- Select for balanced games as the fitness function



# Evolutionary game design

- Ludi system goes a step further
- Criteria other than balance, e.g. aesthetics
- Search a larger space, via recombination of game elements
- Distinguish variants from new games (game novelty, game distance metrics)
- Wider range of boards

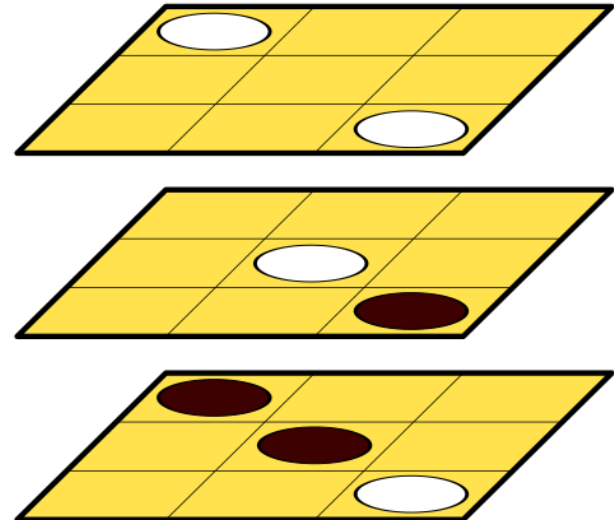
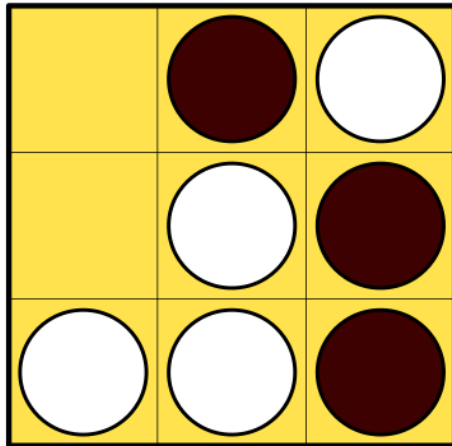
# The Ludi system



## Tic-tac-toe in Ludi

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

(size 3 3) and (size 3 3 3)



# Playing an arbitrary game

- Play the game using standard game-tree search
- Needs a state evaluation function!
- Twenty different advisors provide various assessments of a particular board state
- For each game, the linear combination of these advisors is evolutionarily optimized

# Evaluating a candidate game

- Self-play the game using the optimized evaluation
- Measure various *aesthetic criteria*: aspects of how the game is played, the ruleset, and the outcomes
- Combine the scores into a fitness value

# Aesthetic criteria

- 16 intrinsic: based on rules and equipment
- 11 viability: based on game outcomes
  - Completion, duration, etc.
- 30 quality: based on trends in play
  - Drama, uncertainty, etc.

# Combining the criteria

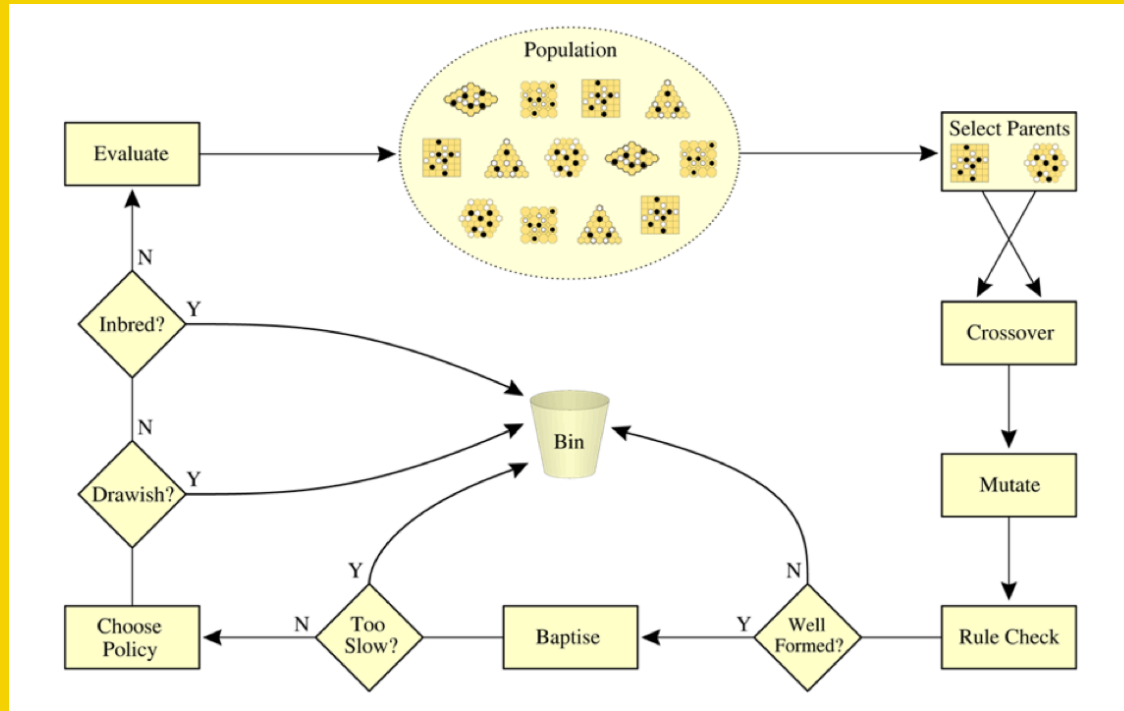
- The 57 criteria need to be weighted into a single fitness value
- First, 79 sample games were scored
- Then human players played and ranked some of these games
- The weight vector was fit using cross-entropy



# Searching the space of games

- A population was seeded with existing games
- New games evolved in a manner similar to genetic programming
  - Probabilistic selection
  - Subtree crossover

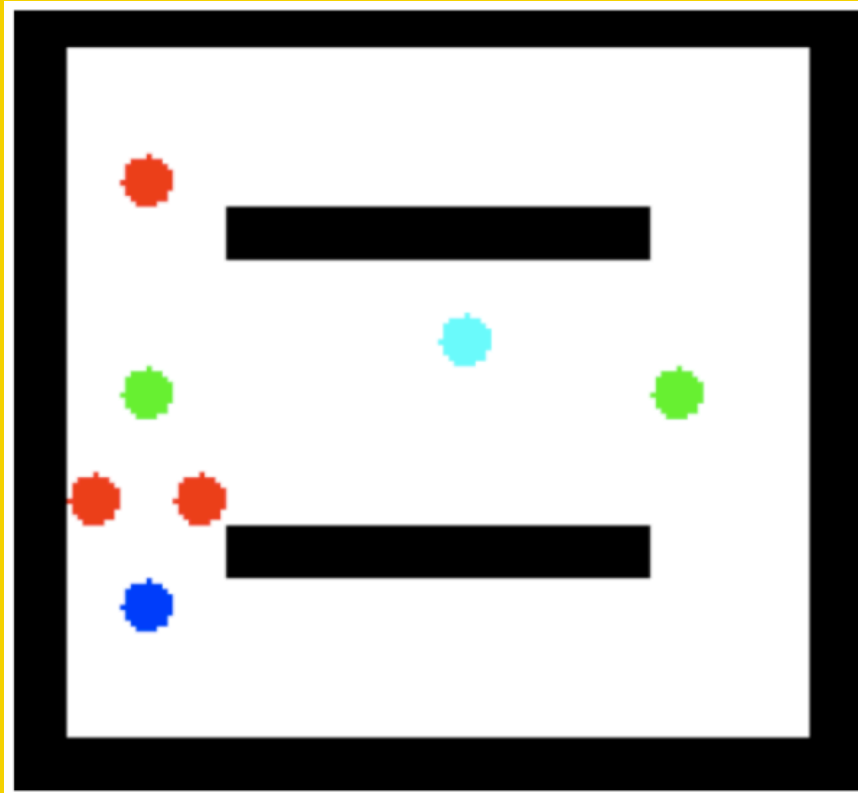
# Searching the space of games



# What about real-time games?

- Next cluster of experiments are on *graphical logic* games
- Games based on 2d movement of entities, simple physics, rules triggered by collision between entities

# Togelius & Schmidhuber



IT UNIVERSITY OF COPENHAGEN

# Game axioms

- The game takes place in a gridworld, fixed walls, one agent, red green and blue things
- Agent can move up, down, left, right
- Things might or might not move
- Effect of things colliding with each other or the agent depends on the color and the rules

# Rule representation

- Maximum time (0-100)
- Desired score (0-10)
- Number of items of each color (0-20)
- Movement logic for each color (1-5)
- Collision effects (4 x 4 table)
- Collision score effects (4 x 4 table)

# Movement logics

- 1. Still: does not move
- 2. Random short: change direction every time step
- 3. Random long: change direction every 1-10 time steps
- 4. Clockwise: turn right when facing wall
- 5. Counter-clockwise

# Collision effects

- 1. None: nothing happens
- 2. Die: if agent dies, game ends; if thing dies, thing disappears
- 3. Teleport: move to a random free position, at least two steps from agent
- Score effects: -1, 0, or +1



# Collision effects

- 1. None: nothing happens
- 2. Die: if agent dies, game ends; if thing dies, thing disappears
- 3. Teleport: move to a random free position, at least two steps from agent
- Score effects: -1, 0, or +1

# What is a *good* game of this kind?

- A fun one, of course!
- Hypothesis: fun == learning (Raph Koster)
- Hypothesis 2: fun =? learnability *by an algorithm*
- Games that are not learnable are no fun
- Games that can be won without having to learn much are no fun

# Variations Forever

- Goals:
- Meta-game based on exploration of new types of games
- Generator capable of generating a somewhat wider range of games
- Represent rules in logic, and generate games with desired properties

```
space_resolution(32,24) .
space_topology(spherical) .
background(grids; stars) .
active_agent(red; yellow; white; cyan) .
agent_movement(red,asteroids; white,asteroids;
               yellow,roguelike; cyan,pacman) .
agent_population(red,many; white,singleton;
               yellow,singleton; cyan,many) .
agent_collide_effect(red,white,kill;
                   cyan,yellow,kill) .
player_agent(white) .
obstacle_distribution(enclosure; random_walls;
                   random_blocks) .
obstacle_collide_effect(red,kill; white,kill) .
goal(kill all(red)) .
```

# Searching the rule space

- Answer-set programming: logical inference/solving
- Ask a question, and get a set of answers
- "Show me a game that implements mechanic X."

```
pushes(A,B) :-  
    on_collide(A,B,bounce),  
    on_collide(B,A,bounce).  
  
kills(A,B) :- on_collide(A,B,kill).  
  
indirectly_pushes(A,B) :- pushes(A,B).  
  
indirectly_pushes(A,C) :- pushes(A,B),  
    indirectly_pushes(B,C).  
  
winnable_via(indirect_push_kill(A,C)) :-  
    indirectly_pushes(A,B), kills(B,C).  
  
compute {  
    player_agent(A), goal(kill_all(B)),  
    winnable_via(indirect_push_kill(A,B)) }.  
}
```

# VGDL

- Videogame description language
- Aimed at a *simple*, human-readable/writable representation of graphical logic games
- Wider range of games
- Not yet automatically generated
- But specifications can be played and simulated in pyVGDL

# VGDL

```
WWWWWWWWWWWWWWWW
wA          w  w
w  w          w
w  w  w  +ww
www w1  wwwwww
w          w G w
w 1          ww
w      1      ww
WWWWWWWWWWWWWWWW
```



- <http://www.idsia.ch/~tom/publications/pyvgdl.pdf>



# VGDL principles

- Objects moving in a 2d game world
  - Both grid-based and continuous physics versions exist
- Rules are triggered by local interactions, which depend on object properties (extensible)
- Examples include frogger, missile command, etc.
- <https://github.com/schaul/py-vgdl/>