

# Artificial Intelligence and the Future of Games as an Expressive Medium

EXPRESSIVE INTELLIGENCE STUDIO

Michael Mateas  
Computer Science Department  
University of California, Santa Cruz



# Gaming has become a mass medium

- The old stereotype of videogames as the obsession of (nerdy) teenage boys is no longer true
- Game industry statistics
  - The average age of game players is 35
  - 40% of all game players are women
  - 33% of all game players are women over 18
  - 18% of game players are boys age 17 or younger
  - 26% of Americans over age 50 have played videogames (up from 9% in 1999)
- Games are regularly discussed in mainstream culture
  - Note the big spread on Spore in the *New York Times* yesterday

# Games and expressive content

- Games are not restricted to “mere” entertainment
  - Immersive, high-agency way to deeply experience content
  - Express content as rule systems
- Games are becoming a medium for
  - Education and training
  - Public policy debate
  - Editorial commentary
  - Documentary
- Question: how can AI be used to further the expressive potential of games?

# Two examples of expressive AI

- Interactive drama
  - Autonomous characters with rich personality and emotion
  - Dynamic stories
- Automated game design support
  - Support novice and expert game designers
  - Enable new, radically customized games

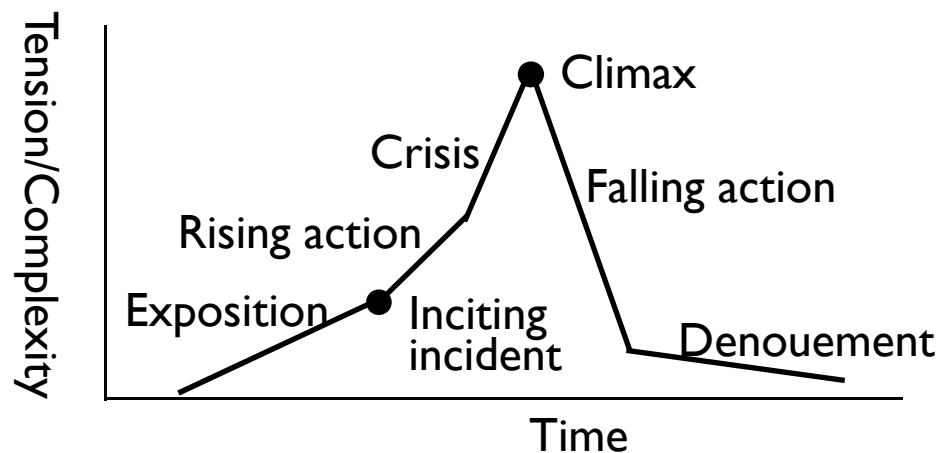
# Façade



[www.interactivestory.net](http://www.interactivestory.net)

# Interactive drama = character + story

## Plot structure (global constraints)

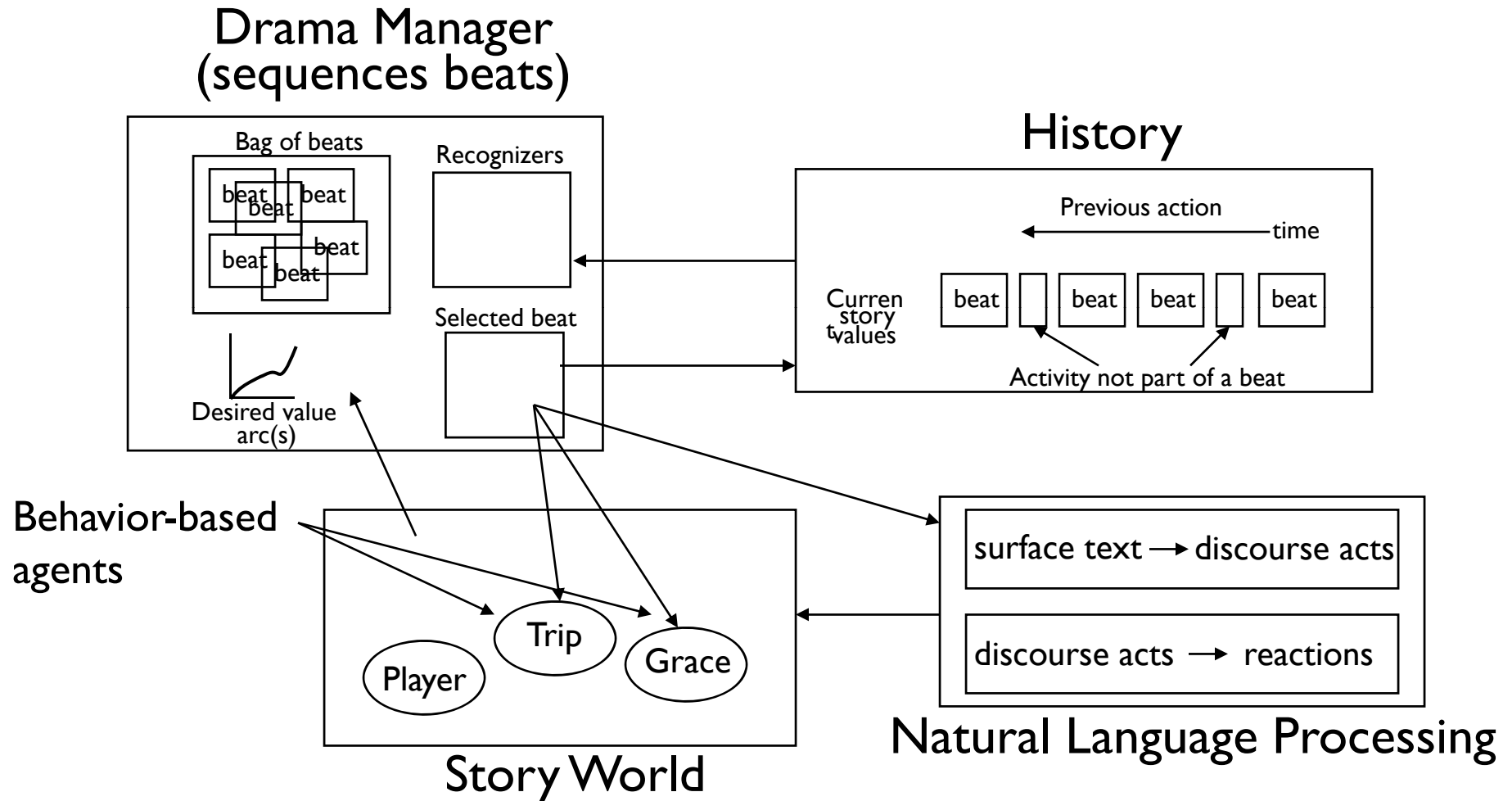


## Characters (consistency, inner life)

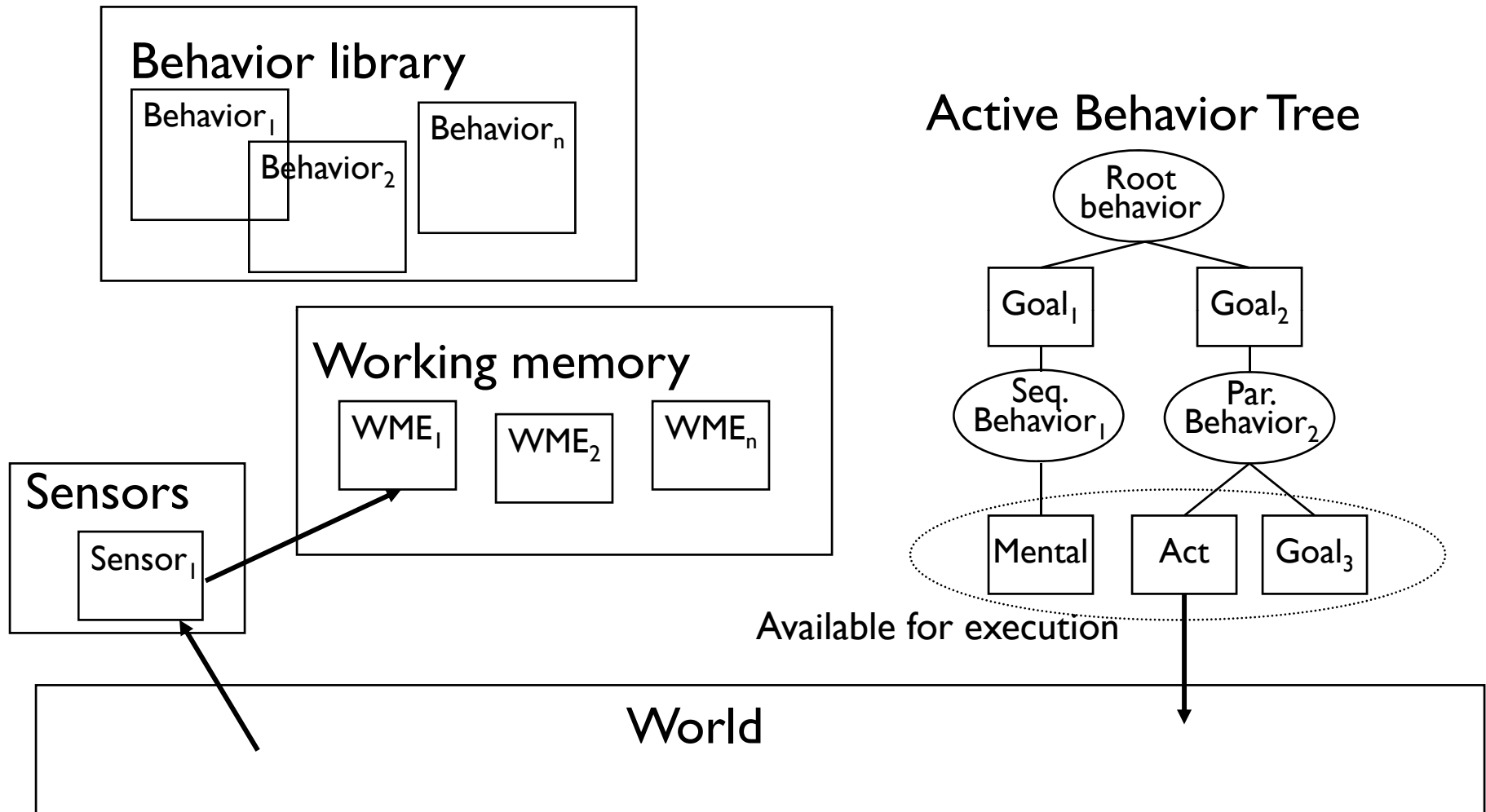


- Personality
- Emotion
- Self motivation
- Change
- Social relationships
- Consistency
- Illusion of life

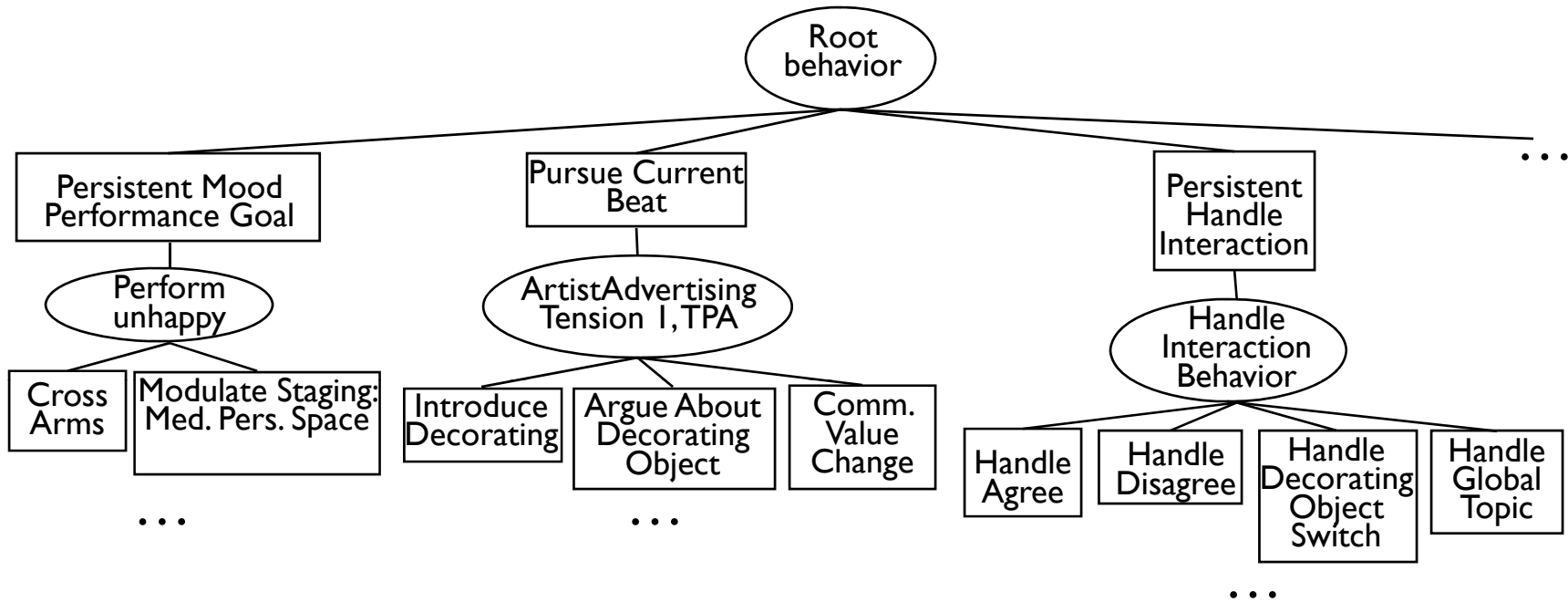
# Façade architecture



# An ABL Agent



# Façade character snapshot



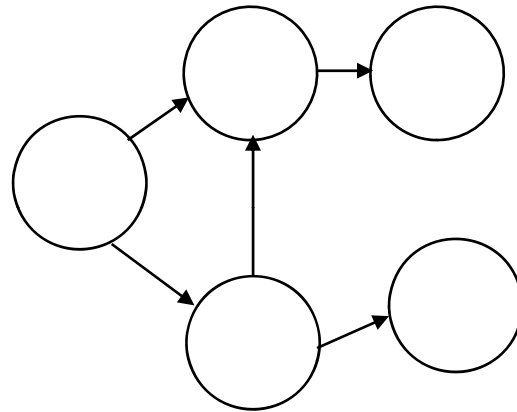
**Beat state** dialog cue = head nod, beat gist not occurred, staging = conv. distance, dec. object = couch, ...

**Emotion state** surface emotion = happy, deep emotion = unhappy, transient emotion = surprised, ...

**In a typical snapshot, there are dozens of parallel goals, 200+ decision cycles a second, hundreds of working memory elements modulating behavior (e.g. multiple emotions, beat state, detailed staging, episodic state, mixin progression state, etc.)**

# The problem of branching scenarios

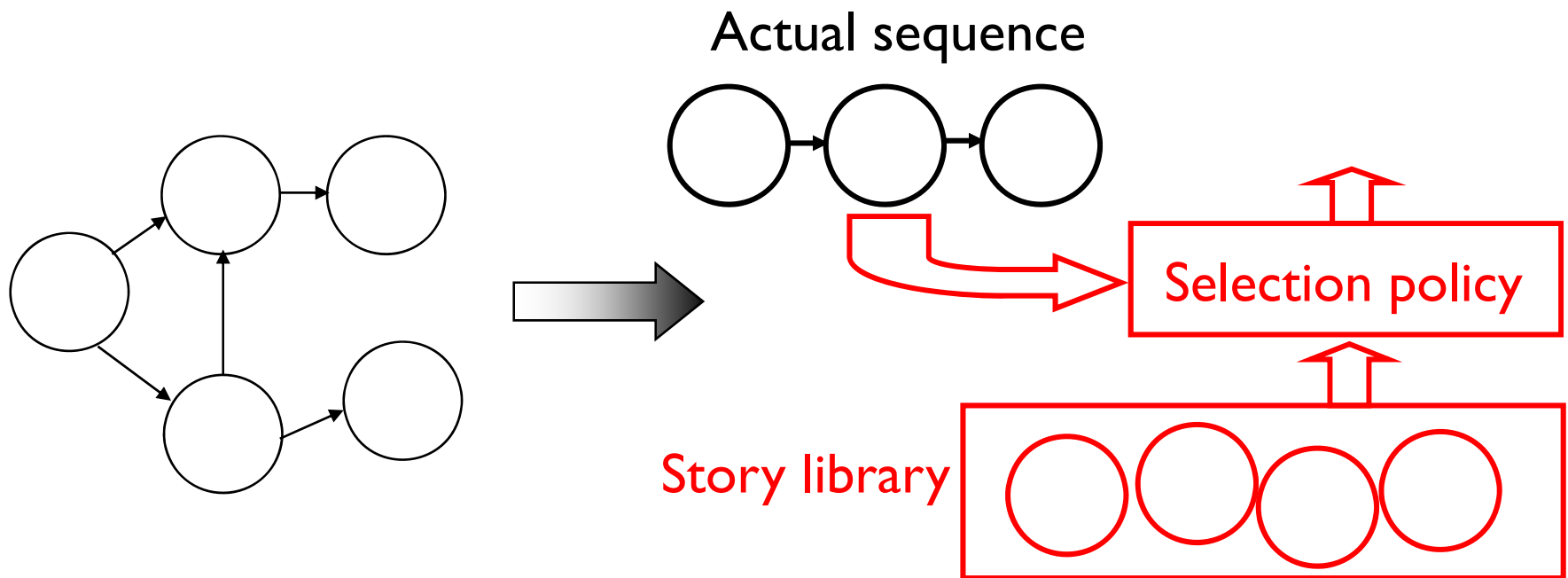
## The Enemy



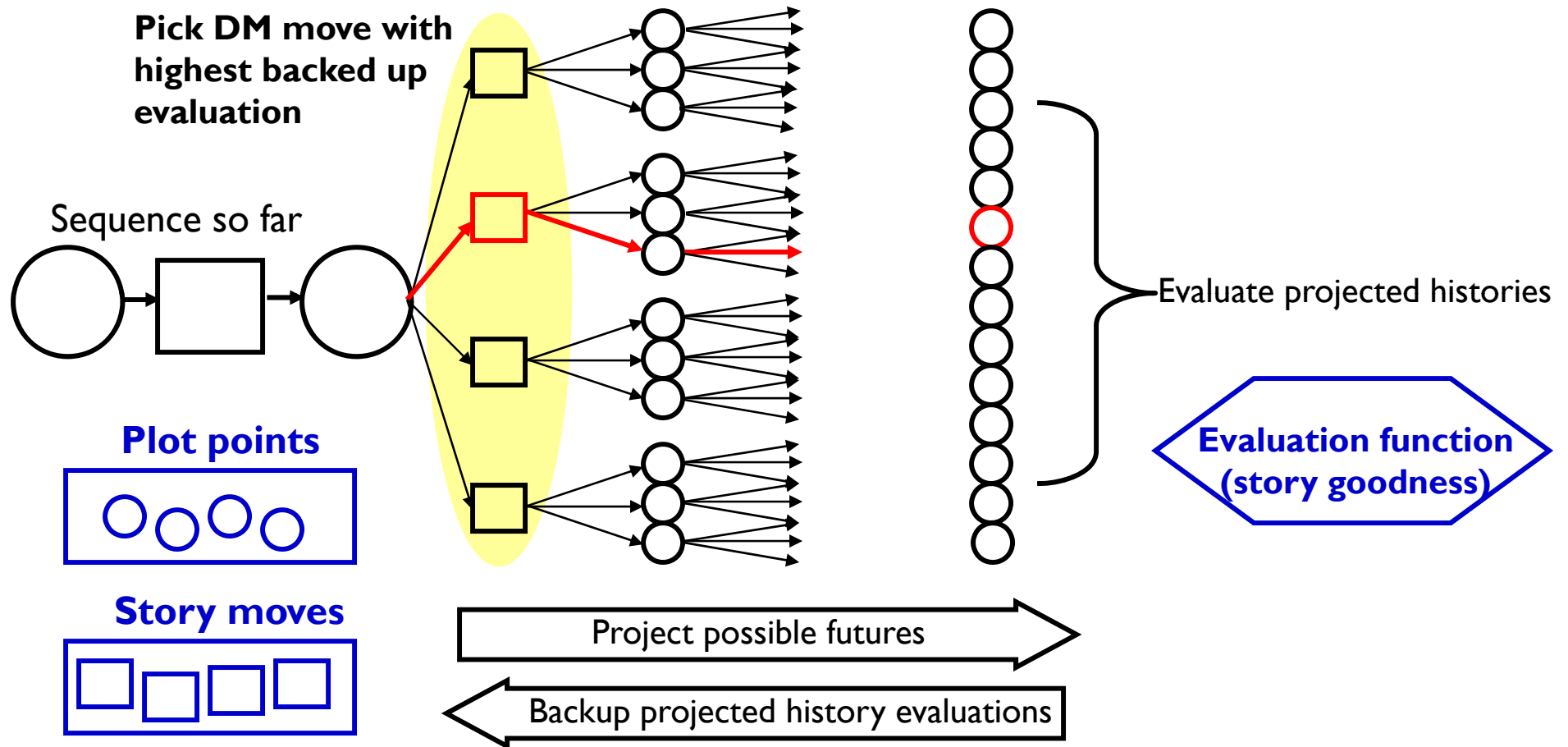
- Story structure has strong authorial direction **but**
  - All interaction paths must be pre-coded by author
  - Can only make very small stories
  - Bits of story can't be incrementally added

# Drama management

- Policy for “story piece” selection
- An alternative to explicitly coded links

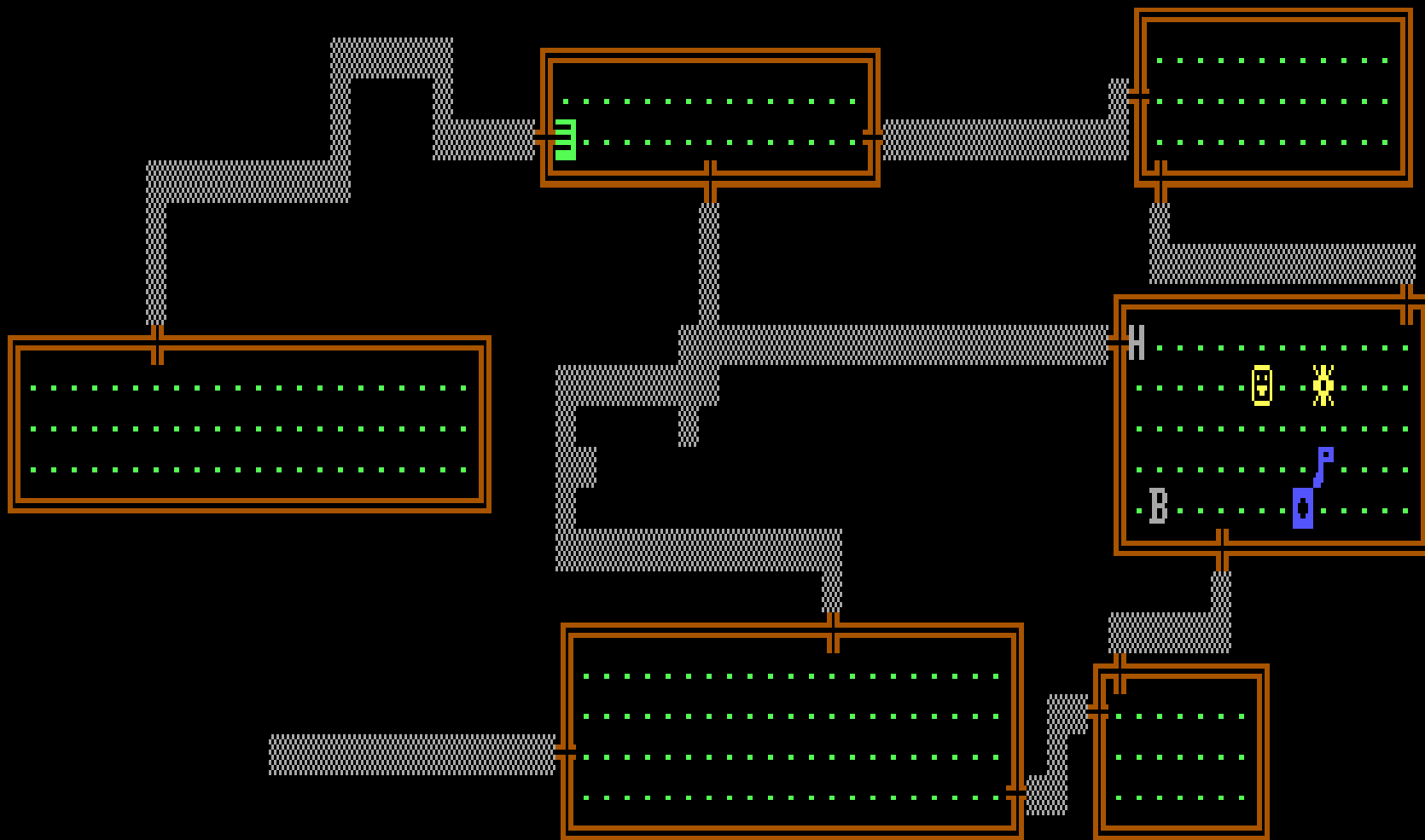


# Optimization-based drama management



# What is automated support for game design?

- An automated game designer reasons about game mechanics, representations, and player input devices to answer questions about or automatically generate a game
- An automated designer constitutes an operational theory of game design
- An automated designer can support human game designers, or move design to the meta-level



Level:1

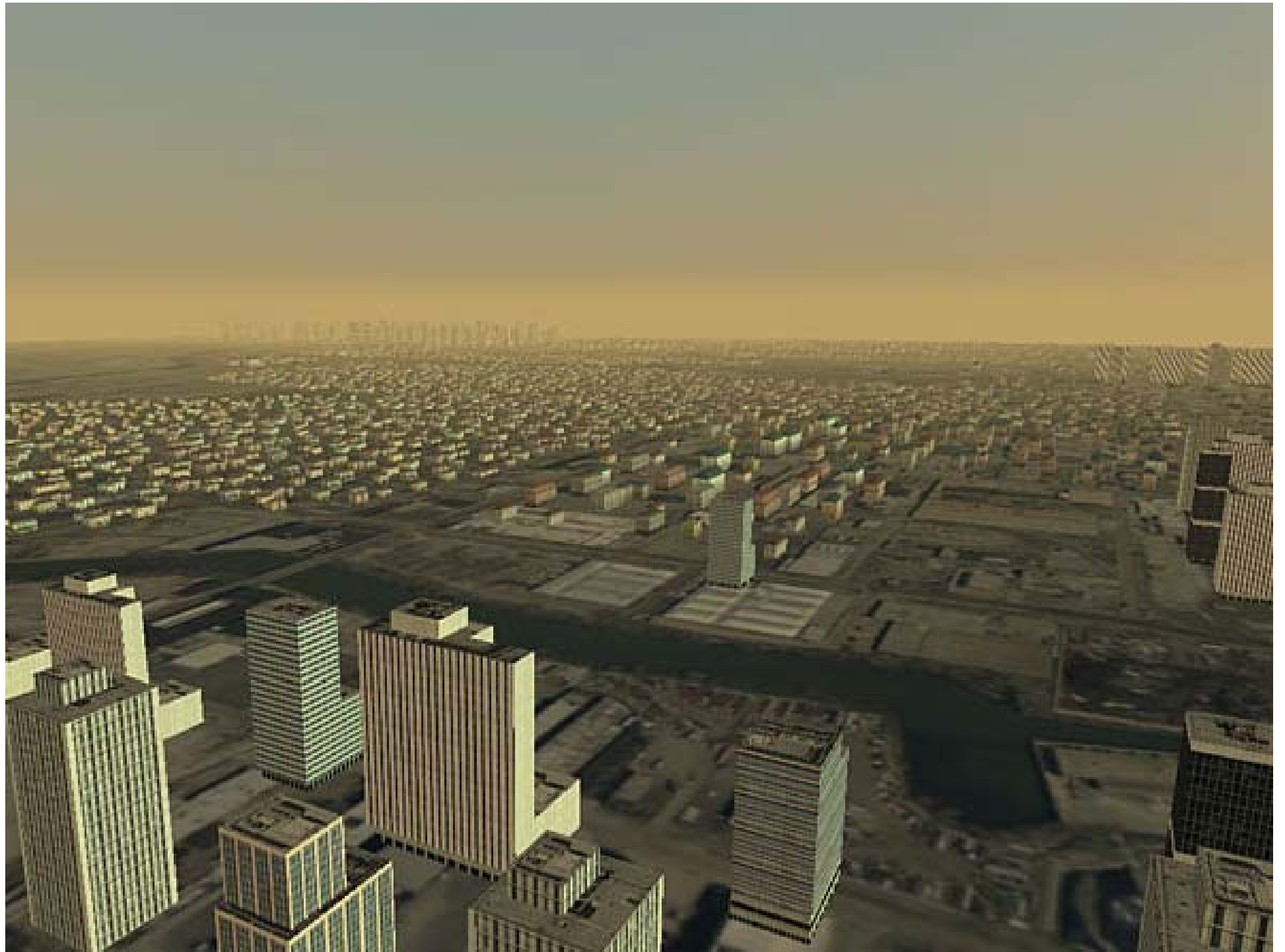
Hits:12(12)

Str:16(16)

Gold:75

Armor:5

Exp:1/5



# Help novice designers

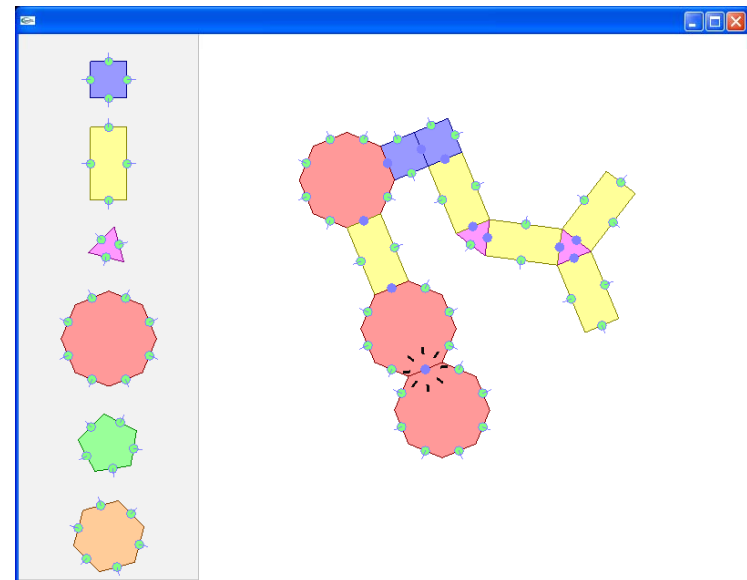
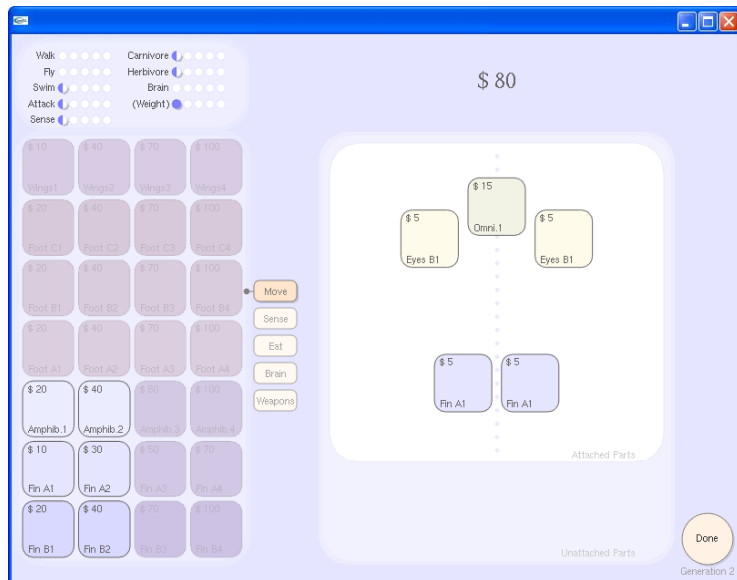
- Many people want to build small expressive or informative games



- There are assistants for game implementation
  - GameMaker, Alice, etc.
- None for game design

# Make prototypes more informative

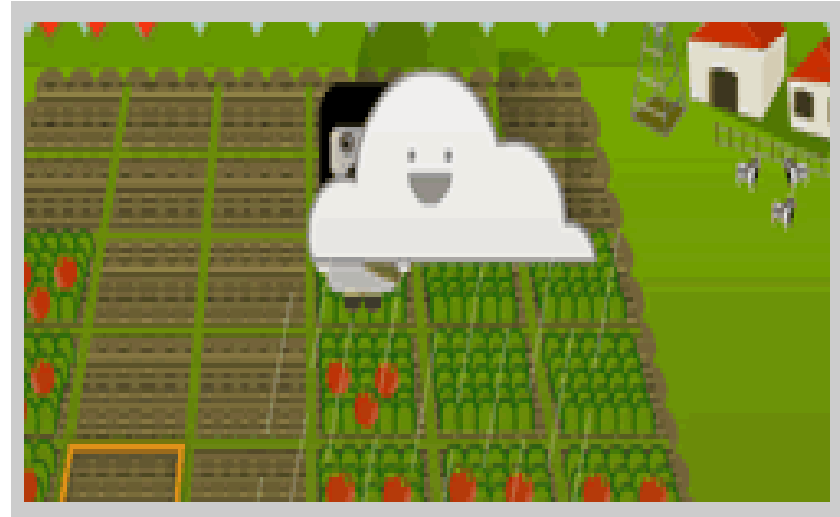
- Rapid prototyping is becoming a standard practice for expert designers



- While prototypes support experimentation with raw mechanics, they don't help designers think about implications of mechanics
  - Build, play, and hope for insight

# Enable new genres

- Some game genres require rapid deployment in response to specific events



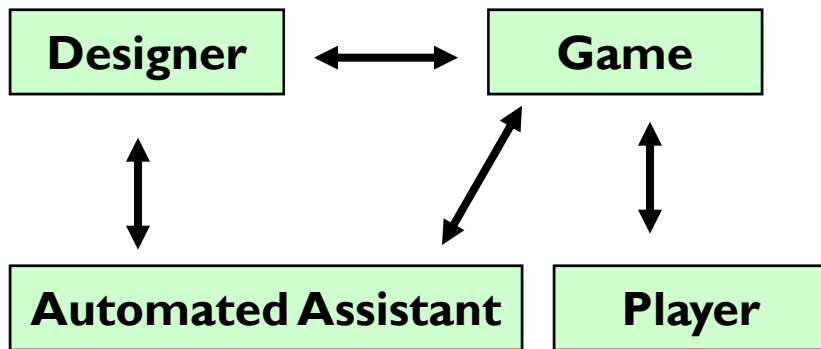
- Need automated design to enable these genres
  - Newsgames
  - Game-a-day customized games
  - Educational games customized to specific learners

# Understand game design

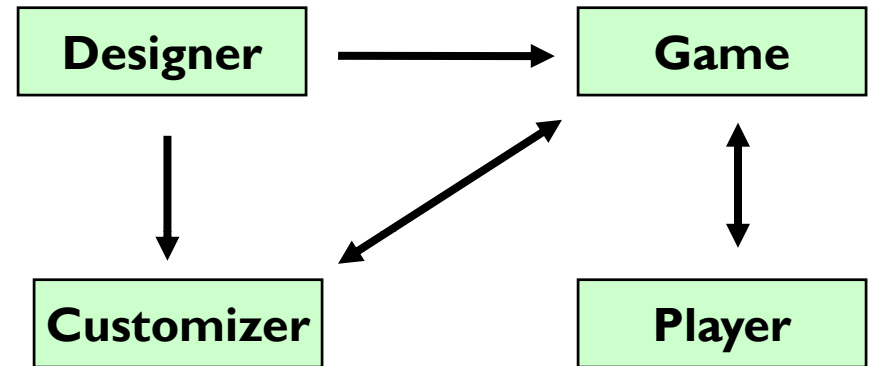
- Support formal game studies
- Can serve same role in understanding games as story understanding and generation models serve in understanding narrative
- There have been repeated calls for a “language of game design” (Doug Church’s formal abstract design tools, etc.)
- The knowledge representation and engineering required for automated game design support is precisely a formal theory of game design

# Automated design roles

## Design-time Assistance



## Per-Player Customization



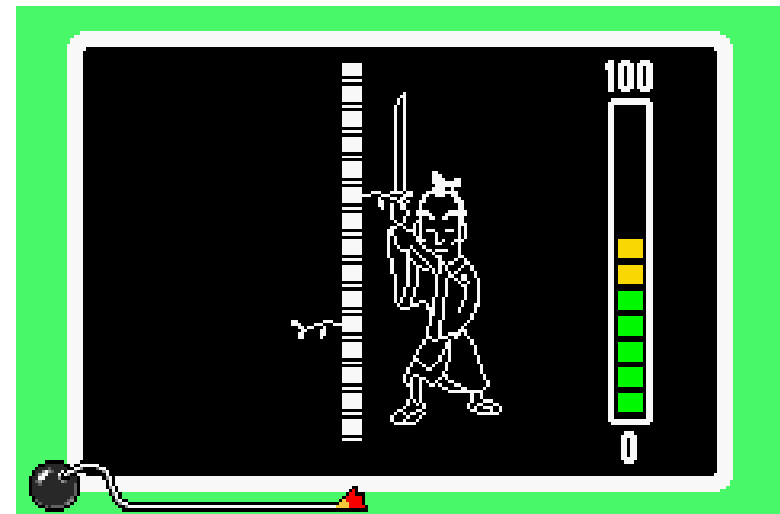
## Automated Game Design



# Factoring the game-design process

- Abstract game mechanics
  - State and state evolution
- Concrete game representation
  - Audiovisual representation of the abstract mechanics
- Thematic content
  - Real-world references of the game
- Control mappings
  - How a player interacts with the game

# Factoring examples



# Factoring examples

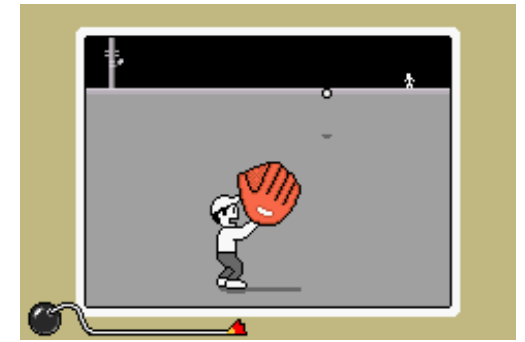
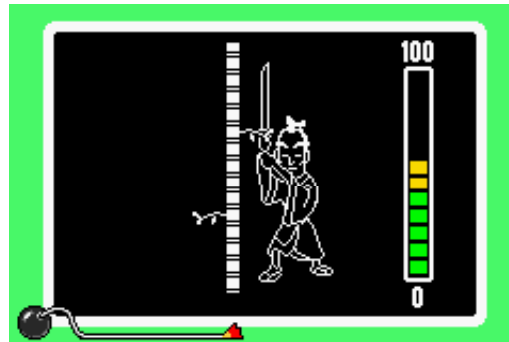


# Thematic reasoning prototype

- Automatically design *WarioWare*-style microgames with sensible real-world references
- Focus on thematic design
  - Little work in this area
  - Important for many serious games in which the game's real-world references are key to its meaning
- Existing systems focus on:
  - Abstract dynamics
    - METAGAME's chess-like games (Pell 1992)
    - Balanced board games (Hom & Marks 2007)
  - “Compiling” abstract dynamics to an interface
    - EGGG (Orwant 2000)
  - Generating new representations
    - Level generation, procedural content generation, etc.

# WarioWare

- Nintendo game, presenting a series of very simple microgames
- Each game lasts a few seconds, and has one mechanic with simple controls



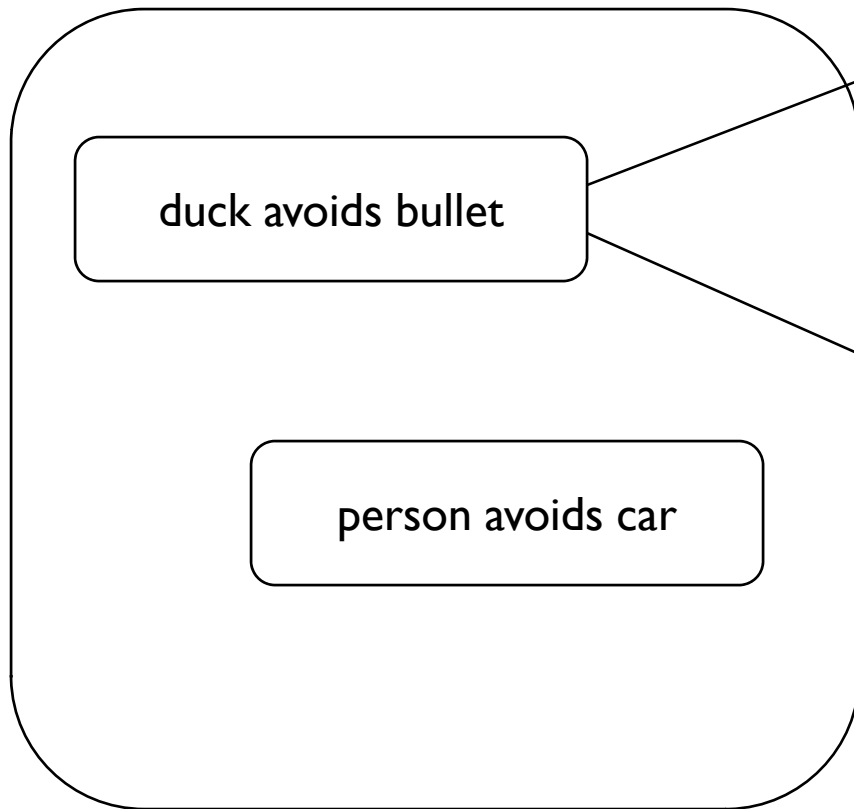
- Why *WarioWare*?
  - Rich theme space
  - Simple mechanics space lets us focus on themes
  - Game designers find it interesting (Gingold 2005)

# Decomposing *WarioWare*

- Consider games on two levels: abstract and concrete
- The *abstract* game is thematic, with high-level dynamics
  - Bundles thematic elements and high-level game dynamics
  - Person avoiding a car
  - Duck avoiding a bullet
  - Mechanic pumping up a tire
- The *concrete* game is a templated bundle of J2ME code
  - Executable code that bundles dynamics, representation, and control mapping
  - First-person crosshairs, player wins if she shoots [Sprite-A]
  - 2d side view, player plays [Sprite-A] and wins if she avoids [Sprite-B] for 5 seconds
- Generation problem: Generate abstract games that “make sense”, and plug them into concrete games

# Avoid abstract game

Attacker (capable of attacking)  
attacks  
Avoider (capable of avoiding)



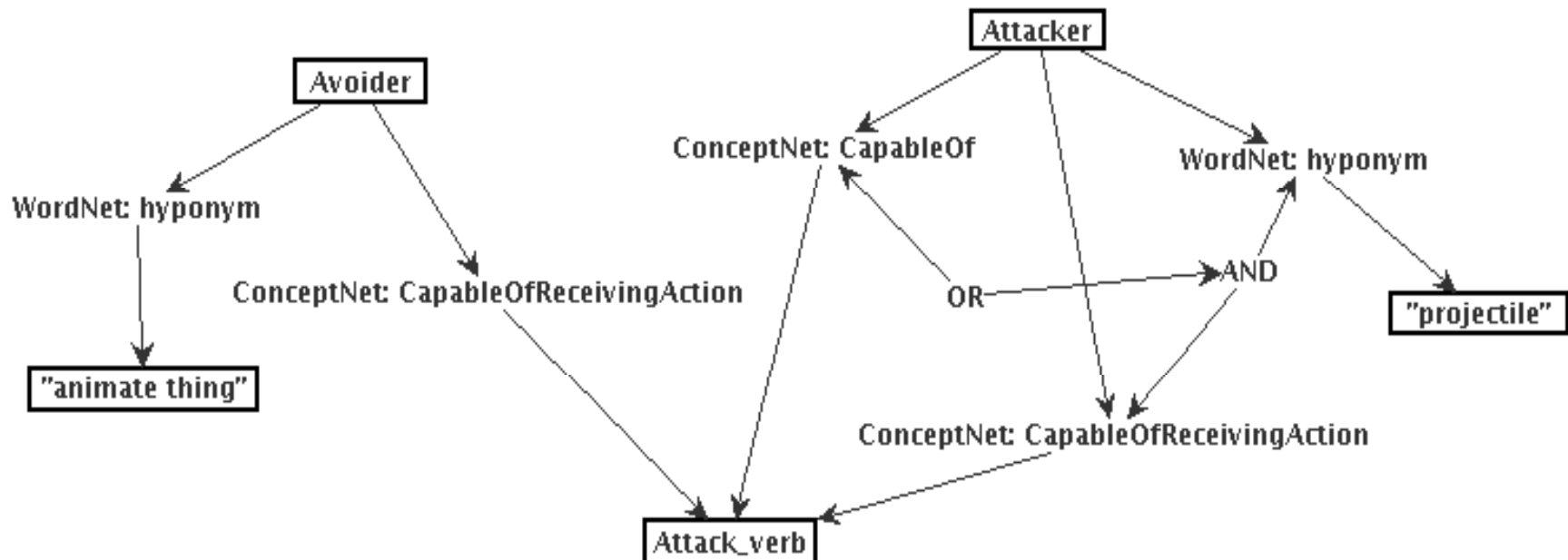
Shooter can be chosen  
if attacker is projectile

Dodger can be chosen  
for all attacker role fillers



# Constraint-based abstract game design

- Define spaces of abstract games:
  - Game nouns and verbs
  - Constraints on how they must relate
  - Defined in terms of common-sense knowledge bases



# Reasoning about mechanics

- The thematic reasoning system procedurally encodes mechanics
  - Therefore, difficult to reason about mechanics
- We're using the event calculus to reason about mechanics and state representation
- Event calculus work driven by two applications
  - Game design assistant for expert designers
  - Automated design discovery

# Introduction to event calculus

- A temporal logic for reasoning about system dynamics
  - Recently championed by Eric Mueller for common-sense reasoning
- Based on *fluents*, *events* and *timepoints*, related by four basic predicates
  - HoldsAt(*fluent*, *time*)
  - Happens(*event*, *time*)
  - Initiates(*event*, *fluent*, *time*)
  - Terminates(*event*, *fluent*, *time*)
- The meaning of these predicates is defined by the event calculus axioms
  - E.g.  
; If an initiating event happens, the fluent it initiates is henceforth true and inertial.  
(Happens(*event*,*time*) & Initiates(*event*,*fluent*,*time*)) →  
(HoldsAt(*fluent*,*time*+1) & !ReleasedAt(*fluent*,*time*+1))
- The situation calculus is another standard logic for reasoning about dynamics.
- We use the event calculus because
  - We find it more natural for expressing game mechanics
  - We don't need the branching time model of the situation calculus

# Applying event calculus to games

- Fluents encode game state
- Events encode player input (button press) and game-state initiated events (collision, player wins)
- Game mechanics are state-evolution rules, i.e. the changes in game state caused by events, and the events triggered by game state
  - Colliding with an enemy (event) causes player to lose health (fluent value change)
  - Reaching 100 points (state trigger) causes player to win game (event)

# Simple tile world

- Sprites occupying the same x, y tile location collide
  - $((\text{sprite1} \neq \text{sprite2}) \wedge \text{HoldsAt}(\text{At}(\text{sprite1}, x, y), \text{time}) \wedge \text{HoldsAt}(\text{At}(\text{sprite2}, x, y), \text{time})) \rightarrow \text{Happens}(\text{Collide}(\text{sprite1}, \text{sprite2}), \text{time})$
- Inventory is empty at start of game, the *gain* and *lose* events add and remove objects from inventory, and, to use an inventory item, it has to be in your inventory
  - $\neg \text{HoldsAt}(\text{InInventory}(\text{item}), 0)$
  - $\text{Initiates}(\text{Gain}(\text{item}), \text{InInventory}(\text{item}), \text{time})$
  - $\text{Terminates}(\text{Lose}(\text{item}), \text{InInventory}(\text{item}), \text{time})$
  - $\text{Happens}(\text{UseOn}(\text{item}, \text{object}), \text{time}) \rightarrow \text{HoldsAt}(\text{InInventory}(\text{item}), \text{time})$
- Collision with an item you can pick up causes a gain event
  - $\text{Happens}(\text{Collide}(\text{sprite1}, \text{sprite2}), \text{time}) \wedge \text{HoldsAt}(\text{HasSprite}(\text{sprite1}, \text{Player}), \text{time}) \wedge \text{HoldsAt}(\text{HasSprite}(\text{sprite2}, \text{item}), \text{time}) \wedge \text{HoldsAt}(\text{PickupableItem}(\text{item}), \text{time}) \rightarrow \text{Happens}(\text{Gain}(\text{item}), \text{time})$



# Tile-world example

Exit		Wall	
	Door (locked)		Key
Wall			
			Player

Play

Save state

## Saved game states

Initial

Player wins with key

Player wins without key

Player on top of wall

Plan between states

Edit state

## Mechanics

cardinal movement

pick up objects

diagonal movement

impassable walls

time limit

## Watched state

At(Player, \_, \_): (4,4)

InInventory(Player, \_): [none]

# Why use logic?

- Deduction = forward simulation
  - We can forward simulate the game just as you would with procedural code
- Abduction = planning
  - We can ask questions about reachable game states, about ways of overcoming challenges, etc.
- Induction = player modeling
  - We can take concrete player traces (either observed or abduced) and build player models

# Higher-level design languages

- Event calculus representations are an “assembly language” of game design
  - Prone to subtle logic bugs, e.g. implication for rules
- We are developing higher-level languages for specific sub-domains of game design that compile to event calculus
  - Language will be part of design assistant interface
  - Will serve as highly elaboration tolerant syntax for discovery system

# Interviewing game designers

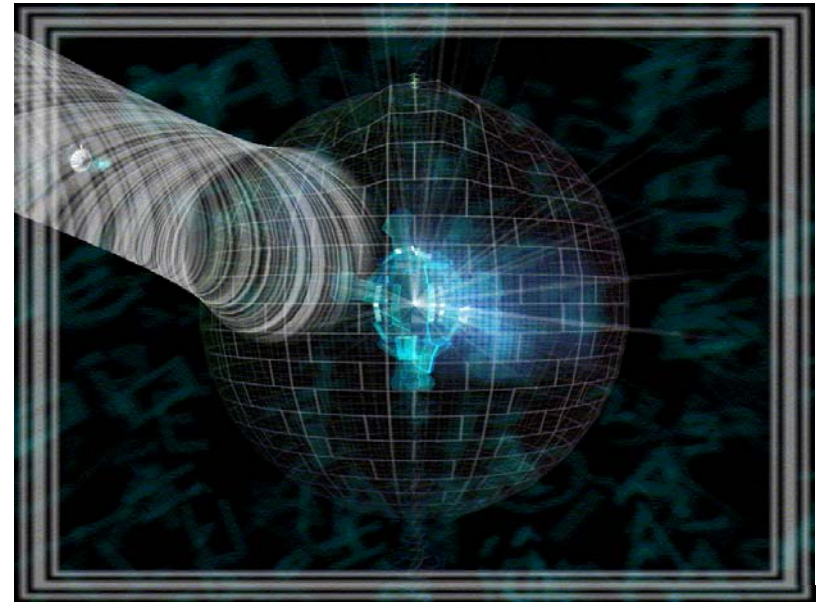
- Interviewing expert designers for requirements analysis for both the AI back-end and the design interface
- Guided ethnographic interviews
  - Process maps to understand prototyping process
  - Focusing prototypes of design assistant
  - Guided discussion of game designer prototypes
- Understanding the edges of which design questions we can and can't help with

# Expressive AI

- My group at UC Santa Cruz is focused on AI as an enabler for new expressive media
- Expressive AI is a fundamentally interdisciplinary form of AI research with strong social impact
- Games are poised to become a dominant medium of the 21<sup>st</sup> century and thus is an important target for expressive AI work

# CS: Computer Game Design

- Video games have quickly become one of the biggest mainstream drivers of hardware and software advances
  - The cutting edge of real-time physics simulation, animation, distributed systems, ever more sophisticated AI...
- A technically focused degree program
  - Strong core of computer science
  - Additional depth courses in game design and artistic aspects
  - Creating future leaders who can *radically* innovate
- Strong numbers
  - 99 students starting program Fall 2007
  - 118 students starting Fall 2008
  - One of the largest programs in the school of engineering



**ping.exe**

Cyberspace exploration game by  
UCSC student Nicholas Kent