

# Machine Learning for Computer Games

John E. Laird & Michael van Lent

Game Developers Conference

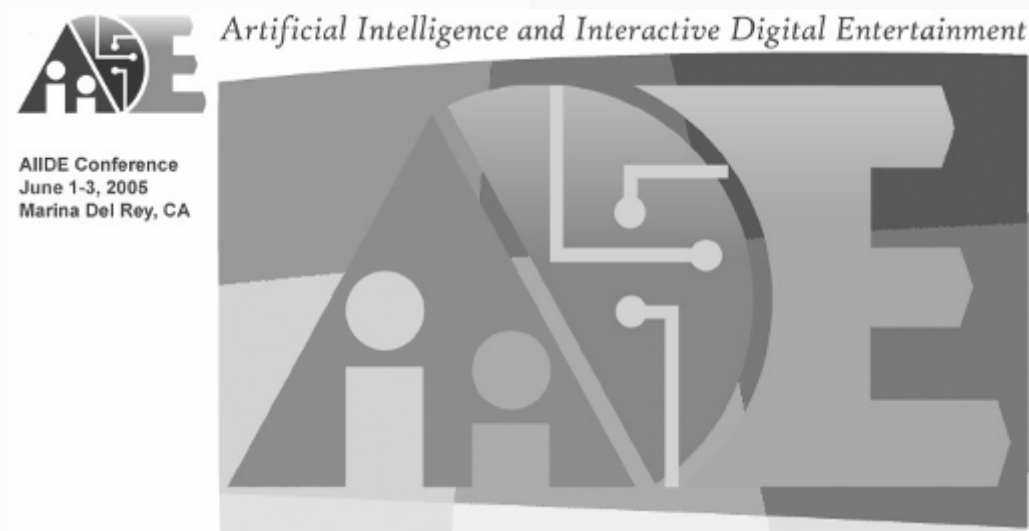
March 10, 2005

<http://ai.eecs.umich.edu/soar/gdc2005>

# Advertisement

## Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)

- June 1-3, Marina Del Rey, CA
- Invited Speakers:
  - Doug Church
  - Chris Crawford
  - Damian Isla (Halo)
  - W. Bingham Gordon
  - Craig Reynolds
  - Jonathan Schaeffer
  - Will Wright
- [www.aiide.org](http://www.aiide.org)



# Who are We?

- **John Laird** (laird@umich.edu)
  - Professor, University of Michigan, since 1986
  - Ph.D., Carnegie Mellon University, 1983
  - Teaching: Game Design and Development for seven years
  - Research: Human-level AI, Cognitive Architecture, Machine Learning
  - Applications: Military Simulations and Computer Games
- **Michael van Lent** (vanlent@ict.usc.edu)
  - Project Leader, Institute for Creative Technology, University of Southern California
  - Ph.D., University of Michigan, 2000
  - Research: Combining AI for commercial game techniques for immersive training simulations.
  - Research Scientist on Full Spectrum Command & Full Spectrum Warrior



# Goals for Tutorial

1. What is machine learning?
  - What are the main concepts underlying machine learning?
  - What are the main approaches to machine learning?
  - What are the main issues in using machine learning?
2. When should it be used in games?
  - How can it improve a game?
  - Examples of possible applications of ML to games
  - When shouldn't ML be used?
3. How do you use it in games?
  - Important ML techniques that might be useful in computer games.
  - Examples of machine learning used in actual games.

# What this is not...

- Not about using learning for board & card games
  - Chess, backgammon, checkers, Othello, poker, blackjack, bridge, hearts, ...
    - Usually assumes small set of moves, perfect information, ...
  - But a good place to look to learn ML techniques
- Not a cookbook of how to apply ML to your game
  - No C++ code

# Tutorial Overview

- I. Introduction to learning and games [.75 hour] {JEL}
- II. Overview of machine learning field [.75 hour] {MvL}
- III. Analysis of specific learning mechanisms [3 hours total]
  - Decision Trees [.5 hour] {MvL}
  - Neural Networks [.5 hour] {JEL}
  - Genetic Algorithms [.5 hour] {MvL}
  - Bayesian Networks [.5 hour] {MvL}
  - Reinforcement Learning [1 hour] {JEL}
- IV. Advanced Techniques [1 hour]
  - Episodic Memory [.3 hour] {JEL}
  - Behavior capture [.3 hour] {MvL}
  - Player modeling [.3 hour] {JEL}
- V. Questions and Discussion [.5 hour] {MvL & JEL}

# Part I

# Introduction

John Laird

# What is learning?

- Learning:
  - “The act, process, or experience of gaining knowledge or skill.”
- Our general definition:
  - The capture and transformation of information into a usable form to improve performance.
- Possible definitions for games
  - The appearance of improvement in game AI performance through experience.
  - Games that get better the longer you play them
  - Games that adjust their tactics and strategy to the player
  - Games that let you train your own characters
  - Really cool games

# Why Learning for Games?

- Improved Game Play
- Cheaper AI development
  - Avoid programming behaviors by hand
- Reduce Runtime Computation
  - Replace repeated planning with cached knowledge
- Marketing Hype

# Improved Game Play I

- Better AI behavior:
  - More variable
  - More believable
  - More challenging
  - More robust
- More personalized experience & more replayability
  - AI develops as human develops
  - AI learns model of player and counters player strategies
- Dynamic Difficulty Adjustment
  - Learns properties of player and dynamically changes game to maximize enjoyment

# Improved Game Play II

- New types of game play
  - Training characters
    - Black & White, Lionhead Studios
  - Create a model of you to compete against others
    - Forza Motorsport, Microsoft Game Studios for XBOX



# Marketing Hype

- *Not only does it learn from its own mistakes, it also learns from yours! You might be able to out think it the first time, but will you out think it the second, third and fourth?*
- *“Check out the revolutionary A.I. Drivatar™ technology: Train your own A.I. "Drivatars" to use the same racing techniques you do, so they can race for you in competitions or train new drivers on your team. Drivatar technology is the foundation of the human-like A.I. in Forza Motorsport.”*
- *“Your creature learns from you the entire time. From the way you treat your people to the way you act toward your creature, it remembers everything you do and its future personality will be based on your actions.”* Preview of Black and White

# Why Not Learning for Games?

- Worse Game Play
- More expensive AI Development
- Increased Runtime Computation
- Marketing Hype Backfire

# Worse Game Play: Less Control

- Behavior isn't directly controlled by game designer
- *Difficult to validate & predict all future behaviors*
- AI can get stuck in a rut from learning
- Learning can take a *long* time to visibly change behavior
- If AI learns from a stupid player, get stupid behavior
  - “Imagine a track filled with drivers as bad as you are, barreling into corners way too hot, and trading paint at every opportunity possible; sounds fun to us.” - *Forza Motorsport*

# Why Not Learning for Games?

- Worse Game Play
- More expensive AI Development
  - Lack of programmers with machine learning experience
  - More time to develop, test & debug learning algorithms
  - More time to test range of behaviors
  - *Difficult to “tweak” learned behavior*
- Increased Runtime Computation
  - Computational and memory overhead of learning algorithm
- Marketing Hype Backfire
  - Prior failed attempts

# Marketing Hype

- *“I seriously doubt that BC3K is the first title to employ this technology at any level. The game has been hyped so much that 'neural net' to a casual gamer just became another buzzword and something to look forward to. At least that's my opinion.”*
- Derek Smart



# Alternatives to Learning

## (How to Fake it)

- Pre-program in multiple levels of performance
  - Dynamically switch between levels as player advances
  - Provides pre-defined set of behaviors that can be tested
- Swap in new components [more incremental]
  - Add in more transitions and/or states to a FSM
  - Add in new rules in a rule-based system
- Change parameters during game play
  - The number of mistakes system makes
  - Accuracy in shooting
  - Reaction time to seeing enemy
  - Aggressiveness, ...

# Indirect Adaptation [Manslow]

- Gather pre-defined data for use by AI decision making
  - What is my kill rate with each type of weapon?
  - What is my kill rate in each room?
  - Where is the enemy most likely to be?
  - Does opponent usually pass on the left or the right?
  - How early does the enemy usually attack
- AI “Behavior” code doesn’t change
  - Makes testing much easier
- AI adapts in selected, but significant ways

# Where can we use learning?

- AIs
  - Change behavior of AI entities
- Game environment
  - Optimize the game rules, terrain, interface, infrastructure, ...

# When can we use learning?

- Off-line: during development
  - Train AIs against experts, terrain, each other
  - Automated game balancing, testing, ...
- On-line: during game play
  - AIs adapt to player, environment
  - Dynamic difficulty adjustment

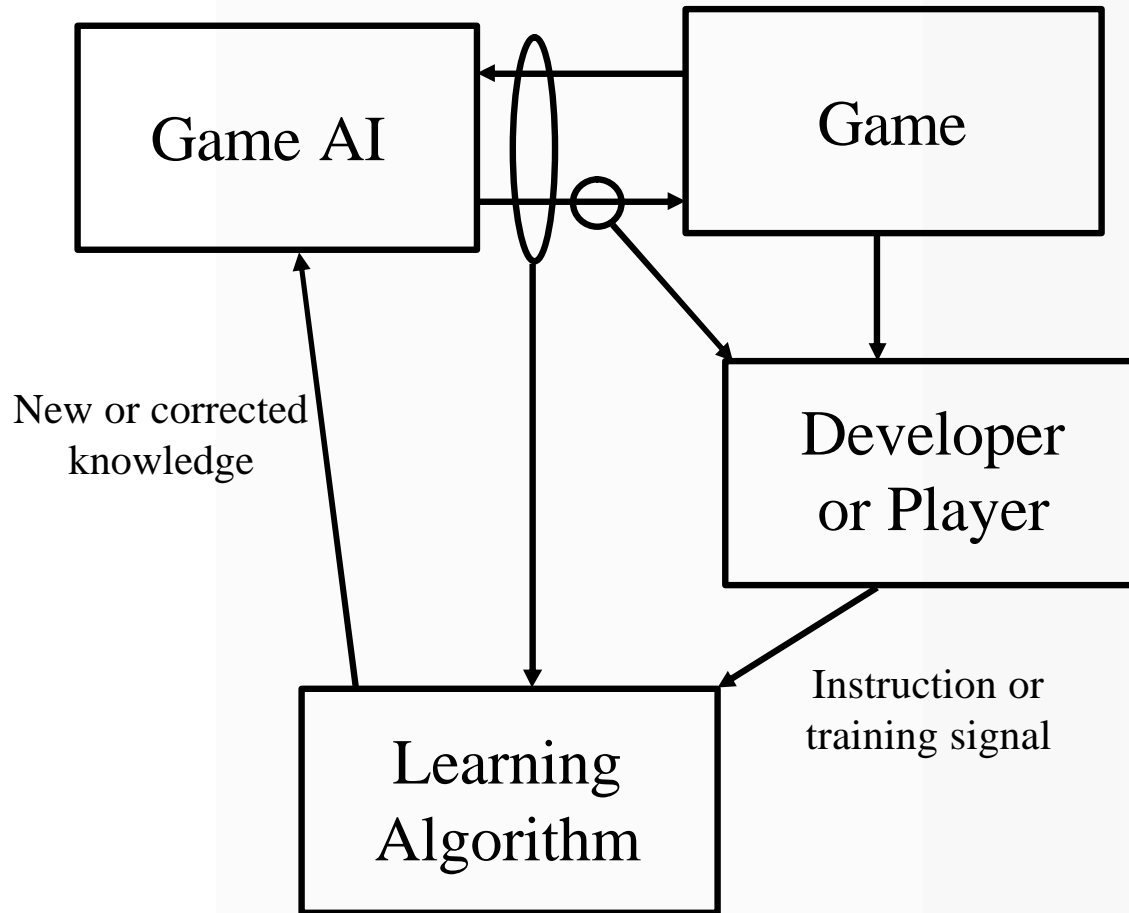
# Basic ML Techniques

- Learning by observation of human behavior
  - Replicate special individual performance
  - Capture variety in expertise, personalities and cultures
  - AI learns from human's performance
- Learning by instruction
  - Non programmers instructing AI behavior
  - Player teaches AI to do his bidding
- Learning from experience
  - Play against other AI and human testers during development
    - Improve behavior and find bogus behavior
  - Play against the environment
    - Find places to avoid, hide, ambush, etc.
  - Adapt tactics and strategy to human opponent

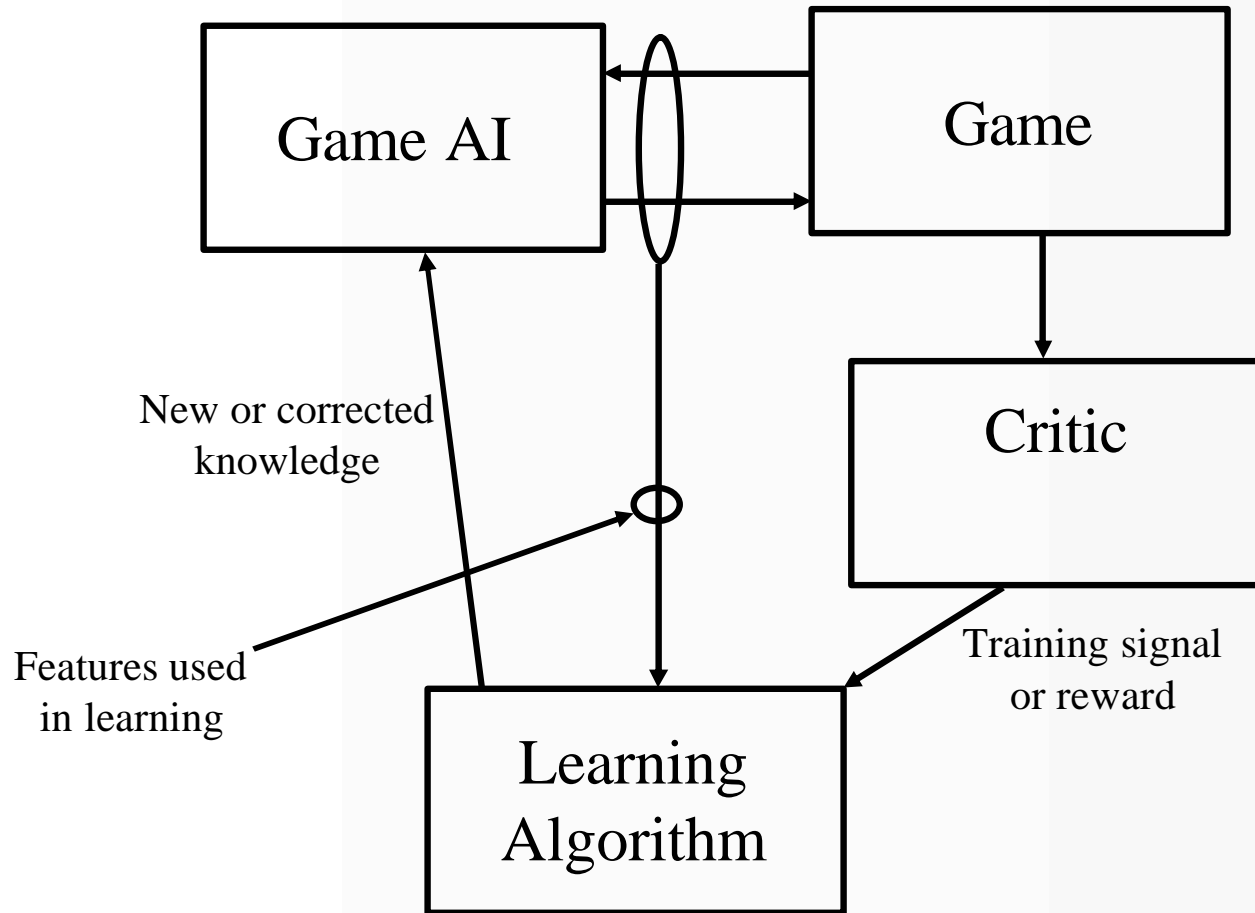
Next



# Learning by Training



# Learning by Experience [Active Learning]



# Game AI Levels

- Low-level actions & Movement
- Situational Assessment (Pattern Recognition)
- Tactical Behavior
- Strategic Behavior
- Opponent Model

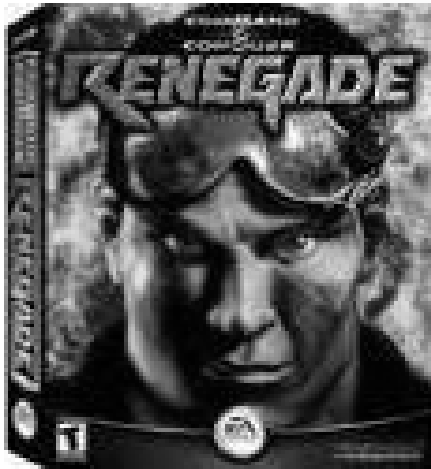
# Actions & Movement: Off Line

- Capture styles of drivers/fighters/skiers
  - More complex than motion capture
  - Includes when to transition from one animation to another
- Train AI against environment:
  - ReVolt: genetic algorithm to discover optimal racing paths



# Actions & Movement: On Line

- Capture style of player for later competition
  - Forza Motorsport
- Learn new paths for AI from humans:
  - Command & Conquer Renegade: internal version noticed paths taken by humans after terrain changes.

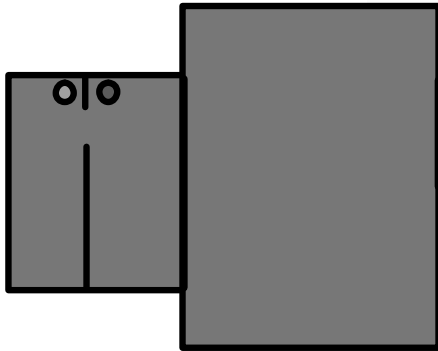


# Demos/Example

- Michiel van de Panne & Ken Alton [UBC]
  - Driving Examples: <http://www.cs.ubc.ca/~kalton/icra.html>
- Andrew Ng [Stanford]
  - Helicopter: <http://www.robotics.stanford.edu/~ang/>

# Learning Situational Assessment

- Learn whether a situation is good or bad
  - Creating an internal model of the environment and relating it to goals
- Concepts that will be useful in decision making and planning
  - Can learn during development or from experience
- Examples
  - Exposure areas (used in path planning)
  - Hiding places, sniping places, dangerous places
  - Properties of objects (edible, destructible, valuable, ...)



# Learning Tactical Behavior

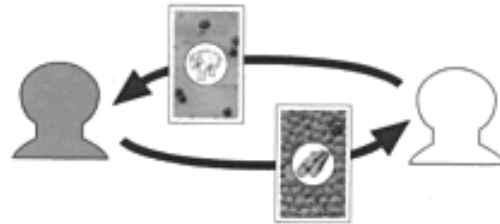
- Selecting and executing appropriate tactics
  - Engage, Camp, Sneak, Run, Ambush, Flee, Flee and Ambush, Get Weapon, Flank Enemy, Find Enemy, Explore
- What weapons work best and when
  - Against other weapons, in what environment, ...
- Train up teammates to fight your style, understand your commands, ...
  - (see talk by John Funge, Xiaoyuan Tu – iKuni, Inc.)
  - Thursday 3:30pm – AK Peters Booth (962)

# Learning Strategic Behavior

- Selecting and executing appropriate strategy
  - Allocation of resources for gathering, tech., defensive, offensive
  - Where to place defenses
  - When to attack, who to attack, where to attack, how to attack
  - Leads to a hierarchy of goals

# Settlers of Catan: Michael Pfeiffer

- Used hierarchical reinforcement learning
  - Co-evolutionary approach
  - Offline: 3000-8000 training games
- Learned primitive actions:
  - Trading, placing roads, ...



- Learning & prior knowledge gave best results

# Review

- When can we use learning?
  - Off-line
  - On-line
- Where can we use learning?
  - Low-level actions
  - Movement
  - Situational Assessment
  - Tactical Behavior
  - Strategic Behavior
  - Opponent Model
- Types of Learning?
  - Learning from experience
  - Learning from training/instruction
  - Learning by observation

# References

- General Machine Learning Overviews:
  - Mitchell: Machine Learning, McGraw Hill, 1997
  - Russell and Norvig: Artificial Intelligence: A Modern Approach, 2003
  - AAI's page on machine learning:
    - <http://www.aaai.org/Pathfinder/html/machine.html>
- Machine Learning for Games
  - <http://www.gameai.com/> - Steve Woodcock's labor of love
  - AI Game Programming Wisdom
  - AI Game Programming Wisdom 2
  - M. Pfeiffer: *Machine Learning Applications in Computer Games*, MSc Thesis, Graz University of Technology, 2003
  - Nick Palmer: Machine Learning in Games Development:
    - <http://ai-depot.com/GameAI/Learning.html>

# Part II

# Overview of Machine Learning

Michael van Lent

# Talk Overview

- Machine Learning Background
- Machine Learning: “The Big Picture”
- Challenges in applying machine learning
- Outline for ML Technique presentations

# AI for Games

- Game AI
  - Entertainment is the central goal
    - The player should win, but only after a close fight
  - Constraints of commercial development
    - Development schedule, budget, CPU time, memory footprint
    - Quality assurance
  - The public face of AI?
- Academic AI
  - Exploring new ideas is the central goal
    - Efficiency and optimality are desirable
  - Constraints of academic research
    - Funding, publishing, teaching, tenure
    - Academics also work on a budget and schedule
  - The next generation of AI techniques?

# Talk Overview

- Machine Learning Background
- Machine Learning “The Big Picture”
- Challenges in applying machine learning
- Outline for ML Technique presentations

# AI: a learning-centric view

Artificial Intelligence requires:

- Architecture and algorithms
- Knowledge
- Interface to the environment

# AI: a learning-centric view

Artificial Intelligence requires:

- Architecture and algorithms
  - Search algorithms
  - Logical & probabilistic inference
  - Planners
  - Expert system shells
  - Cognitive architectures
  - Machine learning techniques
- Knowledge
- Interface to the environment

# AI: a learning-centric view

Artificial Intelligence requires:

- Architecture and algorithms
- Knowledge
  - Knowledge representation
  - Knowledge acquisition
- Interface to the environment

# AI: a learning-centric view

Artificial Intelligence requires:

- Architecture and algorithms
- Knowledge
  - Knowledge representation
    - Finite state machines
    - Rule-based systems
    - Propositional & first-order logic
    - Operators
    - Decision trees
    - Classifiers
    - Neural networks
    - Bayesian networks
  - Knowledge acquisition
- Interface to the environment

# AI: a learning-centric view

Artificial Intelligence requires:

- Architecture and algorithms
- Knowledge
  - Knowledge representation
  - Knowledge acquisition
    - Programming
    - Knowledge engineering
    - Machine Learning
- Interface to the environment

# AI: a learning-centric view

Artificial Intelligence requires:

- Architecture and algorithms
- Knowledge
- Interface to the environment
  - Sensing
  - Acting

# AI: a learning-centric view

Artificial Intelligence requires:

- Architecture and algorithms
- Knowledge
- Interface to the environment
  - Sensing
    - Robotic sensors (sonar, vision, IR, laser, radar)
    - Machine vision
    - Speech recognition
    - Examples
    - Environment features
    - World models
  - Acting

# AI: a learning-centric view

Artificial Intelligence requires:

- Architecture and algorithms
- Knowledge
- Interface to the environment
  - Sensing
  - Acting
    - Navigation
    - Locomotion
    - Speech generation
    - Robotic actuators

# Talk Overview

- Machine Learning Background
- Machine Learning “The Big Picture”
- Challenges in applying machine learning
- Outline for ML Technique presentations

# The Big Picture

Many different ways to group machine learning fields:

(in a somewhat general to specific order)

- **by Problem**
  - What is the fundamental problem being addressed?
  - Broad categorization that groups techniques into a few large classes
- **by Feedback**
  - How much information is given about the right answer?
  - The more information the easier the learning problem
- **by Knowledge Representation**
  - How is the learned knowledge represented/stored/used?
  - Tends to be the basis for a technique's common name
- **by Knowledge Source**
  - Where is the input coming from and in what format?
  - Somewhat orthogonal to the other groupings

# Machine Learning by Problem

- Classification
  - Classify “instances” as one of a discrete set of “categories”
  - Input is often a list of examples
- Clustering
  - Given a data set, identify meaningful “clusters”
    - Unsupervised learning
- Optimization
  - Given a function  $f(x) = y$ , find an input  $x$  with a high  $y$  value
    - Supervised learning
  - Classification can be cast as an optimization problem
    - Function is number of correct classifications on some test set of examples

# Classification Problems

- Task:
  - Classify “instances” as one of a discrete set of “categories”
- Input: set of features about the instance to be classified
  - Inanimate = <true, false>
  - Allegiance = <friendly, neutral, enemy>
  - FoodValue = <none, low, medium, high>
- Output: the category this object fits into
  - Is this object edible? Yes, No
  - How should I react to this object? Attack, Ignore, Heal
- Examples are often split into two data sets
  - Training data
  - Test data

# Example Problem

Classify how I should react to an object in the world

- Facts about any given object include:
  - Inanimate = <true, false>
  - Allegiance = < friendly, neutral, enemy>
  - FoodValue = < none, low, medium, high>
  - Health = <low, medium, full>
  - RelativeHealth = <weaker, same, stronger>
- Output categories include:
  - Reaction = Attack
  - Reaction = Ignore
  - Reaction = Heal
  - Reaction = Eat



- Inanimate=false, Allegiance=enemy, RelativeHealth=weaker => Reaction=Attack
- Inanimate=true, FoodValue=medium => Reaction=Eat
- Inanimate=false, Allegiance=friendly, Health=low => Reaction=Heal
- Inanimate=false, Allegiance=neutral, RelativeHealth=weaker => Reaction=?

# Clustering

Given a list of data points, group them into clusters

- Like classification without the categories identified
- Facts about any given object include:
  - Inanimate = <true, false>
  - Allegiance = <friendly, neutral, enemy>
  - FoodValue = <none, low, medium, high>
  - Health = <low, medium, full>
  - RelativeHealth = <weaker, same, stronger>
- No categories pre-defined
- Find a way to group the following into two groups:
  - Inanimate=false, Allegiance=enemy, RelativeHealth=weaker
  - Inanimate=true, FoodValue=medium
  - Inanimate=false, Allegiance=friendly, Health=low
  - Inanimate=false, Allegiance=neutral, RelativeHealth=weaker

# Optimization

- Task:
  - Given a function  $f(x) = y$ , find an input with a high  $y$  value
  - Input ( $x$ ) can take many forms
    - Feature string
    - Set of classification rules
    - Parse trees of code
- Example:
  - Let  $x$  be a RTS build order  $x = [n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8]$ 
    - $n_i$  means build unit or building  $n$  as the next action
    - If a unit or building isn't available go on to the next action
  - $f([n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8]) =$  firepower of resulting units
  - Optimize the build order for maximum firepower

# Machine Learning by Feedback

- Supervised Learning
  - Correct output is available
  - In Black & White: Examples of things to attack
- Reinforcement Learning
  - Feedback is available but not correct output
  - In Black & White: Getting slapped for attacking something
- Unsupervised Learning
  - No hint about correct outputs
  - In Black & White: Just looking for groupings of objects

# Supervised Learning

- Learning algorithm gets the right answers
  - List of examples as input
  - “Teacher” who can be asked for answers
- Induction
  - Generalize from available examples
    - If X is true in every example X must always be true
  - Often used to learn decision trees and rules
- Explanation-based Learning
- Case-based Learning

# Reinforcement Learning

- Learning algorithm gets positive/negative feedback
  - Evaluation function
  - Rewards from the environment
- Back propagation
  - Pass a reward back across the previous steps
  - Often paired with Neural Networks
- Genetic algorithm
  - Parallel search for a very positive solution
  - Optimization technique
- Q learning
  - Learn the value of taking an action at a state

# Unsupervised Learning

- Learning algorithm gets little or no feedback
  - Don't learn right or wrong answers
- Just recognize interesting patterns of data
  - Similar to data mining
- Clustering is a prime example
- Most difficult class of learning problems

# Machine Learning by Knowledge Representation

- Decision Trees
  - Classification procedure
  - Generally learned by induction
- Rules
  - Flexible representation with multiple uses
  - Learned by induction, genetic algorithms
- Neural Networks
  - Simulates layers of neurons
  - Often paired with back propagation
- Stochastic Models
  - Learning probabilistic networks
  - Takes advantage of prior knowledge

# Machine Learning by Knowledge Source

- Examples
  - Supervised Learning
- Environment
  - Supervised or Reinforcement Learning
- Observation
  - Supervised Learning
- Instruction
  - Supervised or Reinforcement Learning
- Data points
  - Unsupervised Learning

# A Formatting Problem

- Machine learning doesn't generate knowledge
  - Transfers knowledge present in the input into a more useable source
- Examples => Decision Trees
- Observations => Rules
- Data => Clusters

# Talk Overview

- Machine Learning Background
- Machine Learning “The Big Picture”
- Challenges in applying machine learning
- Outline for ML Technique presentations

# Challenges

- What is being learned?
- Where to get good inputs?
- What's the right learning technique?
- When to stop learning?
- How to QA learning?

# What is being learned?

- What are you trying to learn?
  - Often useful to have a sense of good answers in advance
  - Machine learning often finds more/better variations
  - Novel, unexpected solutions don't appear often
- What are the right features?
  - This can be the difference between success and failure
  - Balance what's available, what's useful
  - If features are too good there's nothing to learn
- What's the right knowledge representation?
  - Again, difference between success and failure
  - Must be capable of representing the solution

# Where to get good inputs?

- Getting good examples is essential
  - Need enough for useful generalization
  - Need to avoid examples that represent only a subset of the space
  - Creating a long list of examples can take a lot of time
- Human experts
  - Observations, Logs, Traces
  - Examples
- Other AI systems
  - AI prototypes
  - Similar games

# What's the right learning technique?

- This often falls out of the other decisions
- Knowledge representations tend to be associated with techniques
  - Decision trees go with induction
  - Neural networks go with back propagation
  - Stochastic models go with Bayesian learning
- Often valuable to try out more than one approach

# When to stop learning?

- Sometimes more learning is not better
  - More learning might not improve the solution
  - More learning might result in a worse solution
- Overfitting
  - Learned knowledge is too specific to the provided examples
  - Looks very good on training data
  - Can look good on test data
  - Doesn't generalize to new inputs

# How to QA learning?

- Central challenge in applying machine learning to games
  - Adds a big element of variability into the player's experience
  - Adds an additional risk factor to the development process
- Offline learning
  - The result can undergo standard play testing
  - Might be hard or impossible to debug learned knowledge
    - Neural networks are difficult to understand
- Online learning
  - Constrain the space learning can explore
    - Carefully design and bound the knowledge representation
    - Consider “instincts” or rules than learned knowledge can't violate
  - Allow players to activate/deactivate learning

# Talk Overview

- Machine Learning Background
- Machine Learning “The Big Picture”
- Challenges in applying machine learning
- Non-learning learning
- Outline for ML mechanism presentations
  - Decision Trees
  - Neural Networks
  - Genetic Algorithms
  - Bayesian Networks
  - Reinforcement Learning

# Outline

- Background
- Technical Overview
- Example
- Games that have used this mechanism
- Pros, Cons & Challenges
- References

# General Machine Learning References

- Artificial Intelligence: A Modern Approach
  - Russell & Norvig
- Machine Learning
  - Mitchell
- Gameai.com
- AI Game Programming Wisdom books
- Game Programming Gems

# Decision Trees & Rule Induction

Michael van Lent

# The Big Picture

- Problem
  - Classification
- Feedback
  - Supervised learning
  - Reinforcement learning
- Knowledge Representation
  - Decision tree
  - Rules
- Knowledge Source
  - Examples

# Decision Trees

- Nodes represent attribute tests
  - One child for each possible value of the attribute
- Leaves represent classifications
- Classify by descending from root to a leaf
  - At root test attribute associated with root attribute test
  - Descend the branch corresponding to the instance's value
  - Repeat for subtree rooted at the new node
  - When a leaf is reached return the classification of that leaf
- Decision tree is a disjunction of conjunctions of constraints on the attribute values of an instance

# Example Problem

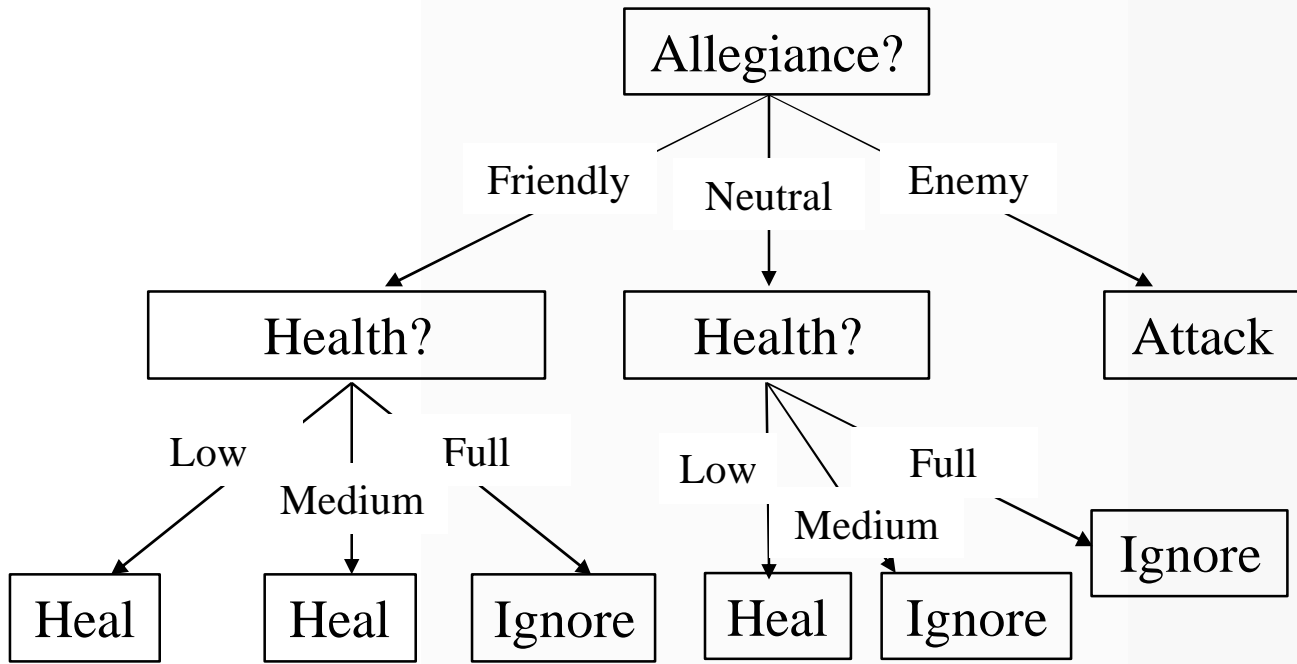
Classify how I should react to an object in the world

- Facts about any given object include:
  - Allegiance = < friendly, neutral, enemy >
  - Health = < low, medium, full >
  - Animate = < true, false >
  - RelativeHealth = < weaker, same, stronger >
- Output categories include:
  - Reaction = Attack
  - Reaction = Ignore
  - Reaction = Heal
  - Reaction = Eat
  - Reaction = Run

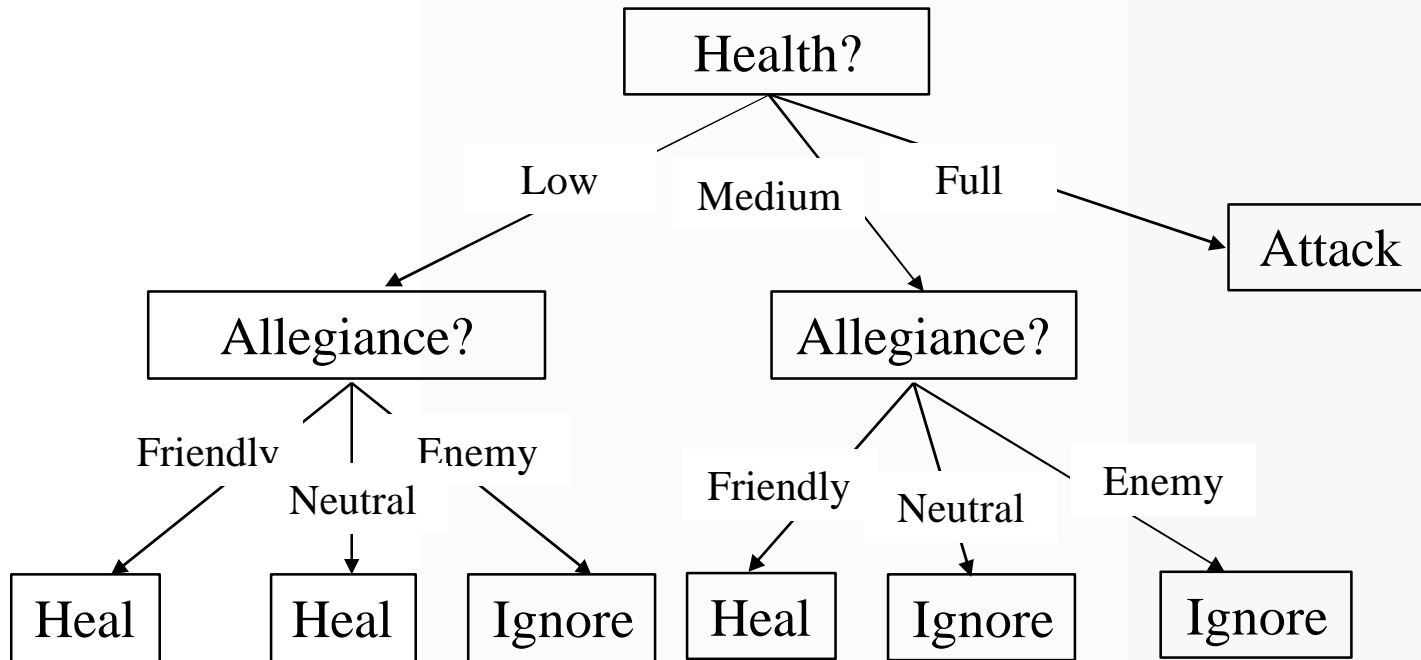


- <friendly, low, true, weaker> => Heal
- <neutral, low, true, same> => Heal
- <enemy, low, true, stronger> => Attack
- <enemy, medium, true, weaker> => Attack

# Classifying with a Decision Tree



# Classifying with a Decision Tree



# Decision Trees are good when:

- Inputs are attribute-value pairs
  - With fairly small number of values
  - Numeric or continuous values cause problems
    - Can extend algorithms to learn thresholds
- Outputs are discrete output values
  - Again fairly small number of values
  - Difficult to represent numeric or continuous outputs
- Disjunction is required
  - Decision trees easily handle disjunction
- Training examples contain errors
  - Learning decision trees
  - More later

# Learning Decision Trees

- Decision trees are usually learned by induction
  - Generalize from examples
  - Induction doesn't guarantee correct decision trees
- Bias towards smaller decision trees
  - Occam's Razor: Prefer simplest theory that fits the data
  - Too expensive to find the very smallest decision tree
- Learning is non-incremental
  - Need to store all the examples
- ID3 is the basic learning algorithm
  - C4.5 is an updated and extended version

# Induction

- If  $X$  is true in every example  $X$  must always be true
  - More examples are better
  - Errors in examples cause difficulty
  - Note that induction can result in errors
- Inductive learning of Decision Trees
  - Create a decision tree that classifies the available examples
  - Use this decision tree to classify new instances
  - Avoid over fitting the available examples
    - One root to node path for each example
    - Perfect on the examples, not so good on new instances

# Induction requires Examples

- Where do examples come from?
  - Programmer/designer provides examples
  - Observe a human's decisions
- # of examples need depends on difficulty of concept
  - More is always better
- Training set vs. Testing set
  - Train on most (75%) of the examples
  - Use the rest to validate the learned decision trees

# ID3 Learning Algorithm

- ID3 has two parameters
  - List of examples
  - List of attributes to be tested
- Generates tree recursively
  - Chooses attribute that best divides the examples at each step

ID3(examples, attributes)

if all examples in same category then

return a leaf node with that category

if attributes is empty then

return a leaf node with the most common category in examples

best = Choose-Attribute(examples, attributes)

tree = new tree with Best as root attribute test

foreach value  $v_i$  of best

examples<sub>i</sub> = subset of examples with best ==  $v_i$

subtree = ID3(examples<sub>i</sub>, attributes – best)

add a branch to tree with best ==  $v_i$  and subtree beneath

return tree

# Examples

- <friendly, low, true, weaker> => Heal
  - <neutral, full, false, same> => Eat
  - <enemy, low, true, weaker> => Eat
  - <enemy, low, true, same> => Attack
  - <neutral, low, true, weaker> => Heal
  - <enemy, medium, true, stronger> => Run
  - <friendly, full, true, same> => Ignore
  - <neutral, full, true, stronger> => Ignore
  - <enemy, full, true, same> => Run
  - <enemy, medium, true, weaker> => Attack
  - <friendly, full, true, weaker> => Ignore
  - <neutral, full, false, stronger> => Ignore
  - <friendly, medium, true, stronger> => Heal
- 13 examples
    - 3 Heal
    - 2 Eat
    - 2 Attack
    - 4 Ignore
    - 2 Run

# Entropy

- Entropy: how “mixed” is a set of examples
  - All one category: Entropy = 0
  - Evenly divided: Entropy =  $\log_2(\# \text{ of examples})$
- Given  $S$  examples Entropy( $S$ ) =  $S \sum -p_i \log_2 p_i$   
where  $p_i$  is the proportion of  $S$  belonging to class  $i$ 
  - 13 examples with 3 heal, 2 attack, 2 eat, 4 ignore, 2 run
    - Entropy([3,2,2,4,2]) = 2.258
  - 13 examples with all 13 heal
    - Entropy ([13,0,0,0,0]) = 0
  - Maximum entropy is  $\log_2 5 = 2.322$ 
    - 5 is the number of categories

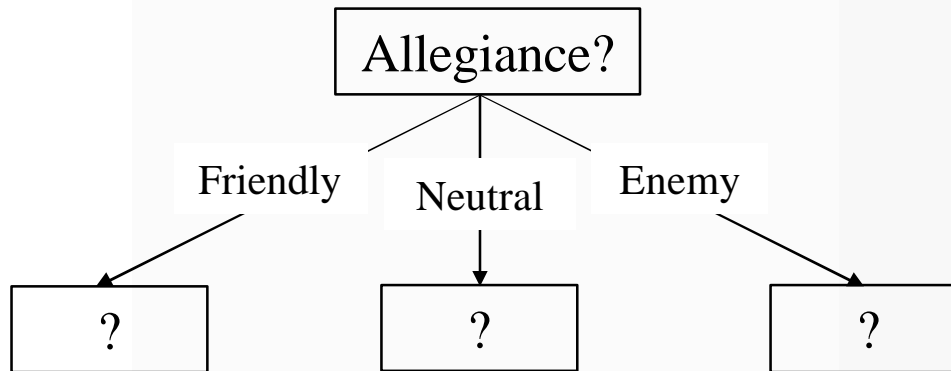
# Information Gain

- Information Gain measures the reduction in Entropy
  - $\text{Gain}(S,A) = \text{Entropy}(S) - \sum_{v \in V} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$
- Example: 13 examples:  $\text{Entropy}([3,2,2,4,2]) = 2.258$ 
  - Information gain of Allegiance =  $\langle \text{friendly, neutral, enemy} \rangle$ 
    - Allegiance = friendly for 4 examples [2,0,0,2,0]
    - Allegiance = neutral for 4 examples [1,1,0,2,0]
    - Allegiance = enemy for 5 examples [0,1,2,0,2]
    - $\text{Gain}(S,\text{Allegiance}) = 0.903$
  - Information gain of Animate =  $\langle \text{true, false} \rangle$ 
    - Animate = true for 11 examples [3,1,2,3,2]
    - Animate = false for 2 examples [0,1,0,1,0]
    - $\text{Gain}(S,\text{Animate}) = 0.216$
  - Allegiance has a higher information gain than Animate
    - So choose allegiance as the next attribute to be tested

# Learning Example

- Information gain of Allegiance
  - 0.903
- Information gain of Health
  - 0.853
- Information gain of Animate
  - 0.216
- Information gain of RelativeHealth
  - 0.442
- So Allegiance should be the root test

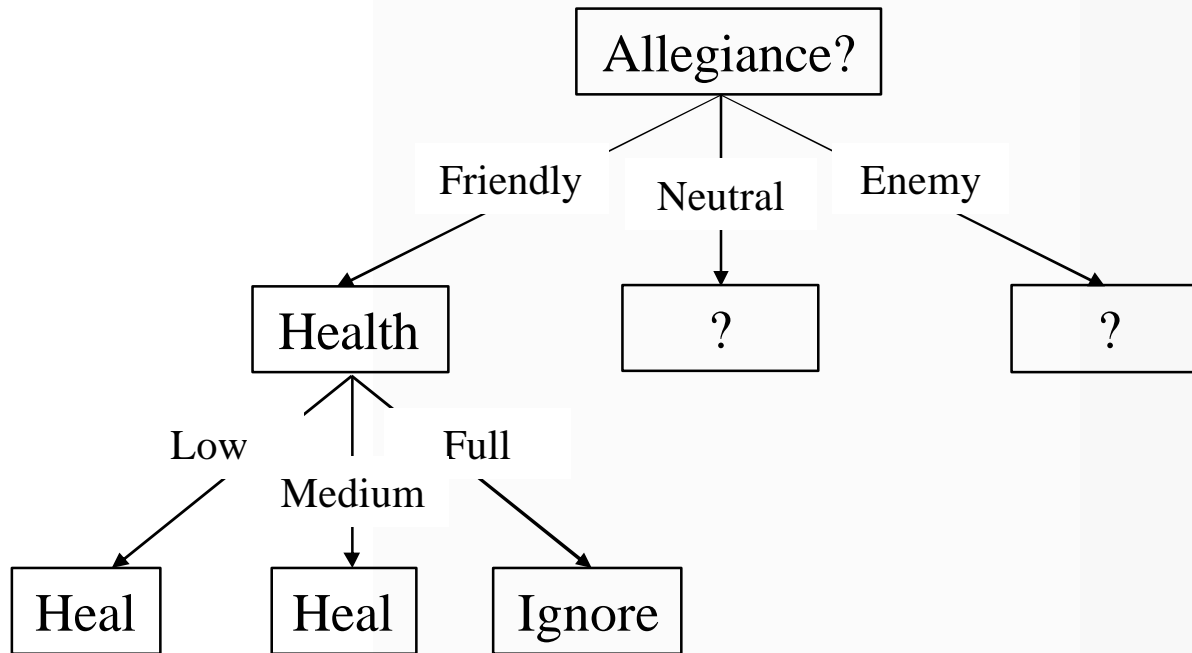
# Decision tree so far



# Allegiance = friendly

- Four examples have allegiance = friendly
  - Two categorized as Heal
  - Two categorized as Ignore
  - We'll denote this now as [# of Heal, # of Ignore]
  - Entropy = 1.0
- Which of the remaining features has the highest info gain?
  - Health: low [1,0], medium [1,0], full [0,2] => Gain is 1.0
  - Animate: true [2,2], false [0,0] => Gain is 0
  - RelativeHealth: weaker [1,1], same [0,1], stronger [1,0] => Gain is 0.5
- Health is the best (and final) choice

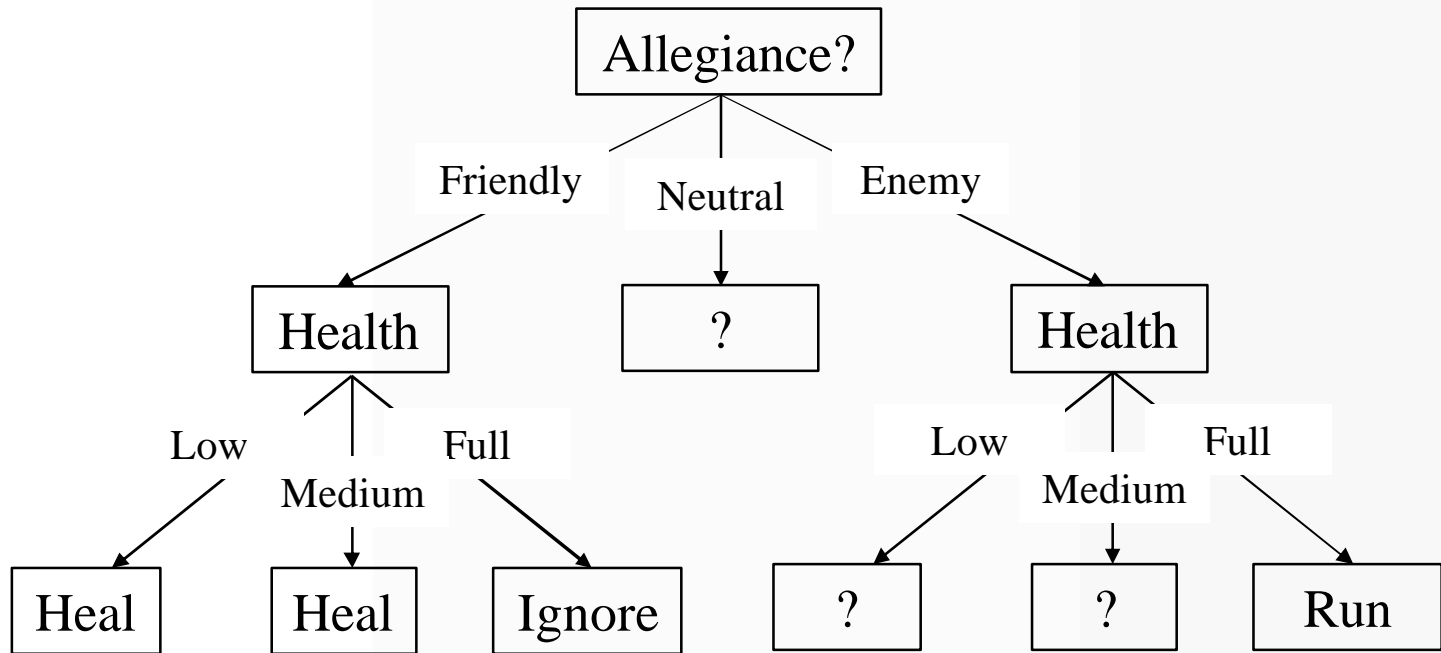
# Decision tree so far



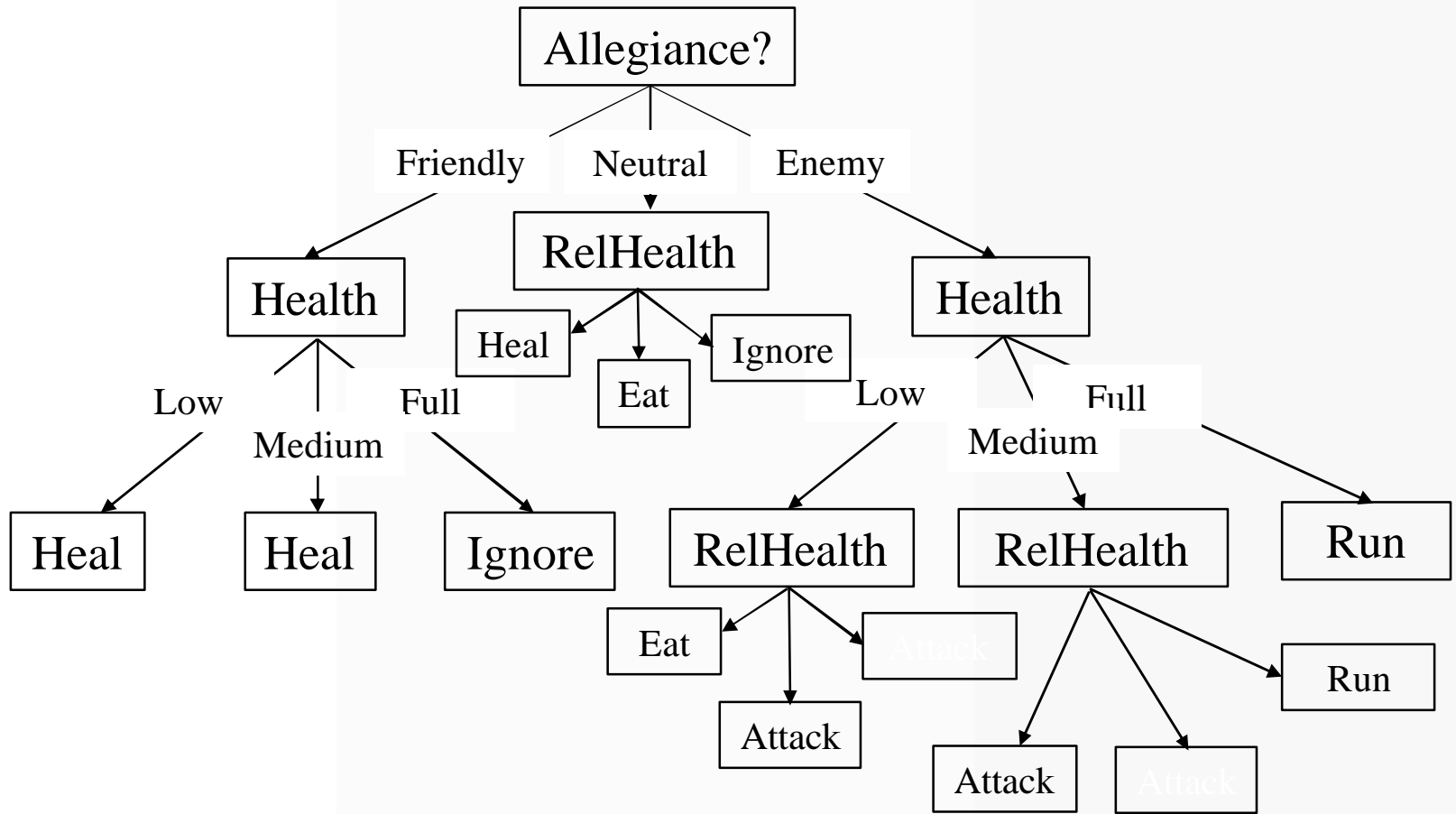
# Allegiance = enemy

- Five examples have allegiance = enemy
  - One categorized as Eat
  - Two categorized as Attack
  - Two categorized as Run
  - We'll denote this now as [# of Eat, # of Attack, # of Run]
  - Entropy = 1.5
- Which of the remaining features has the highest info gain?
  - Health: low [1,1,0], medium [0,1,1], full [0,0,1] => Gain is 0.7
  - Animate: true [1,2,2], false [0,0,0] => Gain is 0
  - RelHealth: weaker [1,1,0], same [0,1,1], stronger [0,0,1] => Gain is 0.7
- Health and RelativeHealth are equally good choices

# Decision tree so far



# Final Decision Tree



# Generalization

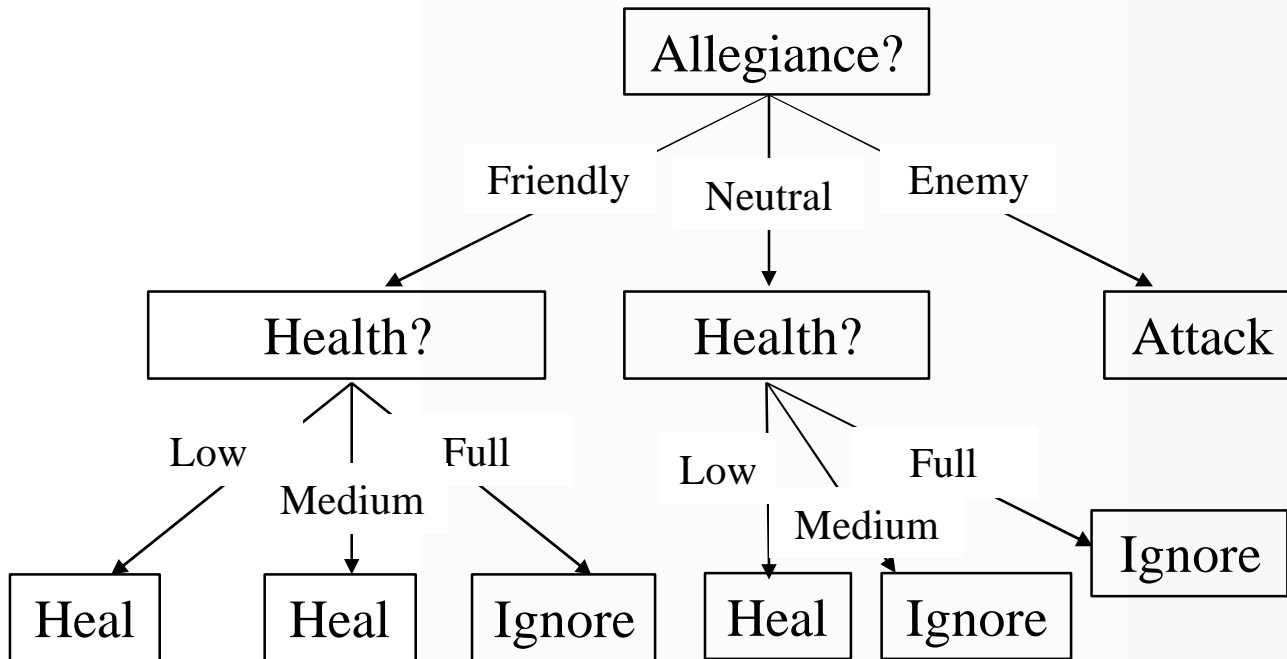
- Previously unseen examples can be classified
  - Each path through the decision tree doesn't test every feature
  - $\langle \text{neutral, low, false, stronger} \rangle \Rightarrow \text{Eat}$
- Some leaves don't have corresponding examples
  - $(\text{Allegiance}=\text{enemy}) \ \& \ (\text{Health}=\text{low}) \ \& \ (\text{RelHealth}=\text{stronger})$
  - Don't have any examples of this case
  - Generalize from the closest example
  - $\langle \text{enemy, low, false, same} \rangle \Rightarrow \text{Attack}$
  - Guess that:  $\langle \text{enemy, low, false, stronger} \rangle \Rightarrow \text{Attack}$

# Decision trees in Black & White

- Creature learns to predict the player's reactions
  - Instead of categories, range [-1 to 1] of predicted feedback
  - Extending decision trees for continuous values
    - Divide into discrete categories
    - ...
- Creature generates examples by experimenting
  - Try something and record the feedback (tummy rub, slap...)
  - Starts to look like reinforcement learning
- Challenges encountered
  - Ensuring everything that can be learned is reasonable
  - Matching actions with player feedback

# Decision Trees and Rules

- Decision trees can easily be translated into rules
  - and vice versa



If (Allegiance=friendly) & ((Health=low) | (Health=medium)) then Heal

If (Allegiance=friendly) & (Health=high) then Ignore

If (Allegiance=neutral) & (Health=low) then Heal

...

If (Allegiance=enemy) then Attack

# Rule Induction

- Specific to General Induction
  - First example creates a very specific rule
  - Additional examples are used to generalize the rule
  - If rule becomes too general create a new, disjunctive rule
- Version Spaces
  - Start with a very specific rule and a very general rule
  - Each new example either
    - Makes the specific rule more general
    - Makes the general rule more specific
  - The specific and general rules meet at the solution

# Learning Example

- First example:  $\langle \text{friendly, low, true, weaker} \rangle \Rightarrow \text{Heal}$ 
  - If (Allegiance=friendly) & (Health=low) & (Animate=true) & (RelHealth=weaker) then Heal
- Second example:  $\langle \text{neutral, low, true, weaker} \rangle \Rightarrow \text{Heal}$ 
  - If (Health=low) & (Animate=true) & (RelHealth=weaker) then Heal
    - Overgeneralization?
  - If ((Allegiance=friendly) | (Allegiance=neutral)) & (Health=low) & (Animate=true) & (RelHealth=weaker) then Heal
- Third example:  $\langle \text{friendly, medium, true, stronger} \rangle \Rightarrow \text{Heal}$ 
  - If ((Allegiance=friendly) | (Allegiance=neutral)) & ((Health=low) | (Health=medium)) & (Animate=true) & ((RelHealth=weaker) | (RelHealth=stronger)) then Heal

# Advanced Topics

- Boosting
  - Manipulate the set of training examples
  - Increase the representation of incorrectly classified examples
- Ensembles of classifiers
  - Learn multiple classifiers (i.e. multiple decision trees)
    - All the classifiers vote on the correct answer (only one approach)
  - “Bagging”: break the training set into overlapping subsets
    - Learn a classifier for each subset
  - Learn classifiers using different subsets of features
    - Or different subsets of categories
  - Ensembles can be more accurate than a single classifier

# Games that use inductive learning

- Decision Trees
  - Black & White
- Rules

# Inductive Learning Evaluation

- Pros
  - Decision trees and rules are human understandable
  - Handle noisy data fairly well
  - Incremental learning
  - Online learning is feasible
- Cons
  - Need many, good examples
  - Overfitting can be an issue
  - Learned decision trees may contain errors
- Challenges
  - Picking the right features
  - Getting good examples

# References

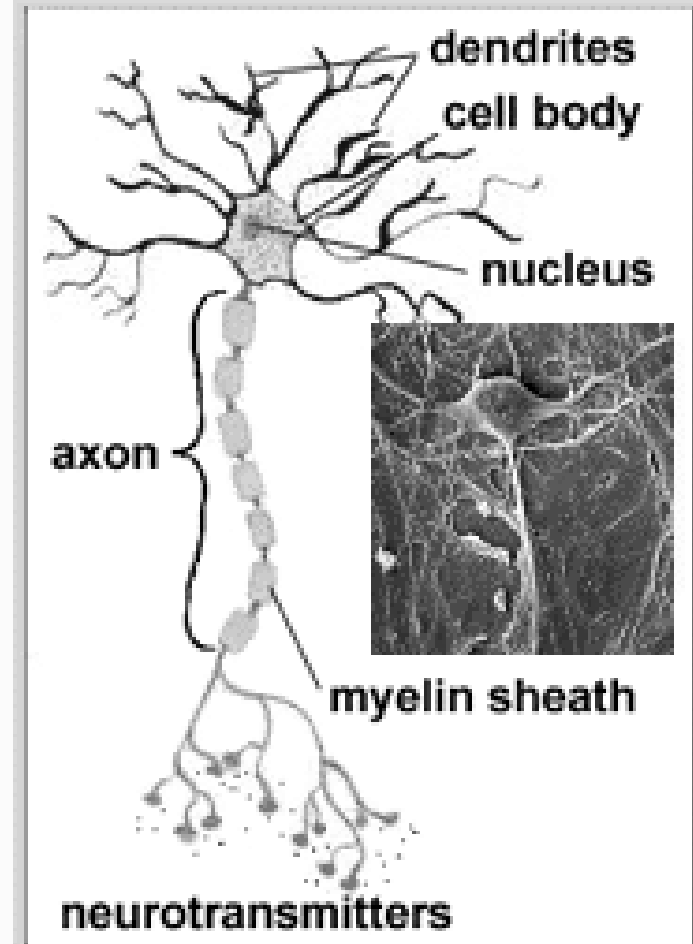
- Mitchell: Machine Learning, McGraw Hill, 1997.
- Russell and Norvig: Artificial Intelligence: A Modern Approach, Prentice Hall, 1995.
- Quinlan: Induction of decision trees, Machine Learning 1:81-106, 1986.
- Quinlan: Combining instance-based and model-based learning, 10<sup>th</sup> International Conference on Machine Learning, 1993.
- AI Game Programming Wisdom.
- AI Game Programming Wisdom 2.

# Neural Networks

John Laird

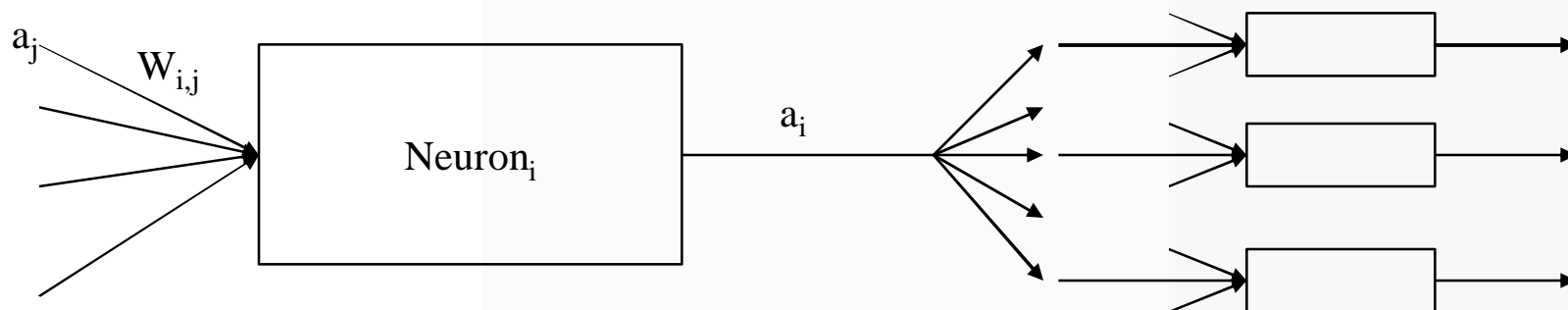
# Inspiration

- Mimic natural intelligence
  - Networks of simple neurons
  - Highly interconnected
  - Adjustable weights on connections
  - Learn rather than program
- Architecture is different
  - Brain is massively parallel
    - $10^{12}$  neurons
  - Neurons are slow
    - Fire 10-100 times a second



# Simulated Neuron

- Neurons are simple computational devices whose power comes from how they are connected together
  - Abstractions of real neurons
- Each neuron has:
  - Inputs/activation from other neurons ( $a_j$ )  $[-1, +1]$
  - Weights of input ( $W_{i,j}$ )  $[-1, +1]$
  - Output to other neurons ( $a_i$ )



# Simulated Neuron

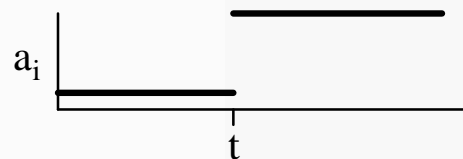
- Neuron calculates weighted sum of inputs ( $in_i$ )

- $in_i = \sum W_{i,j} a_j$

- Threshold function  $g(in_i)$  calculates output ( $a_i$ )

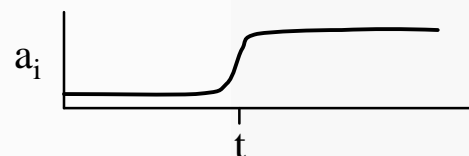
- Step function:

- if  $in_i > t$  then  $a_i = 1$  else  $a_i = 0$

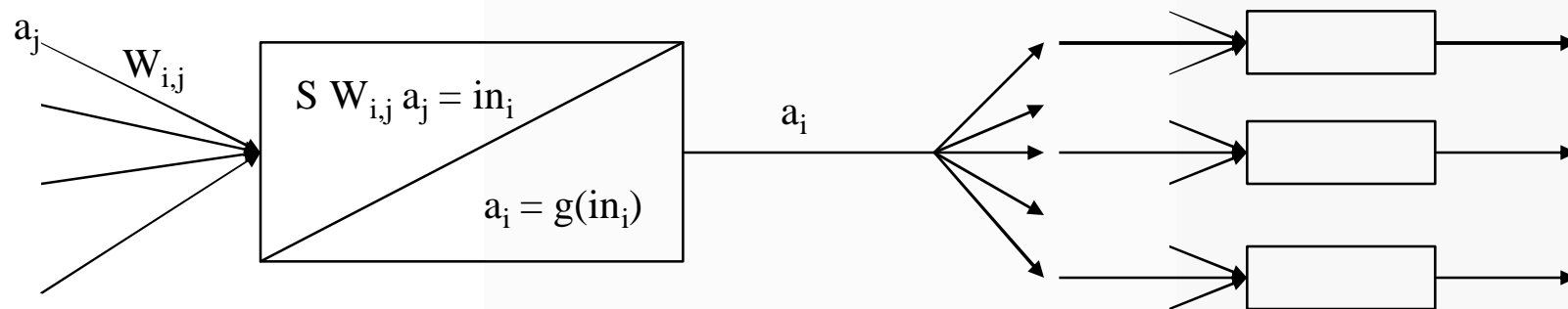


- Sigmoid:

- $a_i = 1/(1+e^{-in_i})$

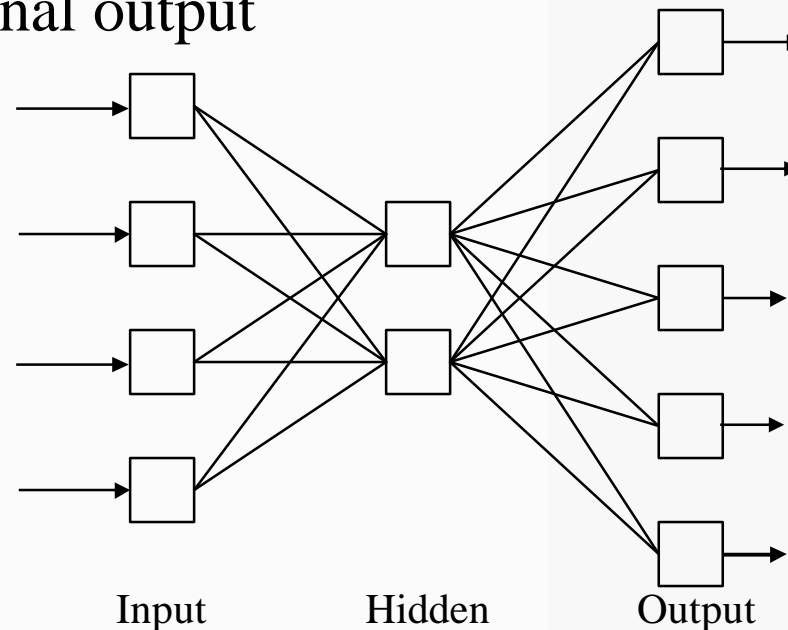


- Output becomes input for next layer of neurons



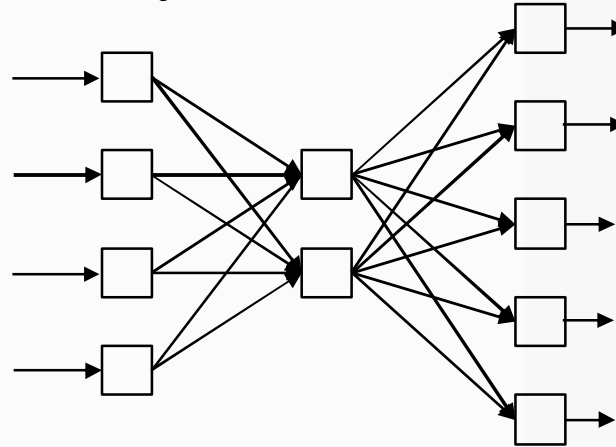
# Network Structure

- Single neuron can represent AND, OR not XOR
  - Combinations of neuron are more powerful
- Neuron are usually organized as layers
  - Input layer: takes external input
  - Hidden layer(s)
  - Output player: external output

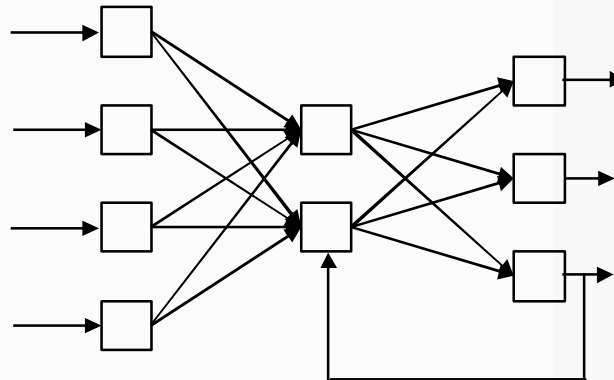


# Feed-forward vs. recurrent

- Feed-forward: outputs only connect to later layers
  - Learning is easier

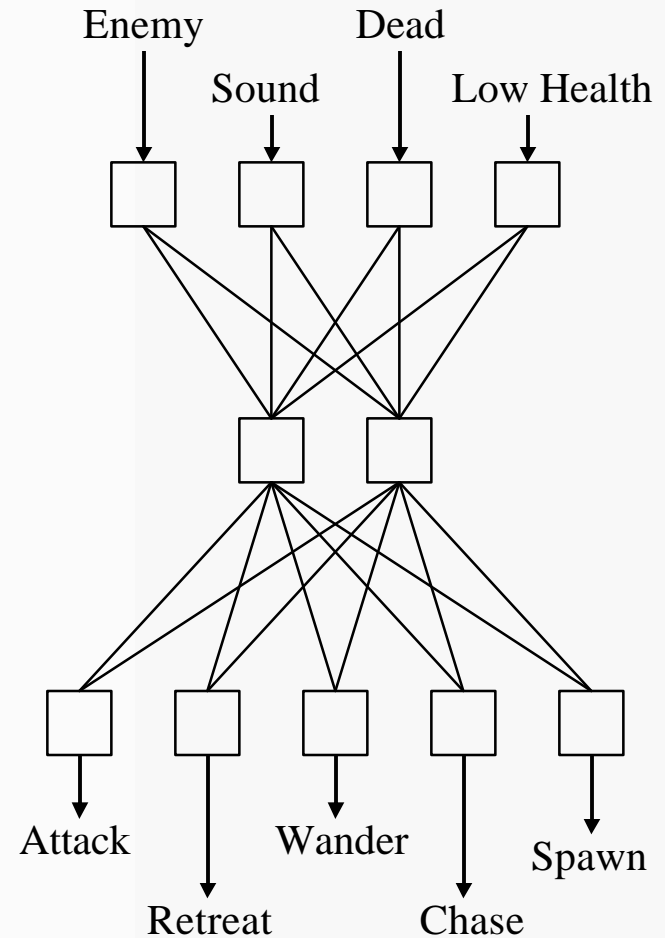


- Recurrent: outputs connect to earlier layers
  - Internal state



# Neural Network for a FPS-bot

- Four input neuron
  - One input for each condition
- Two neuron hidden layer
  - Fully connected
  - Forces generalization
- Five output neuron
  - One output for each action
  - Choose action with highest output
  - Probabilistic action selection



# Learning Weights: Back Propagation

- Learning from examples
  - Examples consist of input and correct output (t)
- Learn if network's output doesn't match correct output
  - Adjust weights to reduce difference
  - Only change weights a small amount (?)
- Basic neuron learning
  - $W_{i,j} = W_{i,j} + ? W_{i,j}$
  - $W_{i,j} = W_{i,j} + ?(t-o)a_j$
  - If output is too high, (t-o) is negative so  $W_{i,j}$  will be reduced
  - If output is too low, (t-o) is positive so  $W_{i,j}$  will be increased
  - If  $a_j$  is negative the opposite happens

# Back propagation algorithm

Repeat

  Foreach e in examples do

    O = Run-Network(network,e)

    // Calculate error term for output layer

    Foreach neuron in the output layer do

$$\text{Err}_k = o_k(1-o_k)(t_k-o_k)$$

    // Calculate error term for hidden layer

    Foreach neuron in the hidden layer do

$$\text{Err}_h = o_h(1-o_h)S w_{kh} \text{Err}_k$$

    // Update weights of all neurons

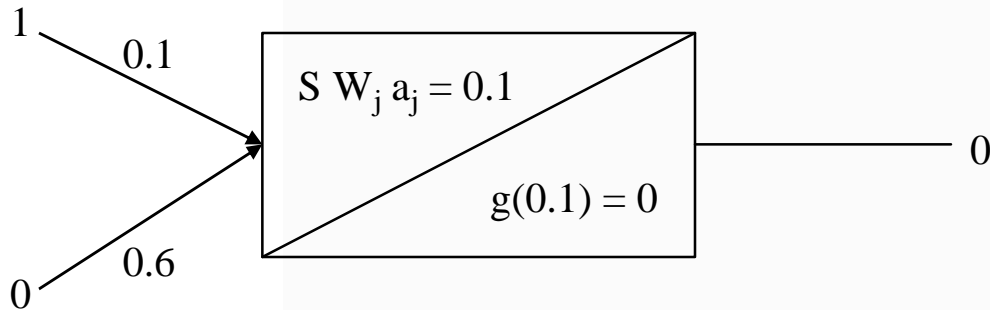
    Foreach neuron do

$$W_{i,j} = W_{i,j} + ? (x_{ij}) \text{Err}_j$$

Until network has converged

# Neural Net Example

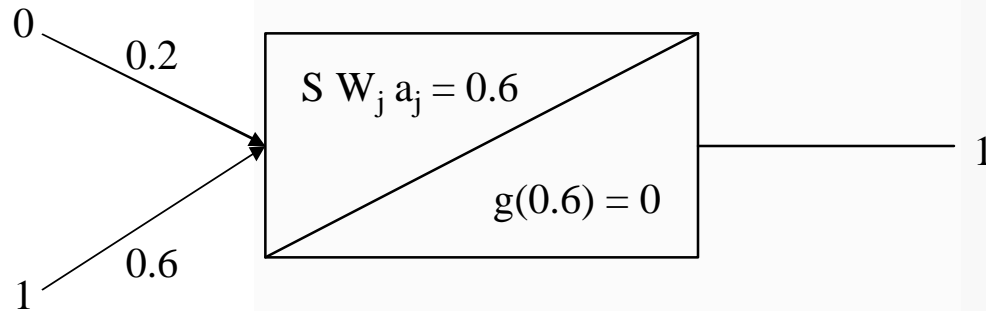
- Single neuron to represent OR
  - Two inputs
  - One output (1 if either inputs is 1)
  - Step function (if weighted sum  $> 0.5$  output  $a = 1$ )



- Error so training occurs

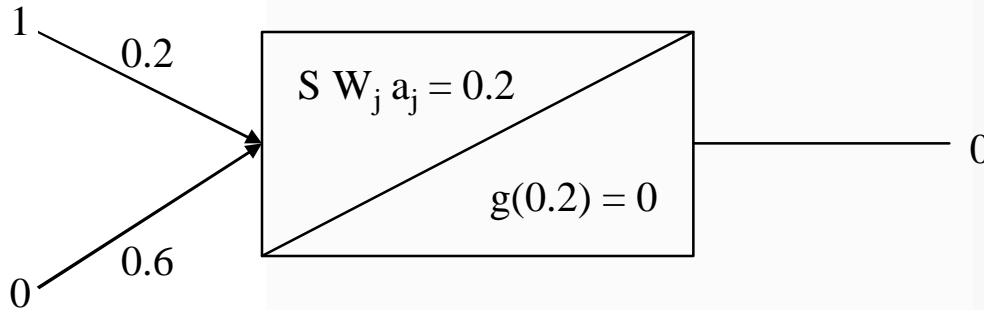
# Neural Net Example

- $W_j = W_j + ? W_j$
- $W_j = W_j + ?(t-o)a_j$
- $W_1 = 0.1 + 0.1(1-0)1 = 0.2$
- $W_2 = 0.6 + 0.1(1-0)0 = 0.6$

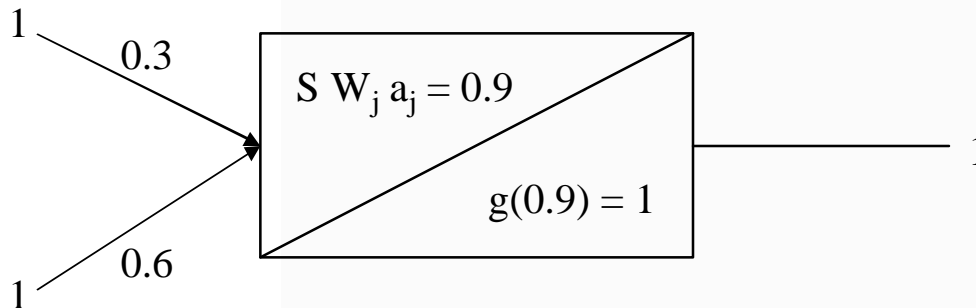


- No error so no training occurs

# Neural Net Example



- Error so training occurs
- $W_1 = 0.2 + 0.1(1-0)1 = 0.3$
- $W_2 = 0.6 + 0.1(1-0)0 = 0.6$



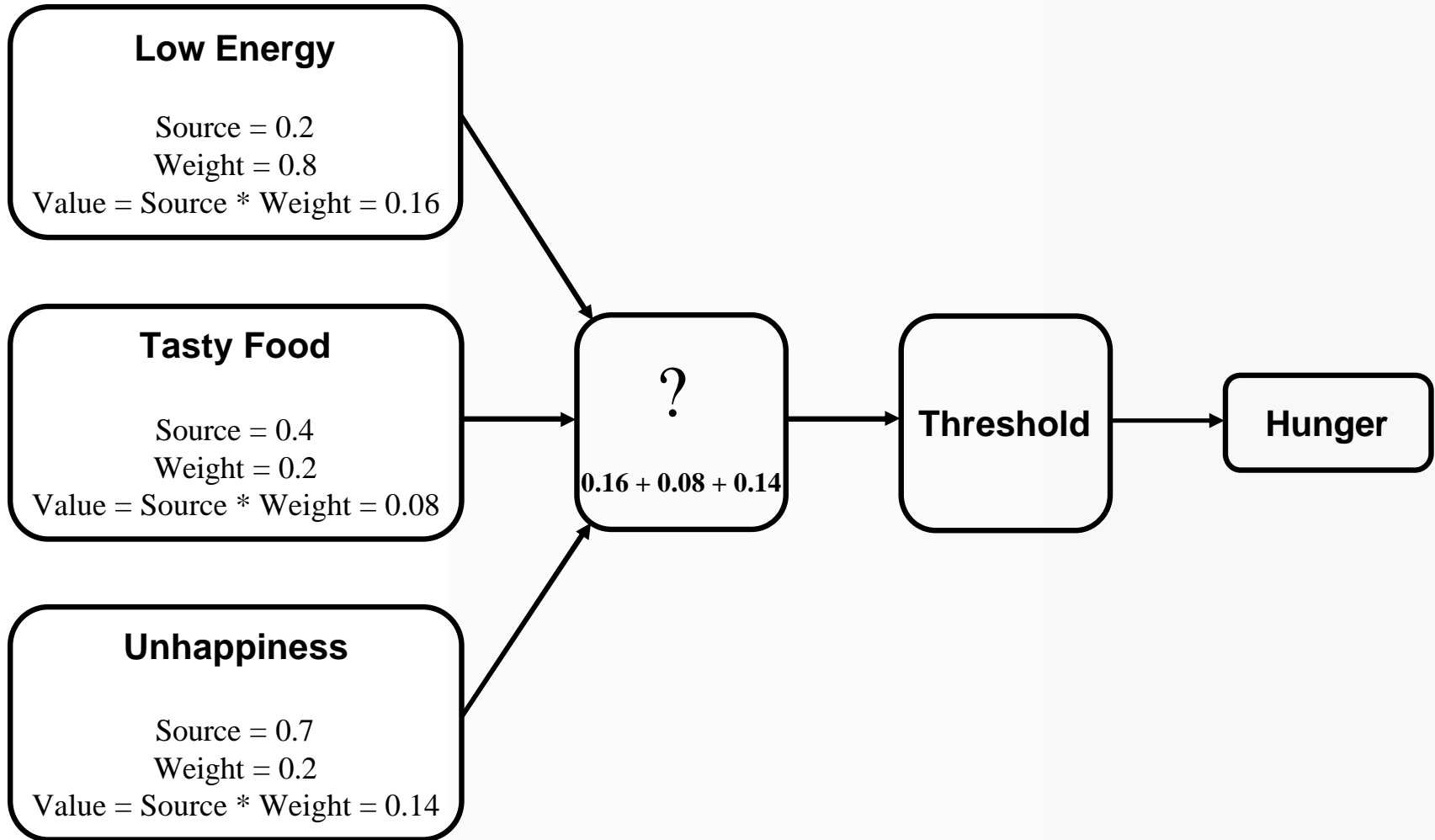
# Using Neural Networks in Games

- Classification/function approximation
  - In game or during development
- Learning to predict the reward associated with a state
  - Can be the core of reinforcement learning
- Situational Assessment/Classification
  - Feelings toward objects in world or other players
  - Black & White BC3K
- Predict enemy action

# Neural Network Example Systems

- BattleCruiser: 3000AD
  - Guide NPC: Negotiation, trading, combat
- Black & White
  - Teach creatures desires and preferences
- Creatures
  - Creature behavior control
- Dirt Track Racing
  - Race track driving control
- Heavy Gear
  - Multiple NNs for control

# NN Example: B & W



# Neural Networks Evaluation

- Advantages
  - Handle errors well
  - Graceful degradation
  - Can learn novel solutions
- Disadvantages
  - Feed forward doesn't have memory of prior events
  - Can't understand how or why the learned network works
  - Usually requires experimentation with parameters
  - Learning takes *lots* of processing
    - Incremental so learning during play might be possible
  - Run time cost is related to number of connections
- Challenges
  - Picking the right features
  - Picking the right learning parameters
  - Getting lots of data

# References

- General AI Neural Network References:
  - Mitchell: Machine Learning, McGraw Hill, 1997
  - Russell and Norvig: Artificial Intelligence: A Modern Approach, Prentice Hall, 2003
  - Hertz, Krogh & Palmer: Introduction to the theory of neural computation, Addison-Wesley, 1991
  - Cowan & Sharp: Neural nets and artificial intelligence, Daedalus 117:85-121, 1988
- Neural Networks in Games:
  - Penny Sweetser, How to Build Neural Networks for Games
    - AI Programming Wisdom 2
  - Mat Buckland, Neural Networks in Plain English, *AI-Junkie.com*
  - John Manslow, Imitating Random Variations in Behavior using Neural Networks
    - AI Programming Wisdom, p. 624
  - Alex Chamandard, The Dark Art of Neural Networks
    - AI Programming Wisdom, p. 640
  - John Manslow, Using a Neural Network in a Game: A Concrete Example
    - Game Programming Gems 2

# Genetic Algorithms

Michael van Lent

# Background

- Evolution creates individuals with higher fitness
  - Population of individuals
    - Each individual has a genetic code
  - Successful individuals (higher fitness) more likely to breed
    - Certain codes result in higher fitness
    - Very hard to know ahead which combination of genes = high fitness
  - Children combine traits of parents
    - Crossover
    - Mutation
- Optimize through artificial evolution
  - Define fitness according to the function to be optimized
  - Encode possible solutions as individual genetic codes
  - Evolve better solutions through simulated evolution

# The Big Picture

- Problem
  - Optimization
  - Classification
- Feedback
  - Reinforcement learning
- Knowledge Representation
  - Feature String
  - Classifiers
  - Code (Genetic Programming)
- Knowledge Source
  - Evaluation function

# Genes

- Gene is typically a string of symbols
  - Frequently a bit string
  - Gene can be a simple function or program
    - Evolutionary programming
- Challenges in gene representation
  - Every possible gene should encode a valid solution
- Common representation
  - Coefficients
  - Weights for state transitions in a FSM
  - Classifiers
  - Code (Genetic Programming)
  - Neural network weights

# Classifiers

- Classification rules encoded as bit strings
  - Bits 1-3: Allegiance (1=friendly, 2=neutral, 3=enemy)
  - Bits 4-6: Health (4=low, 5=medium, 6=full)
  - Bits 7-8: Animate (7=true, 8=false)
  - Bits 9-11: RelHealth (9=weaker, 10=same, 11=stronger)
  - Bits 12-16: Action(Attack, Ignore, Heal, Eat, Run)
- Example
  - If  $((\text{Allegiance}=\text{friendly}) \mid (\text{Allegiance}=\text{neutral})) \ \& \ ((\text{Health}=\text{low}) \mid (\text{Health}=\text{medium})) \ \& \ (\text{Animate}=\text{true}) \ \& \ ((\text{RelHealth}=\text{weaker}) \mid (\text{RelHealth}=\text{stronger}))$  then Heal
  - 110 110 10 101 00100
  - Need to ensure that bits 12-16 are mutually exclusive

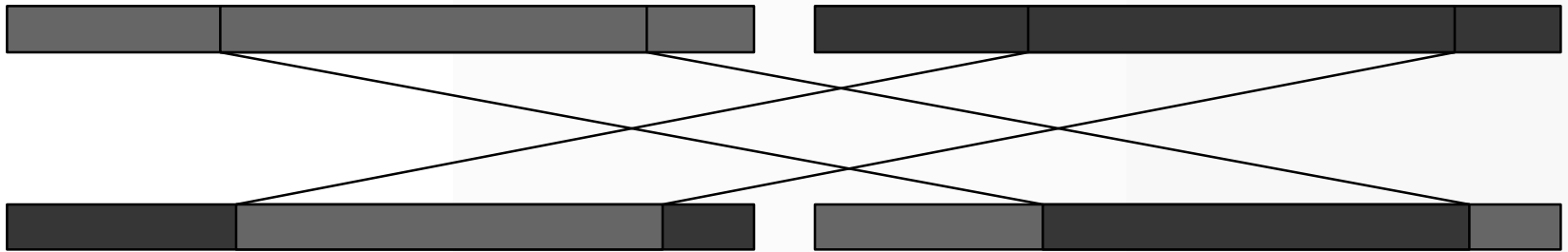
# Genetic Algorithm

```
initialize population p with random genes
repeat
  foreach  $p_i$  in p
     $f_i = \text{fitness}(p_i)$ 
  repeat
     $\text{parent}_1 = \text{select}(p, f)$ 
     $\text{parent}_2 = \text{select}(p, f)$ 
     $\text{child}_1, \text{child}_2 = \text{crossover}(\text{parent}_1, \text{parent}_2)$ 
    if (random < mutate_probability)
       $\text{child}_1 = \text{mutate}(\text{child}_1)$ 
    if (random < mutate_probability)
       $\text{child}_2 = \text{mutate}(\text{child}_2)$ 
    add  $\text{child}_1, \text{child}_2$  to  $p'$ 
  until  $p'$  is full
   $p = p'$ 
```

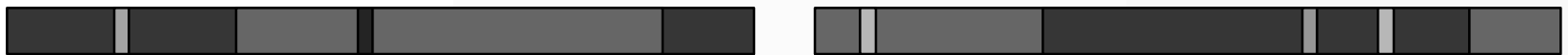
- $\text{Fitness}(\text{gene})$ : the fitness function
- $\text{Select}(\text{population}, \text{fitness})$ : weighted selection of parents
- $\text{Crossover}(\text{gene}, \text{gene})$ : crosses over two genes
- $\text{Mutate}(\text{gene})$ : randomly mutates a gene

# Genetic Operators

- Crossover
  - Select two points at random
  - Swap genes between two points



- Mutate
  - Small probably of randomly changing each part of a gene



# Example: Evaluation

- Initial Population:

- 110 110 10 110 01000

(friendly | neutral) & (low | medium) & (true) & (weaker | same) => Ignore

- 001 010 00 101 00100

(enemy) & (medium) & (weaker | stronger) => Ignore

- 010 001 11 111 10000

(friendly | neutral) & (low | medium) & (true) & (weaker | same) => Heal

- 000 101 01 010 00010

(low | full) & (false) & (same) => Eat

- Evaluation:

- 110 110 10 110 01000: Fitness score = 47

- 010 001 11 111 10000: Fitness score = 23

- 000 101 01 010 00010: Fitness score = 39

- 001 010 00 101 00100: Fitness score = 12

# Example: Genetic Operators

- Crossover:

- 110 110 10 110 01000
- 000 101 01 010 00010

Crossover after bit 7:

- 110 110 1
- 1 010 00010
- 000 101 0
- 0 110 01000

- Mutations

- 110 110 11 011 00010
- 000 101 00 110 01000

- Evaluate the new population

- Repeat

# Advanced Topics

- Competitive evaluation
  - Evaluate each gene against the rest of the population
- Genetic programming
  - Each gene is a chunk of code
  - Generally represented as a parse tree
- Punctuated Equilibria
  - Evolve multiple parallel populations
  - Occasionally swap members
  - Identifies a wider range of high fitness solutions

# Games that use GAs

- Creatures
  - Creatures 2
  - Creatures 3
  - Creatures Adventures
- Seaman
- Nooks & Crannies
- Return Fire II

# Genetic Algorithm Evaluation

- Pros
  - Powerful optimization technique
    - Parallel search of the space
  - Can learn novel solutions
  - No examples required to learn
- Cons
  - Evolution takes lots of processing
    - Not very feasible for online learning
  - Can't guarantee an optimal solution
  - May find uninteresting but high fitness solutions
- Challenges
  - Finding correct representation can be tricky
    - The richer the representation, the bigger the search space
  - Fitness function must be carefully chosen

# References

- Mitchell: Machine Learning, McGraw Hill, 1997.
- Holland: Adaptation in natural and artificial systems, MIT Press 1975.
- Back: Evolutionary algorithms in theory and practice, Oxford University Press 1996.
- Booker, Goldberg, & Holland: Classifier systems and genetic algorithms, Artificial Intelligence 40: 235-282, 1989.
- AI Game Programming Wisdom.
- AI Game Programming Wisdom 2.

# Bayesian Learning

Michael van Lent

# The Big Picture

- Problem
  - Classification
  - Stochastic Modeling
- Feedback
  - Supervised learning
- Knowledge Representation
  - Bayesian classifiers
  - Bayesian Networks
- Knowledge Source
  - Examples

# Background

- Most learning approaches learn a single best guess
  - Learning algorithm selects a single hypothesis
  - Hypothesis = Decision tree, rule set, neural network...
- Probabilistic learning
  - Learn the probability that a hypothesis is correct
  - Identify the most probable hypothesis
  - Competitive with other learning techniques
  - A single example doesn't eliminate any hypothesis
- Notation
  - $P(h)$ : probability that hypothesis  $h$  is correct
  - $P(D)$ : probability of seeing data set  $D$
  - $P(D|h)$ : probability of seeing data set  $D$  given that  $h$  is correct
  - $P(h|D)$ : probability that  $h$  is correct given that  $D$  is seen

# Bayes Rule

- Bayes rule is the foundation of Bayesian learning

$$P(h | D) = \frac{P(D | h)P(h)}{P(D)}$$

- As  $P(D|h)$  increases, so does  $P(h|D)$
- As  $P(h)$  increases, so does  $P(h|D)$
- As  $P(D)$  increases, probability of  $P(h|D)$  decreases

# Example

- A monster has two attacks, A and B:
  - Attack A does 11-20 damage and is used 10% of the time
  - Attack B does 16–115 damage and is used 90% of the time
  - You have counters A' (for attack A) and B' (for attack B)
- If an attack does 16-20 damage, which counter to use?
  - $P(A|\text{damage}=16-20)$  greater or less than 50%?
- We don't know  $P(A|16-20)$ 
  - We do know  $P(A)$ ,  $P(B)$ ,  $P(16-20|A)$ ,  $P(16-20|B)$
  - We only need  $P(16-20)$
  - $P(16-20) = P(A) P(16-20|A) + P(B) P(16-20|B)$

# Example (cont'd)

- Some probabilities
  - $P(A) = 10\%$
  - $P(B) = 90\%$
  - $P(16-20|A) = 50\%$
  - $P(16-20|B) = 5\%$

$$P(A|16-20) = \frac{P(16-20|A)P(A)}{P(16-20)}$$

$$P(A|16-20) = \frac{0.5(0.1)}{(0.1)(0.5) + (0.9)(0.05)}$$

$$P(A|16-20) = \frac{0.05}{0.05 + 0.045} = \frac{0.05}{0.095} = 0.5263 = 52.63\%$$

- So counter A' is the slightly better choice

# Bayes Optimal Classifier

- Given data  $D$ , what's the probability that a new example falls into category  $c$
- $P(\text{example}=c|D)$  or  $P(c|D)$
- Best classification is highest  $P(c|D)$

$$\max_{c_i \in C} P(c_i|D) = \max_{c_i \in C} \sum_{h_j \in H} P(c_i|h_j)P(c_j|D)$$

- This approach tends to be computationally expensive
  - Space of hypothesis is generally very large

# Example Problem

Classify how I should react to an object in the world

- Facts about any given object include:
  - Allegiance = < friendly, neutral, enemy >
  - Health = < low, medium, full >
  - Animate = < true, false >
  - RelativeHealth = < weaker, same, stronger >
- Output categories include:
  - Reaction = Attack
  - Reaction = Ignore
  - Reaction = Heal
  - Reaction = Eat
  - Reaction = Run



- <friendly, low, true, weaker> => Heal
- <neutral, low, true, same> => Heal
- <enemy, low, true, stronger> => Attack
- <enemy, medium, true, weaker> => Attack

# Naïve Bayes Classifier

- Each example is a set of feature values
  - friendly, low, true, weaker
- Given a set of feature values, find the most probable category
- Which is highest:
  - $P(\text{Attack} \mid \text{friendly, low, true, weaker})$
  - $P(\text{Ignore} \mid \text{friendly, low, true, weaker})$
  - $P(\text{Heal} \mid \text{friendly, low, true, weaker})$
  - $P(\text{Eat} \mid \text{friendly, low, true, weaker})$
  - $P(\text{Run} \mid \text{friendly, low, true, weaker})$

$$c_{nb} = \max_{c_i \in C} P(c_i \mid f_1, f_2, f_3, f_4)$$

# Calculating Naïve Bayes Classifier

$$C_{nb} = \max_{c_i \in C} P(c_i | f_1, f_2, f_3, f_4)$$

$$C_{nb} = \max_{c_i \in C} \frac{P(f_1, f_2, f_3, f_4 | c_i)P(c_i)}{P(f_1, f_2, f_3, f_4)}$$

$$C_{nb} = \max_{c_i \in C} P(f_1, f_2, f_3, f_4 | c_i)P(c_i)$$

- Simplifying assumption: each feature in the example is independent
  - Value of Allegiance doesn't affect value of Health, Animate, or RelativeHealth

$$P(f_1, f_2, f_3, f_4 | c_i) = \prod_j P(f_j | c_i)$$

$$C_{nb} = \max_{c_i \in C} P(c_i) \prod_j P(f_j | c_i)$$

# Example

- Slightly modified 13 examples:
  - <friendly, low, true, weaker> => Heal
  - <neutral, full, false, stronger> => Eat
  - <enemy, low, true, weaker> => Eat
  - <enemy, low, true, same> => Attack
  - <neutral, low, true, weaker> => Heal
  - <enemy, medium, true, stronger> => Run
  - <friendly, full, true, same> => Ignore
  - <neutral, full, true, stronger> => Ignore
  - <enemy, full, true, same> => Run
  - <enemy, medium, true, weaker> => Attack
  - <enemy, low, true, weaker> => Ignore
  - <neutral, full, false, stronger> => Ignore
  - <friendly, medium, true, stronger> => Heal
- Estimate the most likely classification of:
  - <enemy, full, true, stronger>

# Example

- Need to calculate:
  - $P(\text{Attack} | \langle \text{enemy}, \text{full}, \text{true}, \text{stronger} \rangle)$   
 $= P(\text{Attack}) P(\text{enemy} | \text{Attack}) P(\text{full} | \text{Attack}) P(\text{true} | \text{Attack}) P(\text{stronger} | \text{Attack})$
  - $P(\text{Ignore} | \langle \text{enemy}, \text{full}, \text{true}, \text{stronger} \rangle)$   
 $= P(\text{Ignore}) P(\text{enemy} | \text{Ignore}) P(\text{full} | \text{Ignore}) P(\text{true} | \text{Ignore}) P(\text{stronger} | \text{Ignore})$
  - $P(\text{Heal} | \langle \text{enemy}, \text{full}, \text{true}, \text{stronger} \rangle)$   
 $= P(\text{Heal}) P(\text{enemy} | \text{Heal}) P(\text{full} | \text{Heal}) P(\text{true} | \text{Heal}) P(\text{stronger} | \text{Heal})$
  - $P(\text{Eat} | \langle \text{enemy}, \text{full}, \text{true}, \text{stronger} \rangle)$   
 $= P(\text{Eat}) P(\text{enemy} | \text{Eat}) P(\text{full} | \text{Eat}) P(\text{true} | \text{Eat}) P(\text{stronger} | \text{Eat})$
  - $P(\text{Run} | \langle \text{enemy}, \text{full}, \text{true}, \text{stronger} \rangle)$   
 $= P(\text{Run}) P(\text{enemy} | \text{Run}) P(\text{full} | \text{Run}) P(\text{true} | \text{Run}) P(\text{stronger} | \text{Run})$

# Example (cont'd)

- $P(\text{Ignore} | \langle \text{enemy, full, true, stronger} \rangle)$   
=  $P(\text{Ignore}) P(\text{enemy} | \text{Ignore}) P(\text{full} | \text{Ignore}) P(\text{true} | \text{Ignore}) P(\text{stronger} | \text{Ignore})$

$P(\text{Ignore}) = 4 \text{ of } 13 \text{ examples} = 4/13 = 31\%$

$P(\text{enemy} | \text{Ignore}) = 1 \text{ of } 4 \text{ examples} = 1/4 = 25\%$

$P(\text{full} | \text{Ignore}) = 3 \text{ of } 4 \text{ examples} = 3/4 = 75\%$

$P(\text{true} | \text{Ignore}) = 3 \text{ of } 4 \text{ examples} = 3/4 = 75\%$

$P(\text{stronger} | \text{Ignore}) = 2 \text{ of } 4 \text{ examples} = 2/4 = 50\%$

$P(\text{Ignore} | \langle \text{enemy, full, true, stronger} \rangle) = 2.2\%$

# Example (cont'd)

- $P(\text{Run} | \langle \text{enemy, full, true, stronger} \rangle)$   
=  $P(\text{Run}) P(\text{enemy} | \text{Run}) P(\text{full} | \text{Run}) P(\text{true} | \text{Run}) P(\text{stronger} | \text{Run})$

$P(\text{Run}) = 2 \text{ of } 13 \text{ examples} = 2/13 = 15\%$

$P(\text{enemy} | \text{Run}) = 2 \text{ of } 2 \text{ examples} = 100\%$

$P(\text{full} | \text{Run}) = 1 \text{ of } 2 \text{ examples} = 50\%$

$P(\text{true} | \text{Run}) = 2 \text{ of } 2 \text{ examples} = 100\%$

$P(\text{stronger} | \text{Run}) = 1 \text{ of } 2 \text{ examples} = 50\%$

$P(\text{Run} | \langle \text{enemy, full, true, stronger} \rangle) = 3.8\%$

# Result

- $P(\text{Ignore} | \langle \text{enemy, full, true, stronger} \rangle) = 2.2\%$
- $P(\text{Run} | \langle \text{enemy, full, true, stronger} \rangle) = 3.8\%$
- $P(\text{Eat} | \langle \text{enemy, full, true, stronger} \rangle) = 0.1\%$
- $P(\text{Heal} | \langle \text{enemy, full, true, stronger} \rangle) = 0\%$
- $P(\text{Attack} | \langle \text{enemy, full, true, stronger} \rangle) = 0\%$
  
- So Naïve Bayes Classification says Run is most probably
  - 63% of Run being correct
  - 36% of Ignore being correct
  - 1% of Eat being correct

# Estimating Probabilities

- Need lots of examples for accurate estimates
- With only 13 examples:
  - No example of:
    - Health=full for Attack category
    - RelativeHealth=Stronger for Attack
    - Allegiance=enemy for Heal
    - Health=full for Heal
  - Only two examples of Run
    - $P(f_1|Run)$  can only be 0%, 50%, or 100%
    - What if the true probability is 16.2%?
- Need to add a factor to probability estimates that:
  - Prevents missing examples from dominating
  - Estimates what might happen with more examples

# m-estimate

- Solution: m-estimate
  - Establish a prior estimate  $p$ 
    - Expert input
    - Assume uniform distribution
  - Estimate the probability as:

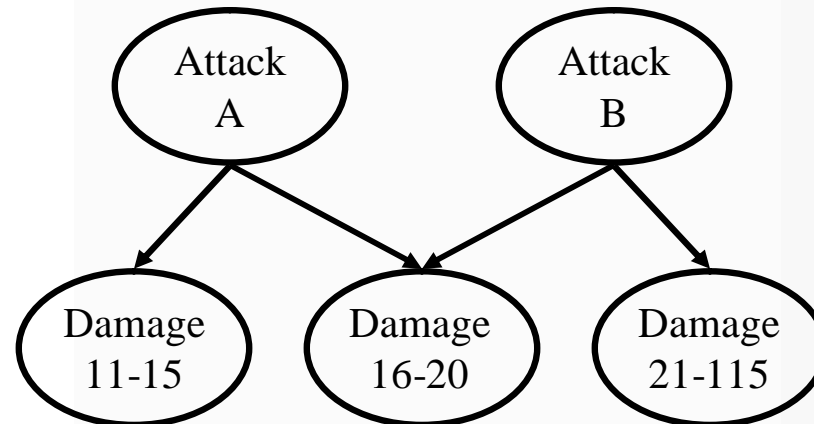
$$\frac{n_c + mp}{n + m}$$

- $m$  is the equivalent sample size
  - Augment  $n$  observed samples with  $m$  virtual samples
- If there are no examples ( $n_c = 0$ ) estimate is still  $> 0\%$
- If  $p(\text{run}) = 20\%$  and  $m = 10$  then  $P(\text{full}|\text{Run})$ :
  - Goes from 50% (1 of 2 examples)
  - to 25%

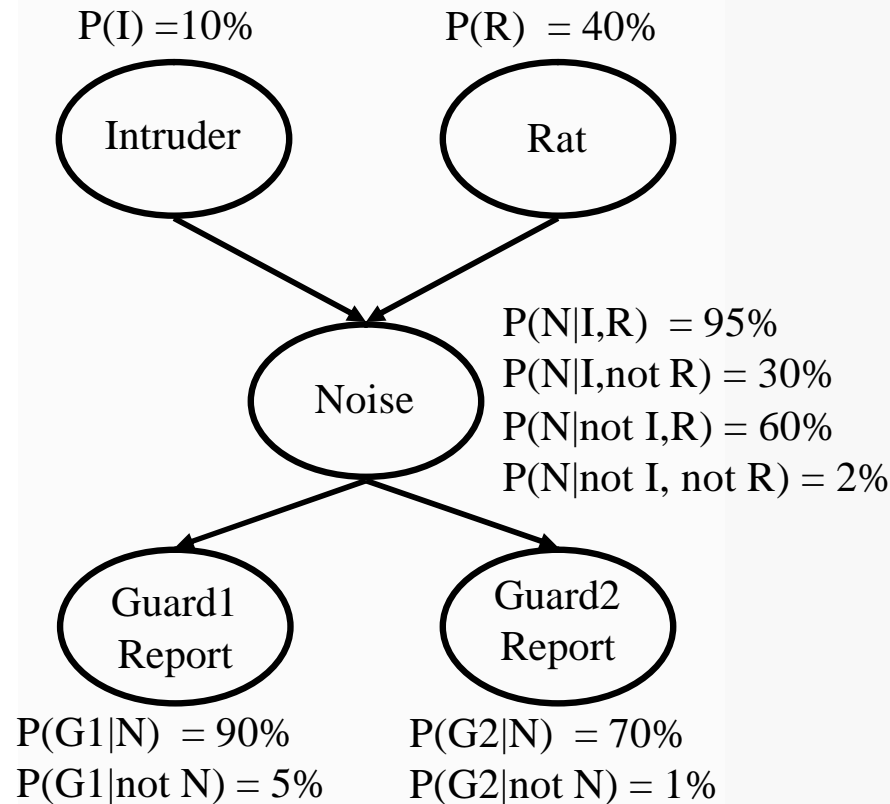
$$\frac{1 + 10(.2)}{2 + 10} = \frac{3}{12} = 0.25$$

# Bayesian Networks

- Graph structure encoding causality between variables
  - Directed, acyclic graph
  - A → B indicates that A directly influences B
    - Positive or negative influence



# Another Bayesian Network



- Inference on Bayesian Networks can determine probability of unknown nodes (Intruder) given some known values
  - If Guard2 reports but Guard 1 doesn't, what's the probability of Intruder?

# Learning Bayesian Networks

- Learning the topology of Bayesian networks
  - Search the space of network topologies
    - Adding arcs, deleting arcs, reversing arcs
    - Are independent nodes in the network independent in the data?
    - Does the network explain the data?
    - Need to weight towards fewer arcs
- Learning the probabilities of Bayesian networks
  - Experts are good at constructing networks
  - Experts aren't as good at filling in probabilities
  - Expectation Management (EM) algorithm
  - Gibbs Sampling

# Bayesian Learning Evaluation

- Pros
  - Takes advantage of prior knowledge
  - Probabilistic predictions (prediction confidence)
  - Handles noise well
  - Incremental learning
- Cons
  - Less effective with low number of examples
  - Can be computationally expensive
- Challenges
  - Identifying the right features
  - Getting a large number of good examples

# References

- Mitchell: Machine Learning, McGraw Hill, 1997.
- Russell and Norvig: Artificial Intelligence: A Modern Approach, Prentice Hall, 1995.
- AI Game Programming Wisdom.

# Reinforcement Learning

John Laird

Thanks for online reference material to: Satinder Singh, Yijue Hou & Patrick Doyle

# Outline of Reinforcement Learning

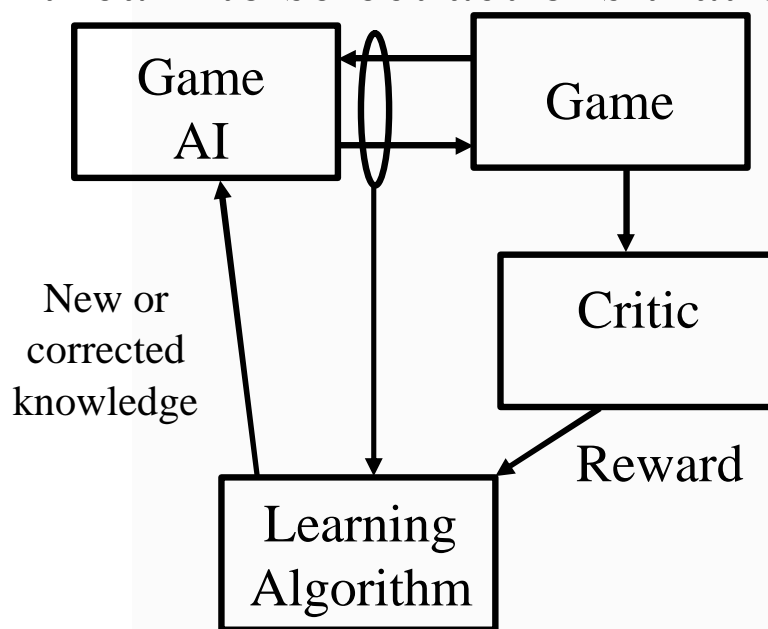
- What is it?
- When is it useful?
- Examples from games
- Analysis

# Reinforcement Learning

- A set of problems, not a single technique:
  - Adaptive Dynamic Programming
  - Temporal Difference Learning
  - Q learning
- Cover story for Neural Networks, Decision Trees, etc.
- Best for tuning behaviors
  - Often requires many training trials to converge
- Very general technique applicable to many problems
  - Backgammon, poker, helicopter flying, truck & car driving

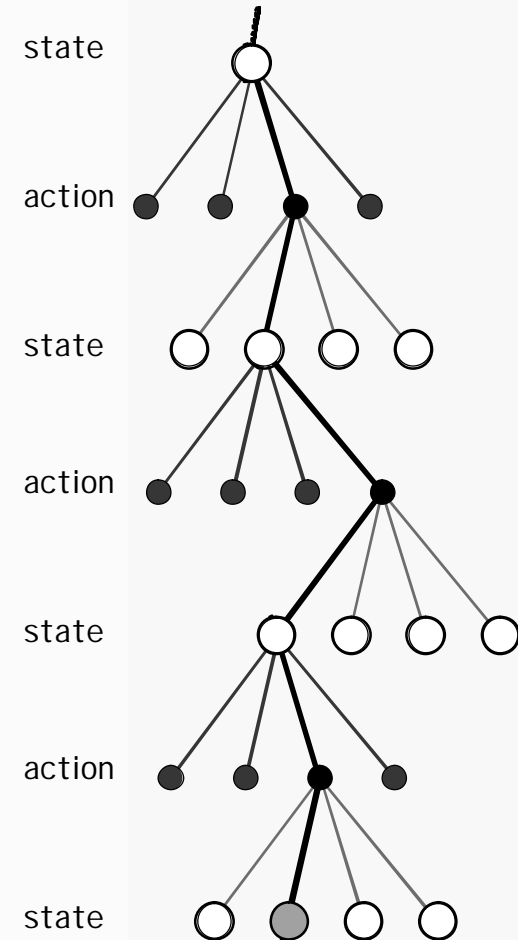
# Reinforcement Learning

- Agent receives some reward/punishment for behavior
  - Is not told directly what to do or what not to do
    - Only whether it has done well or poorly
  - Reward can be intermittent and is often delayed
  - Must solve the temporal credit assignment problem
  - How can it learn to select actions that occur before reward?



# Deathmatch Example

- Learn to kill enemy better
- Possible rewards for Halo
  - +10 kill enemy
  - -3 killed
- State features
  - Health, enemy health
  - Weapon, enemy weapon
  - Relative position and facing of enemy
  - Absolute and relative speeds
  - Relative positions of nearby obstacles



# Two Approaches to Reinforcement Learning

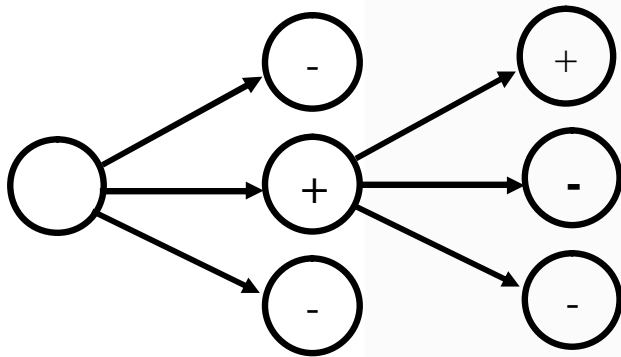
- Passive learning = behavior cloning
  - Examples of behavior are presented to learner
    - Learn a model of a human player
  - Tries to learn a single optimal policy
- Active learning = learning from experience
  - Agent is trying to perform task and learn at same time
  - Must trade off exploration vs. exploitation
  - Can train using against self or humans

# What can be Learned?

- Utility Function:
  - How good is a state?
  - The utility of state  $s_i$ :  $U(s_i)$
  - Choose action to maximize expected utility of result
- Action-Value:
  - How good is a given action for a given state?
  - The expected utility of performing action  $a_j$  in state  $s_i$ :  $V(s_i, a_j)$
  - Choose action with best expected utility for current state

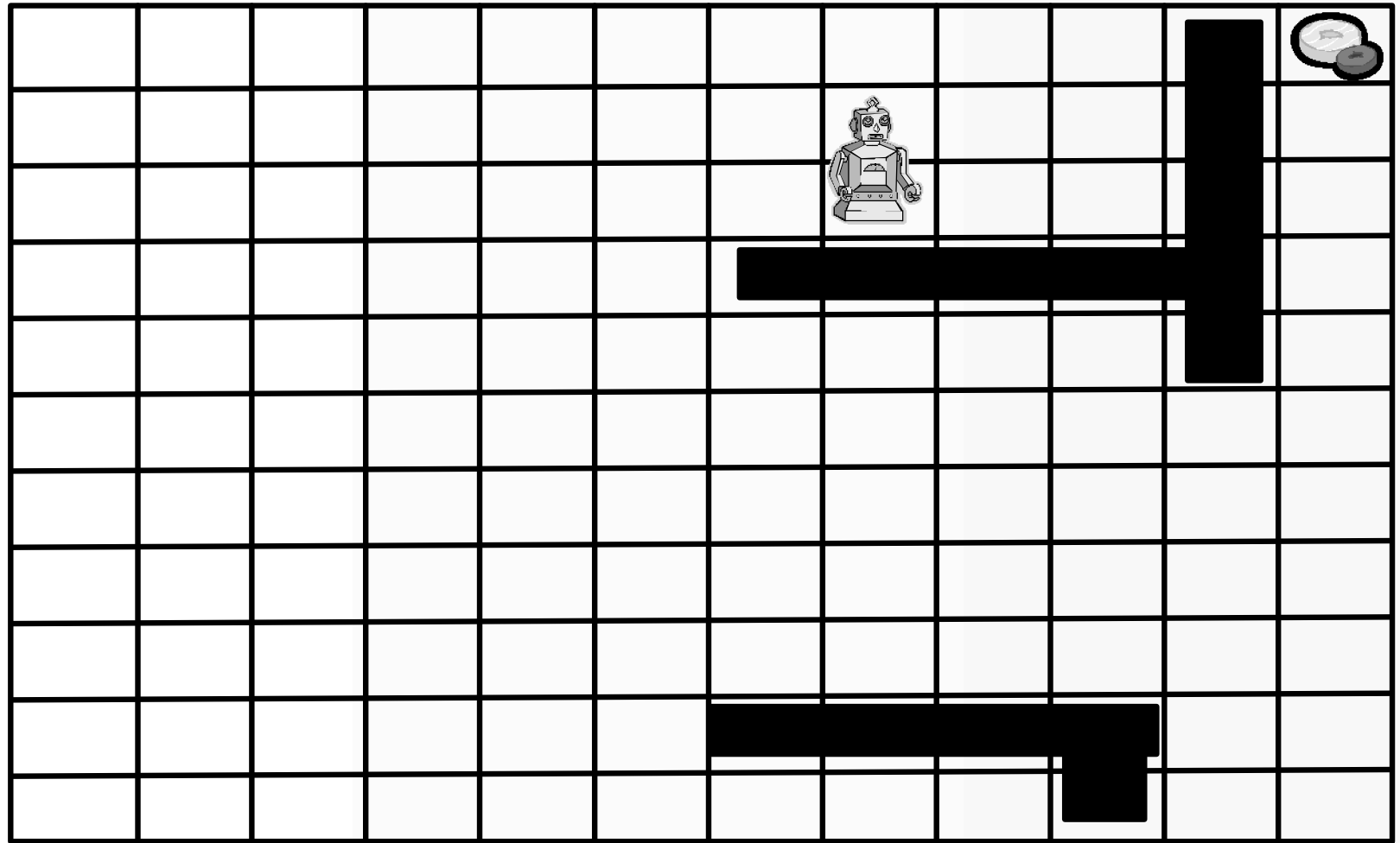
# Utility Function for States: $U(s_i)$

- Agent chooses action to maximize *expected* utility
  - One step look-ahead



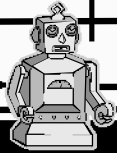





- Agent must have a “model” of environment
  - Possible transitions from state to state
  - Can be learned or preprogrammed

# Trivial Example: Maze Learning

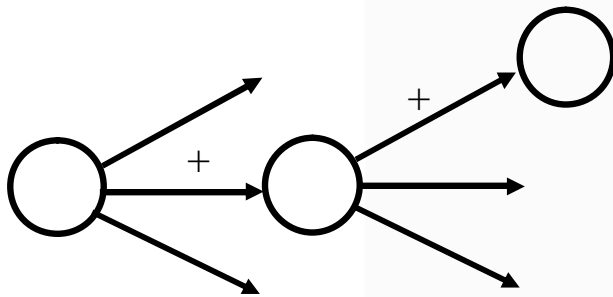


# Learning State Utility Function: $U(s_i)$

					.84	.83	.82	.81	.80		
				.84	.85	.84	.83	.82	.81		.99
				.85	.86	.85		.83	.82		.98
				.86	.87						.97
				.87	.88	.89	.90	.91	.92		.96
				.88	.89	.90	.91	.92	.93	.94	.95
				.87	.88	.89	.90	.91	.92	.93	.94
				.86	.87	.88	.89	.90	.91	.92	.93
				.85	.86	.87	.88	.89	.90	.91	.92
					.85					.90	.91
										.89	.90

# Action Value Function: $V(s_i, a_j)$

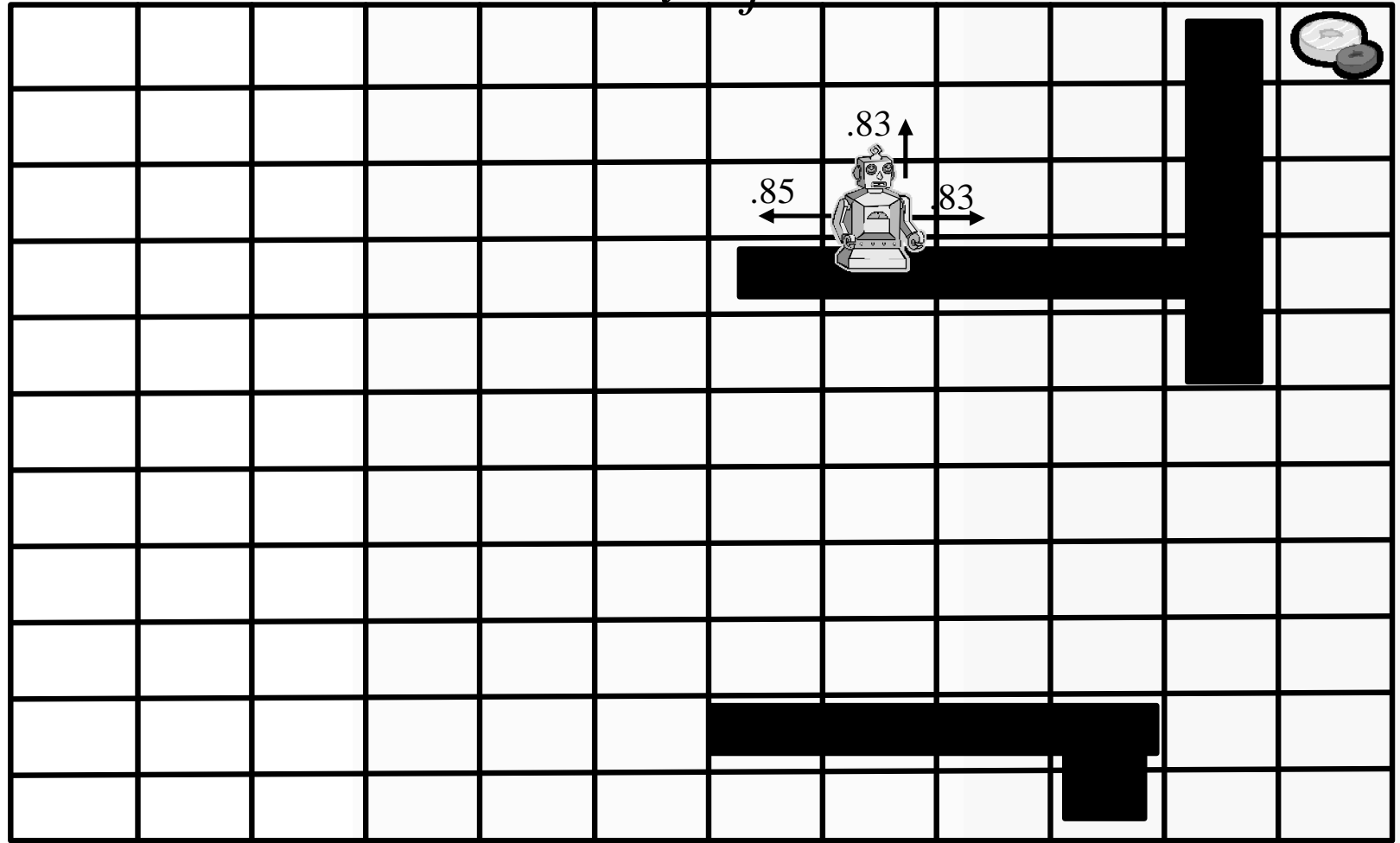
- Agent chooses action that is best for current state
  - Just compare operators – not state



- Agent doesn't need a “model” of environment
  - But must learn separate value for each state-action pair

# Learning Action-Value Function:

$$V(s_i, a_j)$$



# Review of Dimensions

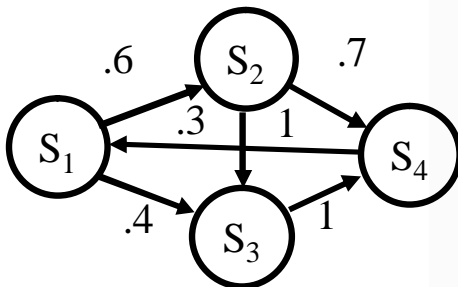
- Source of learning data
  - Passive
  - Active
- What is learned
  - State utility function
  - Action-value

# Passive Utility Function Approaches

- Least Mean Squares (LMS)
- **Adaptive Dynamic Programming (ADP)**
  - Requires a model (M) for learning
- **Temporal Difference Learning (TDL)**
  - Model free learning (uses model for decision making, but not for learning).

# Learning State Utility Function (U)

- Assume  $k$  states in the world
- Agent keeps:
  - An estimate  $U$  of the utility of each state ( $k$ )
  - A table  $N$  of how many times each state was seen ( $k$ )
  - A table  $M$  (the model) of the transition probabilities ( $k \times k$ )
    - likelihood of moving from each state to another state

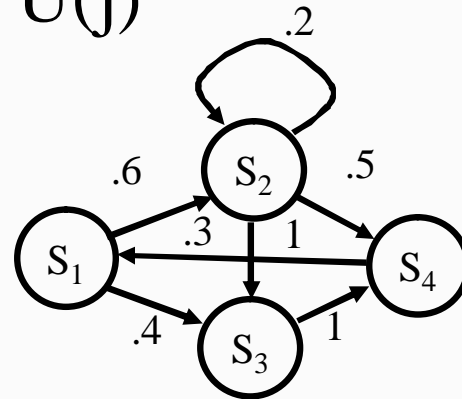


	S1	S2	S3	S4
S1	/	.6	.4	0
S2	0	/	.3	.7
S3	0	0	/	1
S4	1	0	0	/

# Adaptive Dynamic Programming (ADP)

- Utility = reward and probability of future reward
- $U(i) = R(i) + \sum_j M_{ij} * U(j)$

	S1	S2	S3	S4
S1	/	.6	.4	0
S2	0	.2	.3	.5
S3	0	0	/	1
S4	1	0	0	/



Initial Utilities:

S1=.5

S2=.6

S3=.2

S4=.1

State S2 and get reward .3

$$\begin{aligned}
 U(3) &= .3 + 0 * .5 + .2 * .6 + .3 * .2 + .5 * .1 \\
 &= .3 + 0 + .12 + .06 + .05 \\
 &= .53
 \end{aligned}$$

Exact, but inefficient in large search spaces

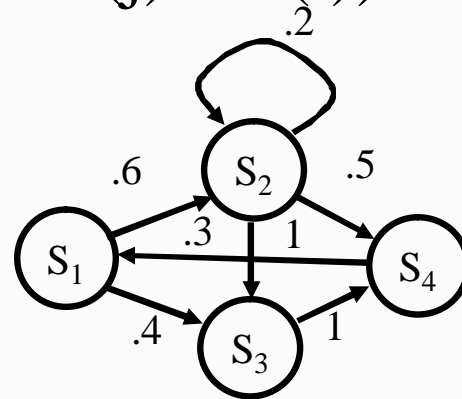
Requires sweeping through complete space

# Temporal Difference Learning

- Approximate ADP
  - Adjust the estimated utility value of the current state based on its immediate reward and the *estimated* value of the next state.
- $U(i) = U(i) + a(R(i) + U(j) - U(i))$ 
  - $a$  is learning rate
  - if  $a$  continually decreases,  $U$  will converge

# Temporal Difference Example

- Utility = reward and probability of future reward
- $U(i) = U(i) + a(R(i) + U(j) - U(i))$



Initial Utilities:

S1=.5

S2=.6

S3=.2

S4=.1

State S2, get reward .3, go to state S3

$$\begin{aligned}U(3) &= .6 + .5 * (.3 + .2 - .6) \\ &= .6 + .5 * (-.1) \\ &= .6 + -.05 \\ &= .55\end{aligned}$$

# TD vs. ADP

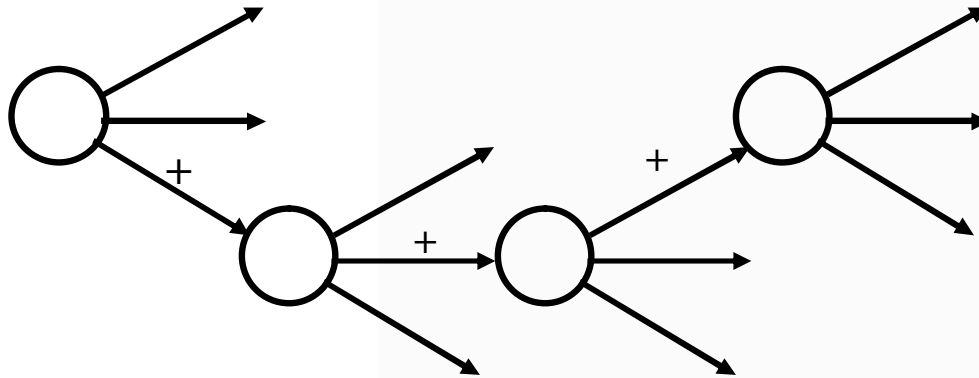
- ADP learns faster
- ADP is less variable
- TD is simpler
- TD has less computation/observation
- TD does not require a model during learning
- TD biases update to observed successor instead of all

# *Active Learning State Utilities: ADP*

- Active learning must decide which action to take and update based on what it does.
- Extend model  $M$  to give the probability of a transition from a state  $i$  to a state  $j$ , *given an action  $a$* .
- Utility is maximum of
- $U(i) = R(i) + \max_a [\text{SUM}_j M_{ij}^a U(j) ]$

# Active Learning State-Action Functions (Q-Learning)

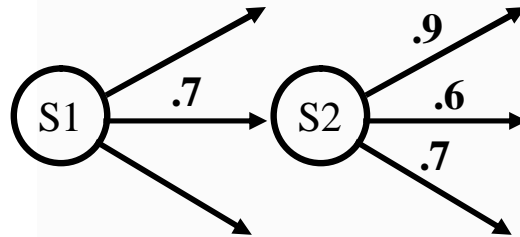
- Combines situation and action:



- $Q(a,i)$  = expected utility of using action  $a$  on state  $i$ .
- $U(i) = \max_a Q(a, i)$

# Q Learning

- ADP Version:  $Q(a, i) = R(i) + \gamma \sum_j M_{ij}^a \max_{a'} Q(a', j)$



- TD:  $Q(a, i) \leftarrow Q(a, i) + \alpha (R(i) + \gamma (\max_{a'} Q(a', j) - Q(a, i)))$ 
  - If  $\alpha$  is .1,  $\gamma$  is .9, and  $R(1) = 0$ ,
  - $= .7 + .1 * (0 + .9 * (\max(.6, .7, .9) - .7))$   
 $= .7 + .1 * .9 * (.9 - .7)$   
 $= .7 + .18$   
 $= .718$
- Selection is biased by expected utilities
  - Balances exploration vs. exploitation
  - With experience, bias more toward higher values

# Q-Learning

- Q-Learning is the first provably convergent *direct adaptive optimal control algorithm*
- Great impact on the field of modern RL
  - smaller representation than models
  - automatically focuses attention to where it is needed, i.e., no sweeps through state space

# Q Learning Algorithm

For each pair  $(a, s)$ , initialize  $Q(a, s)$

Observe the current state  $s$

Loop forever

{

Select an action  $a$  and execute it

$$a = \arg \max_a Q(s, a)$$

Receive immediate reward  $r$  and observe the new state  $s'$

Update  $Q(a, s)$

$$Q(s, a) = Q(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

$s = s'$

}

# Summary Comparison

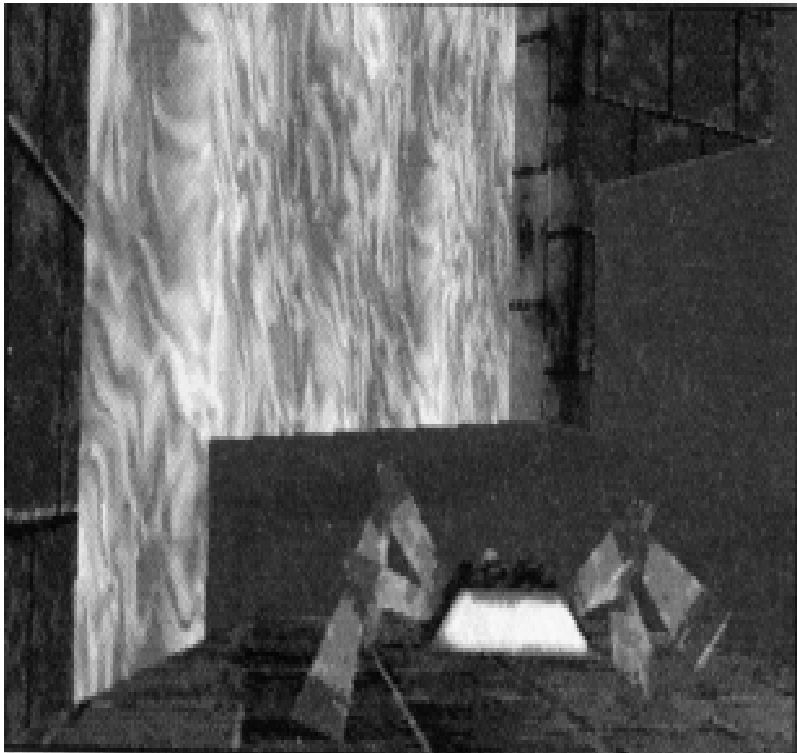
## State Utility Function

- Requires model
- More general/faster learning
  - Learns about states
- Slower execution
  - Must compute follow on states
- If have model of reward, doesn't need environment
- Useful for worlds with model
  - Maze worlds, board games, ...

## State-Action

- Model free
- Less general/slower learning
  - Must learn state-action combinations
- Faster execution
- Preferred for complex worlds where model isn't available

# Anark Software Galapagos



- Player trains creature by manipulating environment
- Creature learns from pain, death, and reward for movement
- Learns to move and classify objects in world based on their pain/death.

# Challenges

- Exploring the possibilities
- Picking the right representation
- Large state spaces
- Infrequent reward
- Inter-dependence of actions
- Complex data structures
- Dynamic worlds
- Setting parameters

# Exploration vs. Exploitation

- Problem: If large space of possible actions, might never experience many of them if learn too quick.
- Exploration: try out actions
- Exploitation: use knowledge to improve behavior
- Compromise:
  - Random selection, but bias choice to best actions
  - Overtime, bias more and more to best actions

# Picking the Right Representation

- Too few features and impossible to learn
  - If learning to drive and can sense acceleration or speed.
- Too many features and can use exact representations
  - See next section

# Large state spaces: Curse of Dimensionality

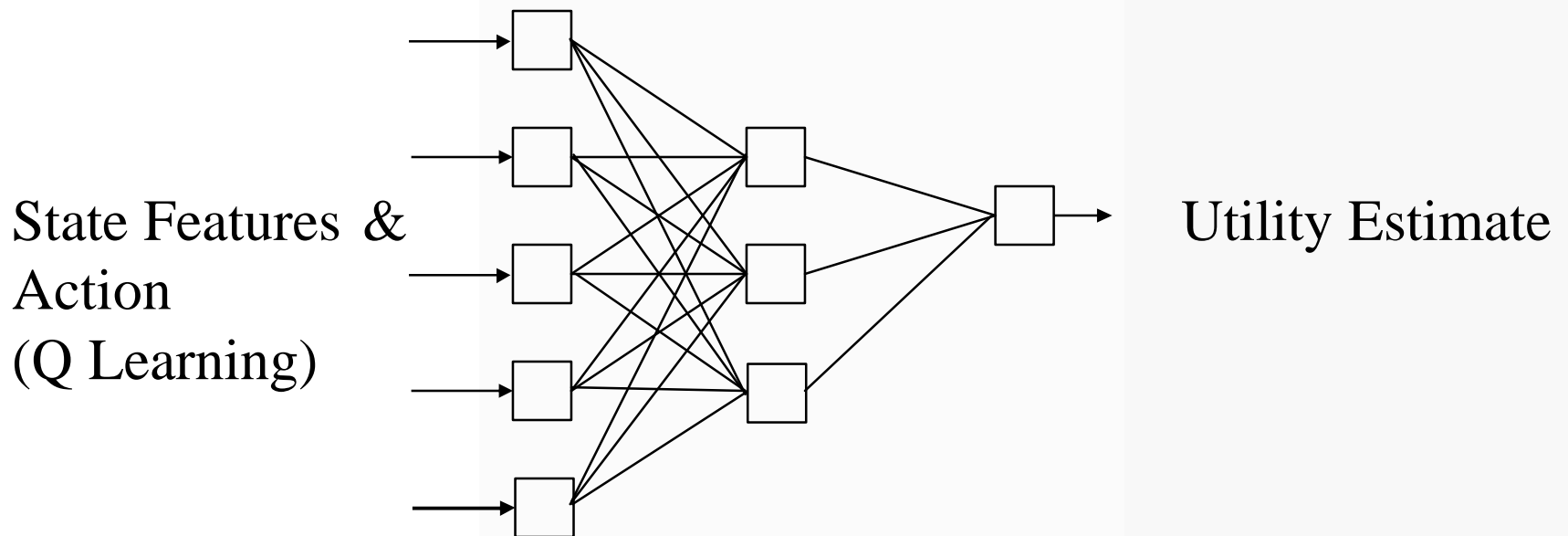
- Look-up Table for Q value
  - AIW 2, pp. 597
  - OK for 2-3 variables
  - Fast learning, but lots of memory
- Issues:
  - Hard to get data that covers the states enough time to learn accurate utility functions
  - Probably many different states have similar utility
  - Data structures for storing utility functions can be very large
  - State-action approaches (Q-learning) exacerbate the problem
- Deathmatch example:
  - Health [10], Enemy Health [10], Relative Distance [10], Relative Heading [10], Relative Opponent Heading [10], Weapon [5], Ammo [10], Power ups [4], Enemy Power ups [4], My Speed [4], His Speed [4], Distances to Walls [5,5,5,5]
  - $8 \times 10^{14}$

# Solution:

- Approximate state space with some function
- Neural Networks, Decision Tree, Nearest Neighbor, Bayesian Network, ...
  - Can be slower than lookup table but much more compact

# Function Approximation: Neural Networks

- Use all features as input with utility as output



- Output could be actions and their utilities?

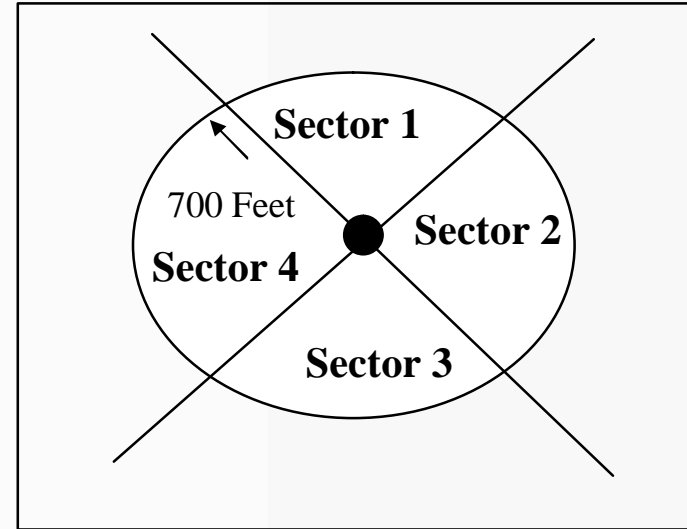
# Geisler – FPS Offline Learning

## Input Features:

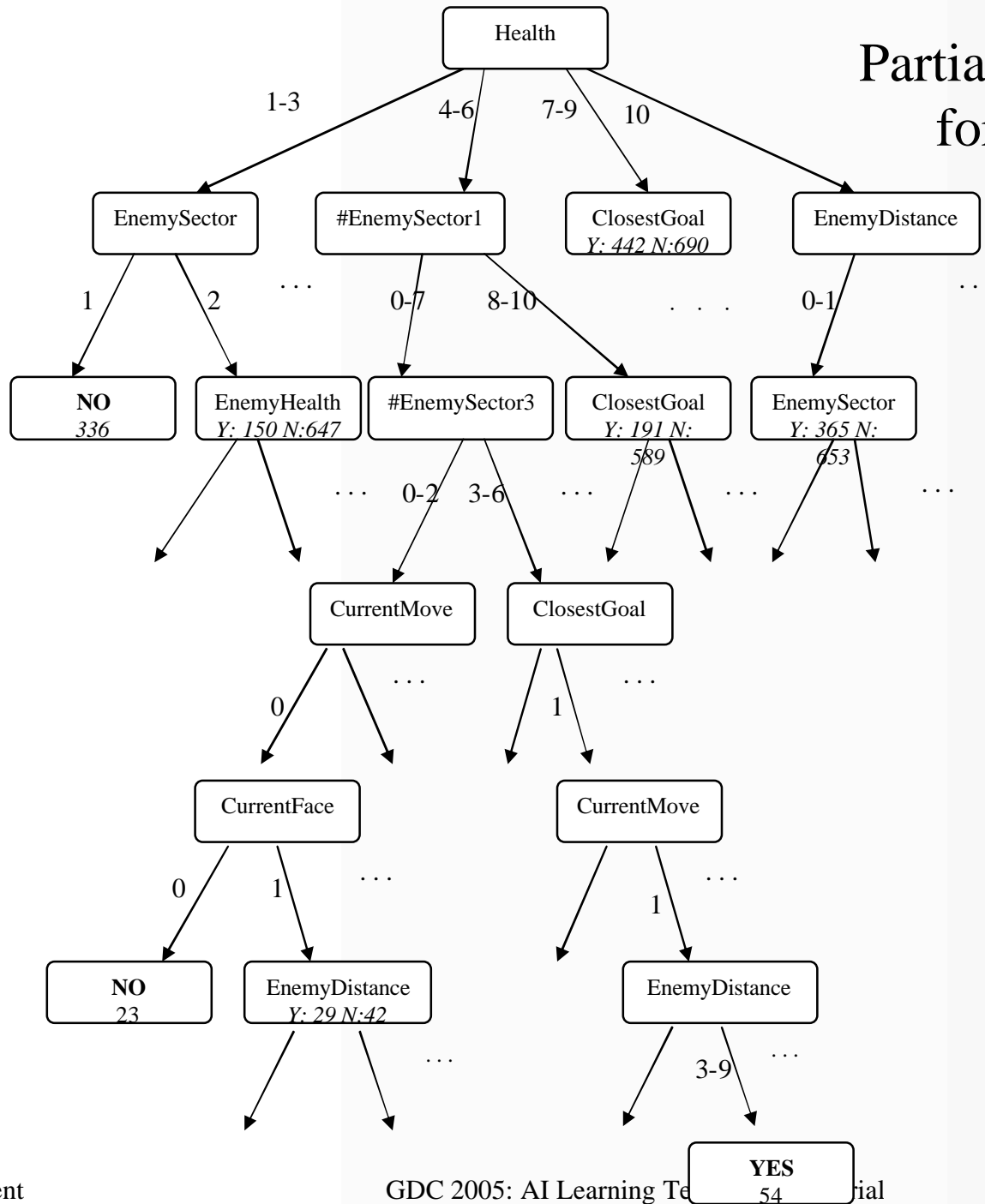
- *Closest Enemy Health*
- *Number Enemies in Sector 1*
- *Number Enemies in Sector 2*
- *Number Enemies in Sector 3*
- *Number Enemies in Sector 4*
- *Player Health*
- *Closest Goal Distance*
- *Closest Goal Sector*
- *Closest Enemy Sector*
- *Distance to Closest Enemy*
- *Current Move Direction*
- *Current Face Direction*

## Output

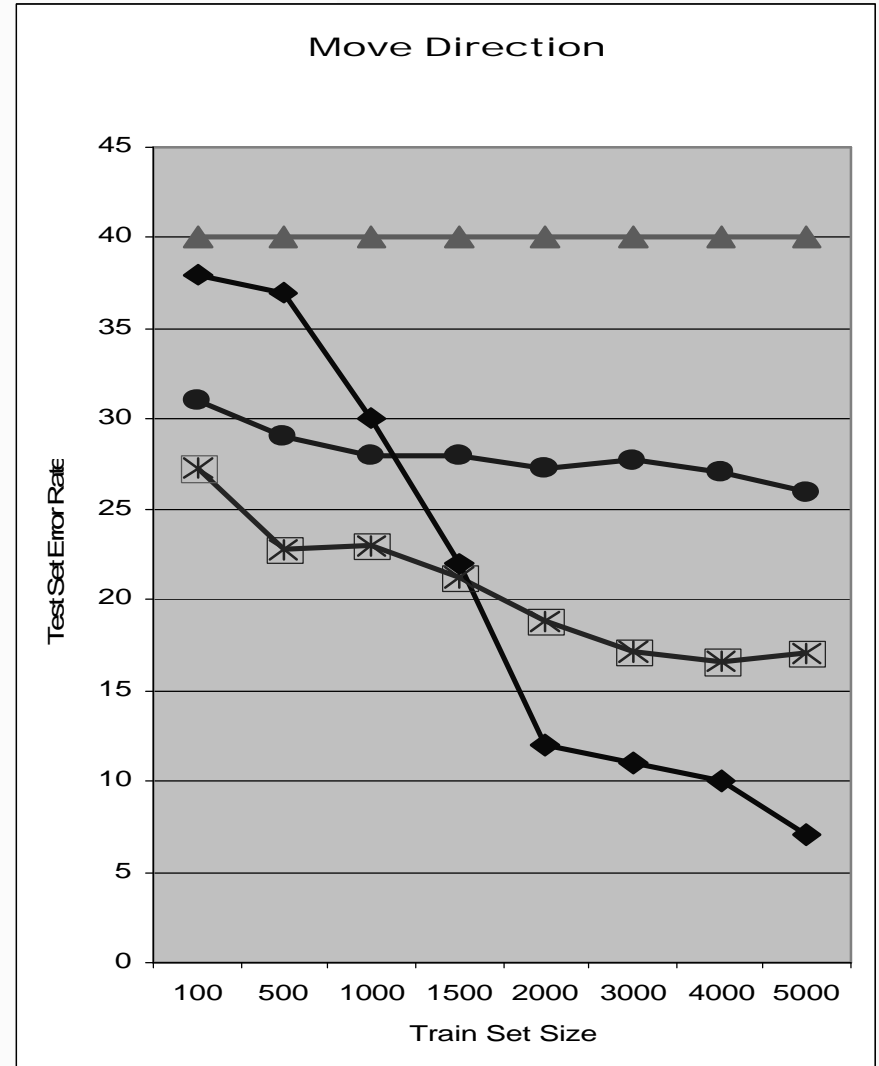
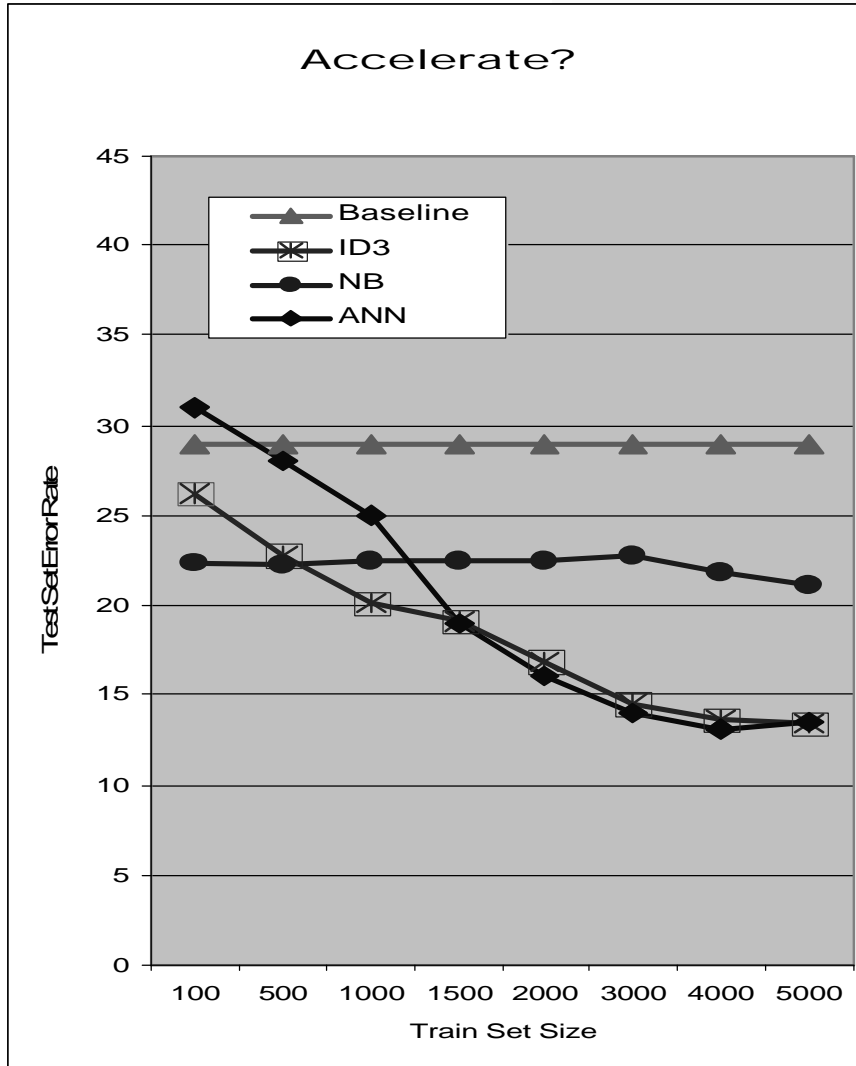
- *Accelerate*
  - *Move Direction*
  - *Facing Direction*
  - *Jumping*
- 
- Tested with Neural Networks, Decision Trees, and Naïve Bayes



# Partial Decision Tree for Accelerate



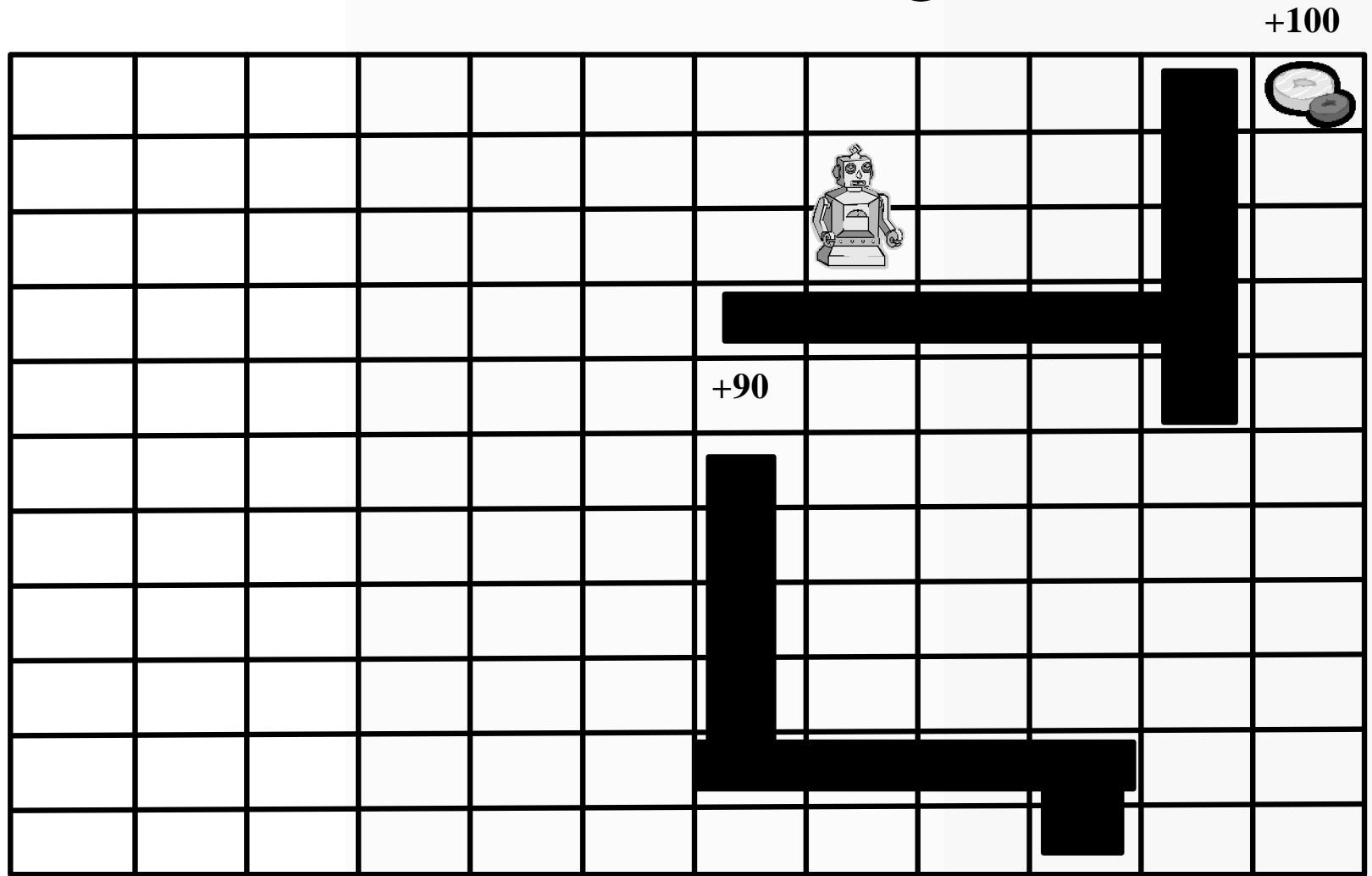
# Results – Error Rates



# Infrequent reward

- **Problem:**
  - If feedback comes only at end of lots of actions, hard to learn utilities of early situations
- **Solution**
  - Provide intermediate rewards
- **Example: FPS**
  - +1 for hitting enemy in FPS deathmatch
  - -1 for getting hit by enemy
  - +.5 for getting behind enemy
  - +.4 for being in place with good visibility but little exposure
- **Risks**
  - Achieving intermediate rewards instead of final reward

# Maze Learning





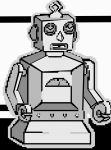
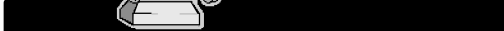


# Many Related Actions

- If try to learn all at once, very slow
- Train up one at a time:
  - 10.4 in AIW2, p.596

# Dynamic world

- Problem:
  - If world or reward changes suddenly, system can't respond
- Solution:
  1. Continual exploration to detect changes
  2. If major changes, restart learning

# Major Change in World

					.84	.83	.82	.81	.80		
				.84	.85	.84	.83	.82	.81		.99
				.85	.86	.85		.83	.82		.98
				.86	.87						.97
				.87	.88	.89	.90	.91	.92		.96
				.88	.89	.90	.91	.92	.93	.94	.95
				.87	.88	.89	.90	.91	.92	.93	.94
				.86	.87	.88	.89	.90	.91	.92	.93
				.85	.86	.87	.88	.89	.90	.91	.92
					.85					.90	.91
										.89	.90

# Setting Parameters

- Learning Rate:  $\alpha$ 
  - If too high, might not converge (skip over solution)
  - If too low, can converge slowly
  - Lower with time:  $k^n$  such as  $.95^n = .95, .9, .86, .81, .7$
  - For deterministic worlds and state transitions, .1-.2 works well
- Discount Factor:  $\gamma$ 
  - Affects how “greedy” agent is for short term vs. long-term reward
  - .9-.95 is good for larger problems
- Best Action Selection Probability:  $\epsilon$ 
  - Increases as game progresses so takes advantage of learning
  - $1 - k^n$

# Analysis

- Advantages:
  - Excellent for tuning parameters & control problems
  - Can handle noise
  - Can balance exploration vs. exploitation
- Disadvantages
  - Can be slow if large space of possible representations
  - Has troubles with changing concepts
- Challenges:
  - Choosing the right approach: utility vs. action-value
  - Choosing the right features
  - Choosing the right function approximation (NN, DT, ...)
  - Choosing the right learning parameters
  - Choosing the right reward function

# References

- John Manslow: Using Reinforcement Learning to Solve AI Control Problems: AI Programming Wisdom 2, p. 591
- Benjamin Geisler, An Empirical Study of Machine Learning Algorithms Applied to Modeling Player Behavior in a “First Person Shooter” Video Game, Masters’ Thesis, U. Wisconsin, 2002.

# Episodic Learning

## [Andrew Nuxoll]



- What is it?
  - Not facts or procedures but memories of specific events
  - Recording and recalling of experiences with the world
- Why study it?
  - No comprehensive computational models of episodic learning
  - No cognitive architectural models of episodic learning
    - If not architectural, interferes with other reasoning
  - Episodic learning will expand cognitive abilities
    - Personal history and identity
    - Memories that can be used for future decision making & learning
    - Necessary for reflection, debriefing, etc.
    - Without it we are trying to build crippled AI systems
  - Mother of all case-based reasoning problems.

# Characteristics of Episodic Memory

## 1. Architectural:

- The mechanism is used for all tasks and does not compete with reasoning.

## 2. Automatic:

- Memories are created without effort or deliberate rehearsal.

## 3. Autonoetic:

- A retrieved memory is distinguished from current sensing.

## 4. Autobiographical:

- The episode is remembered from own perspective.

## 5. *Variable Duration:*

- *The time period spanned by a memory is not fixed.*

## 6. *Temporally Indexed:*

- *The rememberer has a sense of the time when the episode occurred.*

# Advantages of Episodic Memory

- Improves AI behavior
  - Creates a personal history that impacts behavior
    - Knows what it has done – avoid repetition
  - Helps identify significant changes to the world
    - Compare current situation to memory
  - Creates virtual sensors of previously seen aspects of the world
  - Helps explaining behavior
    - History of goals and subgoals it attempted
  - Provide the basis of a simple model of the environment
  - Supports other learning mechanisms

# Why and why not Episodic Memory?

- Advantages:
  - General capability that can be reused on many projects.
  - Might be difficult to identify what to store.
- Disadvantages:
  - Can be replaced with code customized for specific needs.
  - Might be costly in memory and retrieval.

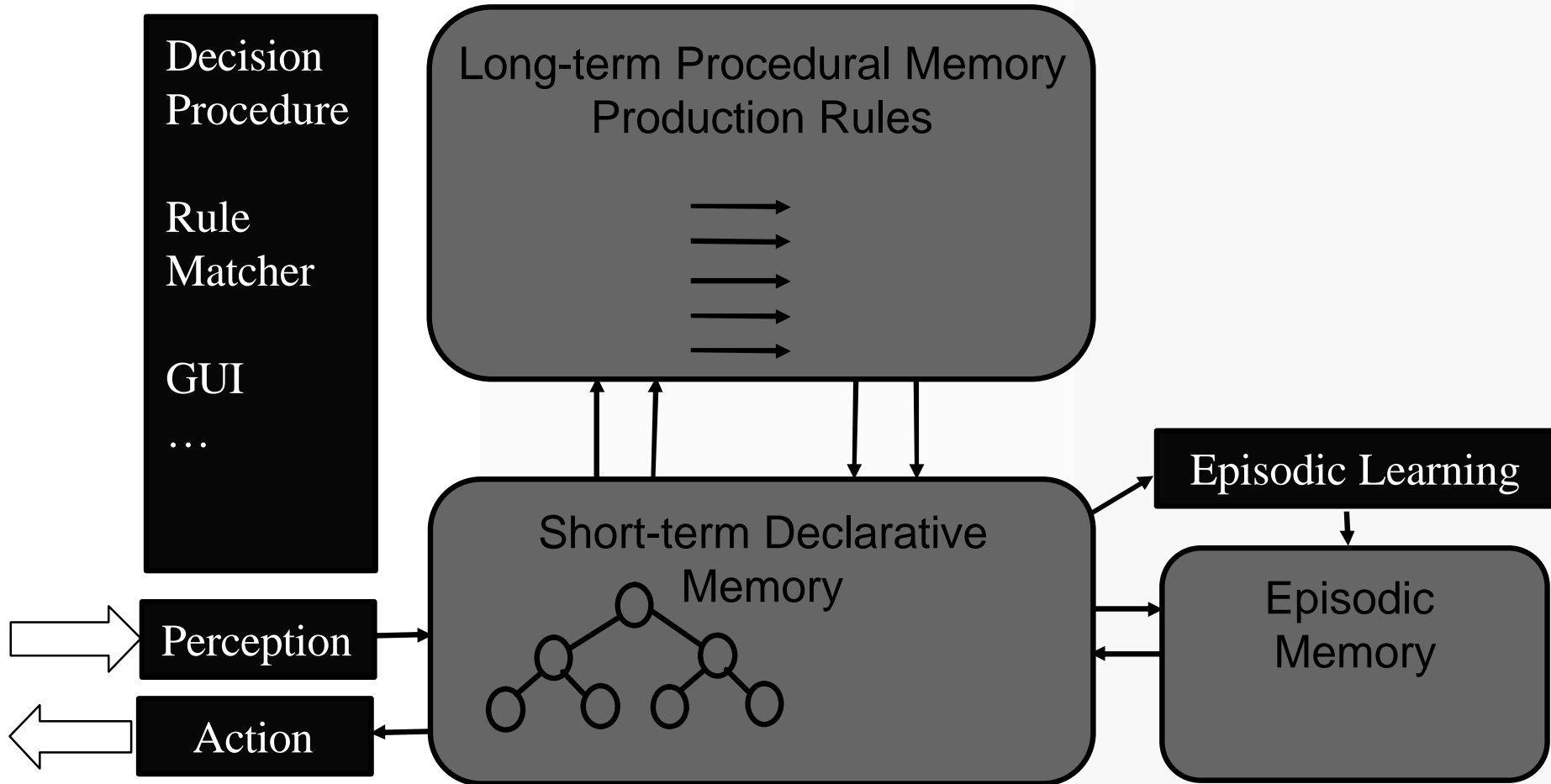
# Implementing Episodic Memory

- Encoding
  - When is an episode stored?
  - What is stored and what is available for cuing retrieval?
- Storage
  - How is it stored for efficient insertion and query?
- Retrieval
  - What is used to cue the retrieval?
  - How is the retrieval efficiently performed?
  - What is retrieved?

# Possible Approach

- When encode:
  - Every encounter between a NPC and the player
  - If NPC goal/subgoal is achieved
- What to store:
  - Where, when, what other entities around, difficulty of achievement, objects that were used, ...
  - Pointer to next episode
- Retrieve based on:
  - Time, goal, objects, place
- Can create efficient hash or tree-based retrieval.

# Soar Structure

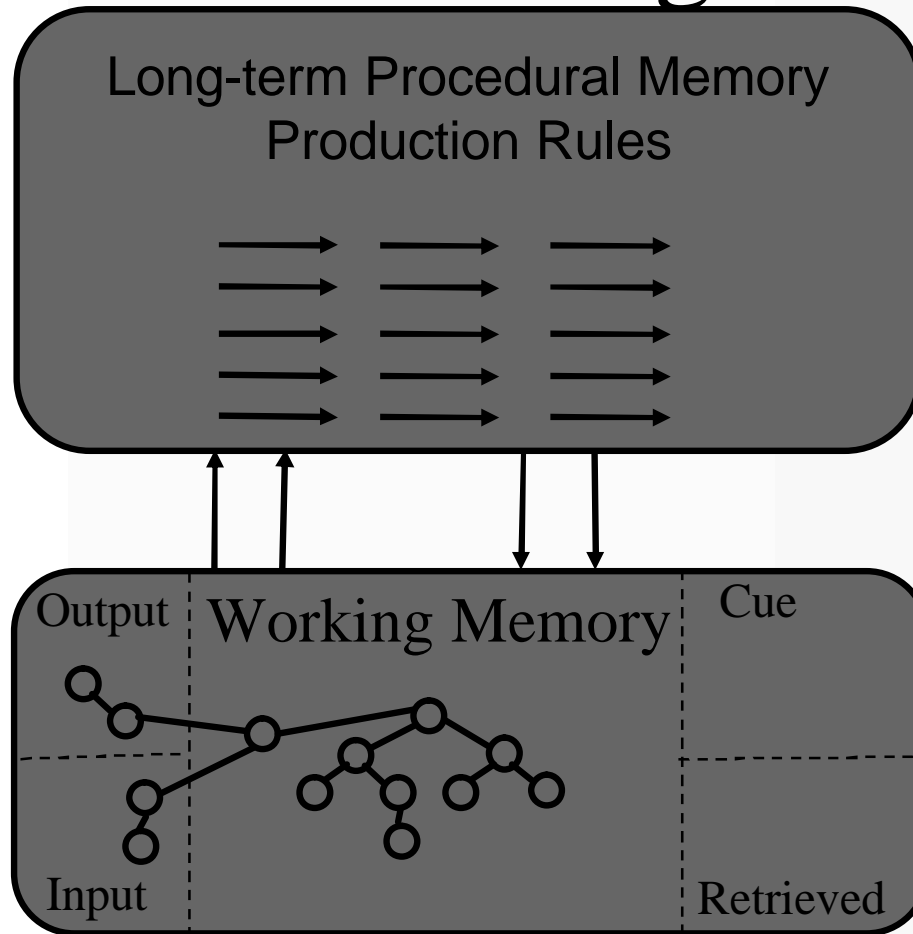


# Implementation Big Picture

Encoding  
Initiation?

Storage

Retrieval



When the agent takes an action.

# Implementation Big Picture

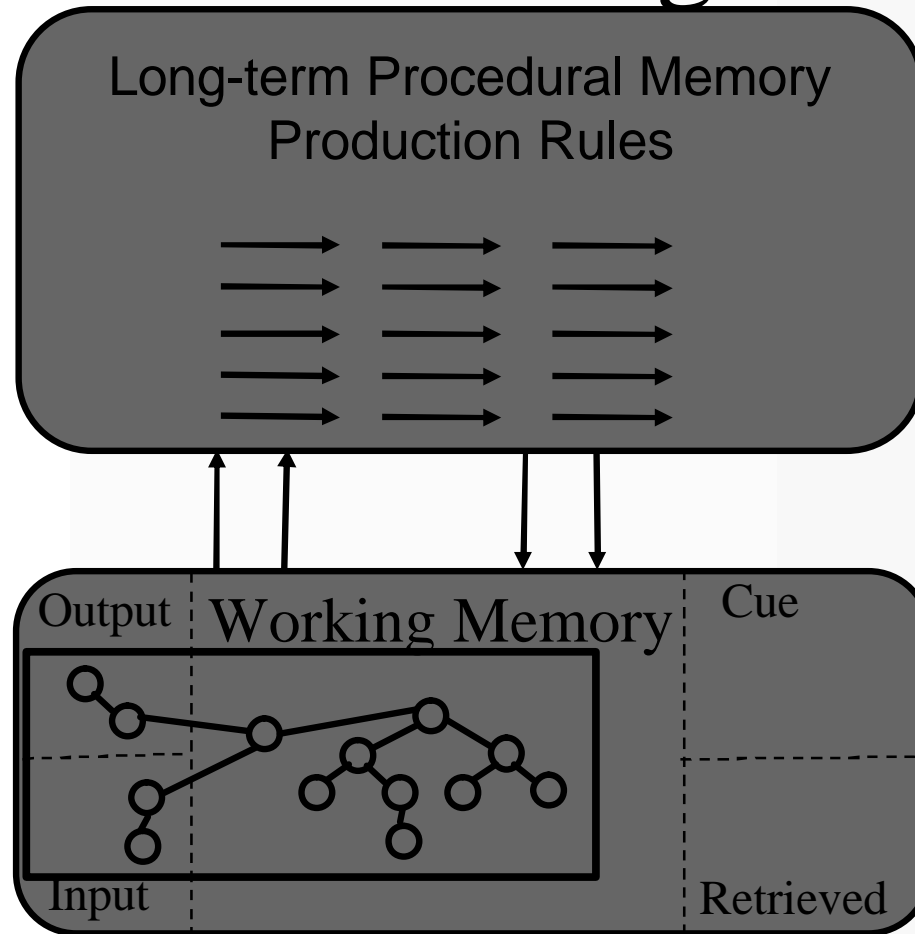
Encoding

Initiation

Content?

Storage

Retrieval



The entire working memory is stored in the episode

# Implementation Big Picture

Encoding

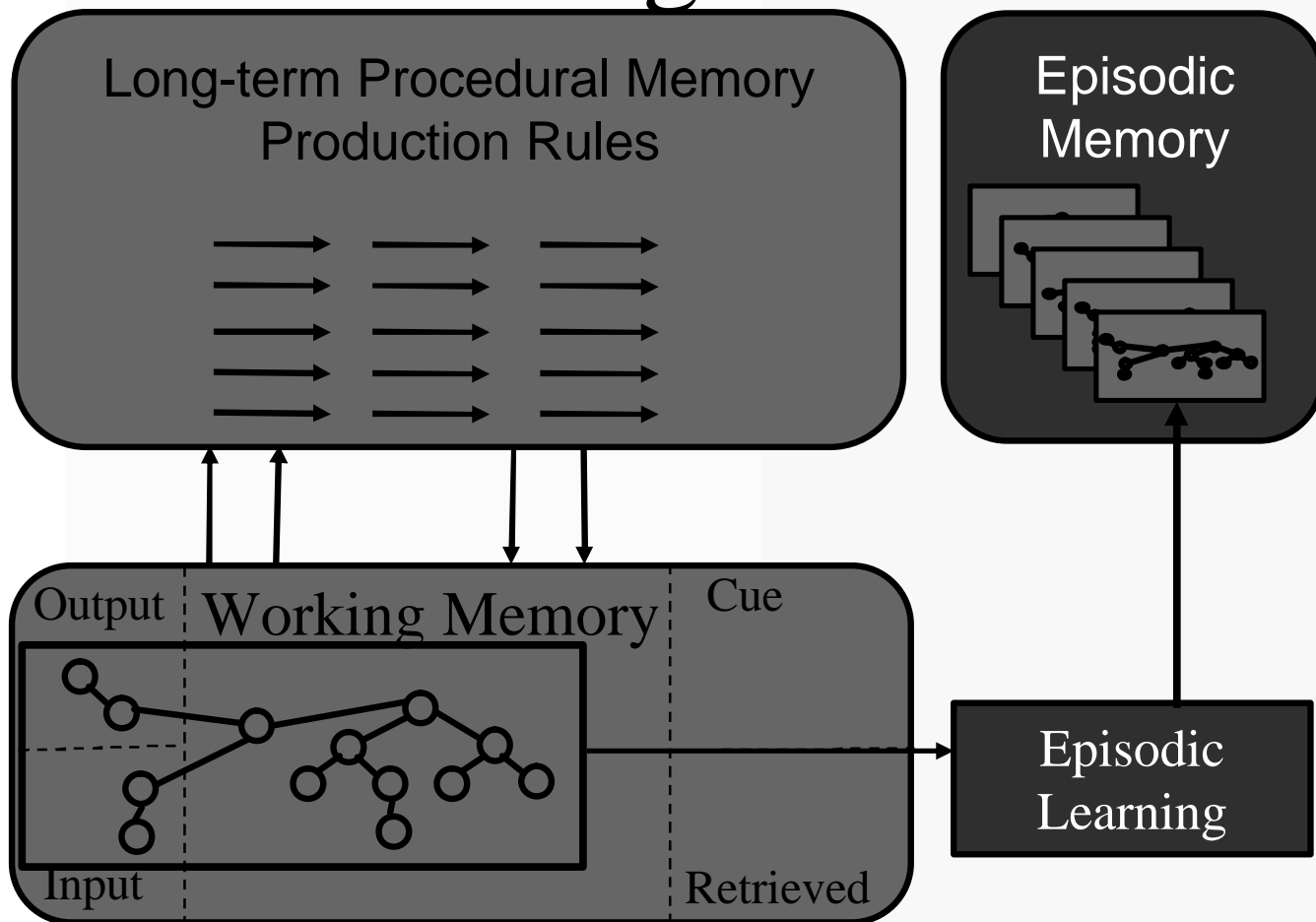
Initiation

Content

Storage

Episode Structure?

Retrieval



Episodes are stored in a separate memory

# Implementation Big Picture

Encoding

Initiation

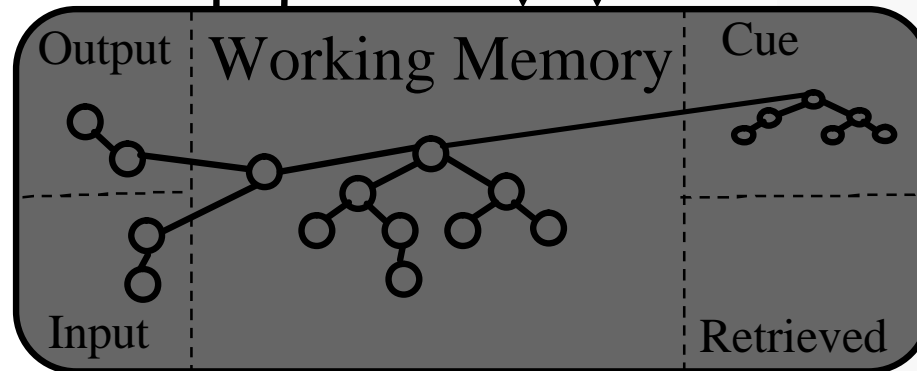
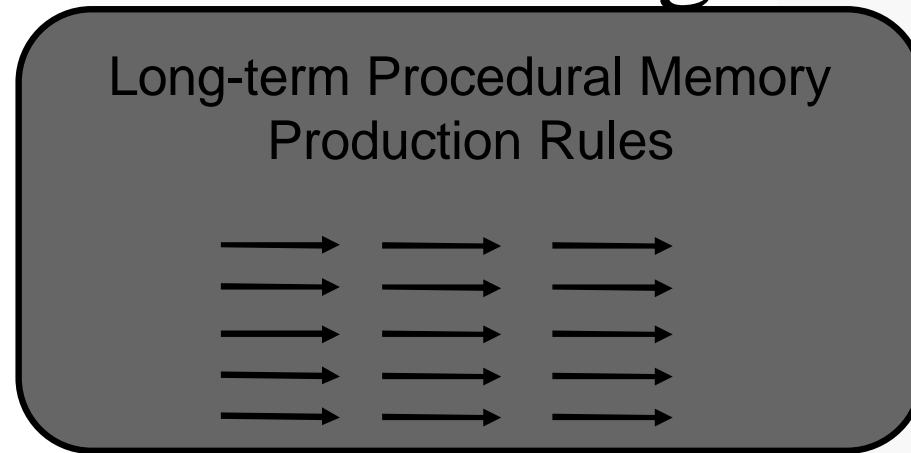
Content

Storage

Episode Structure

Retrieval

Initiation/Cue?



Cue is placed in an architecture specific buffer.

# Implementation Big Picture

Encoding

Initiation

Content

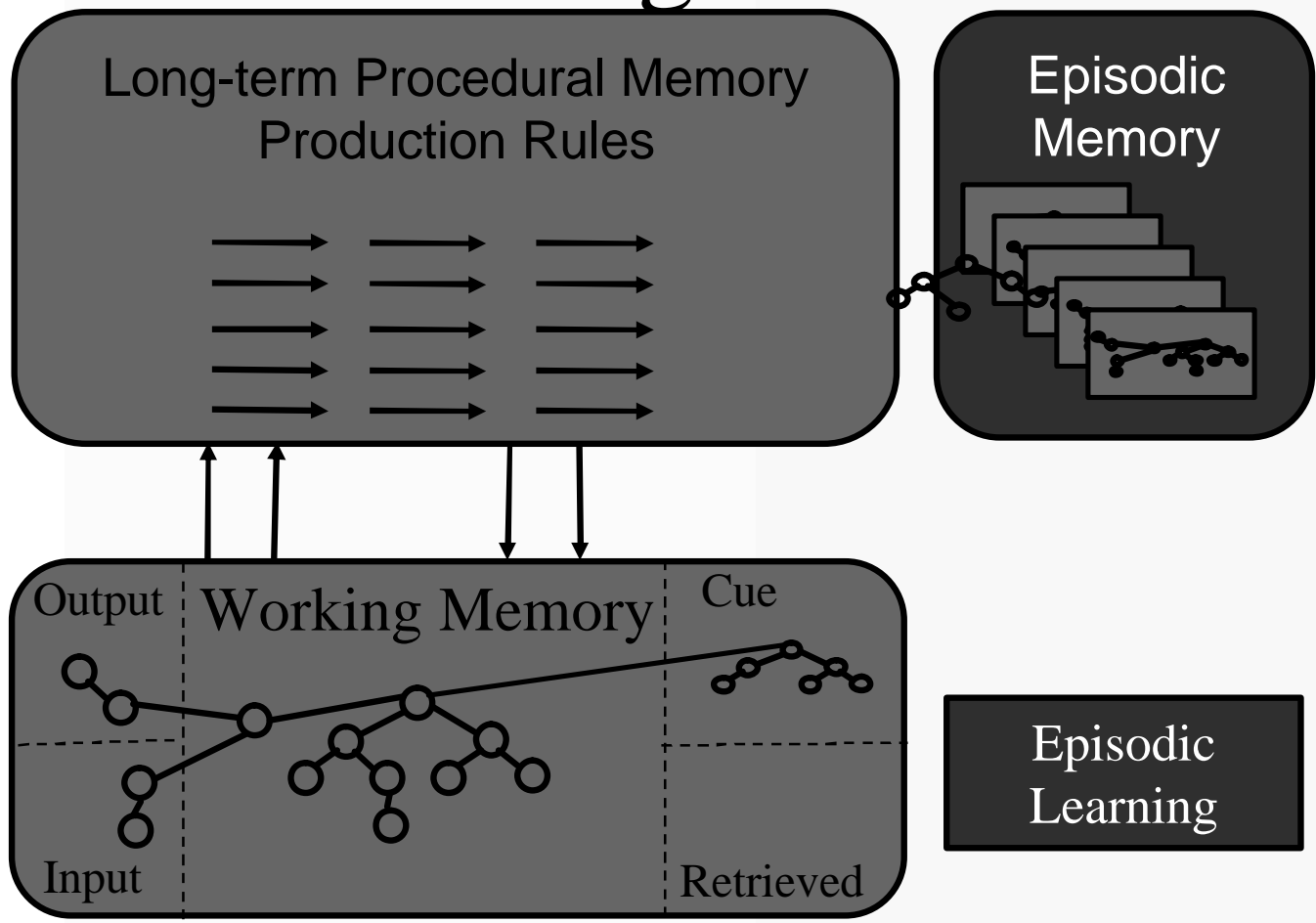
Storage

Episode Structure

Retrieval

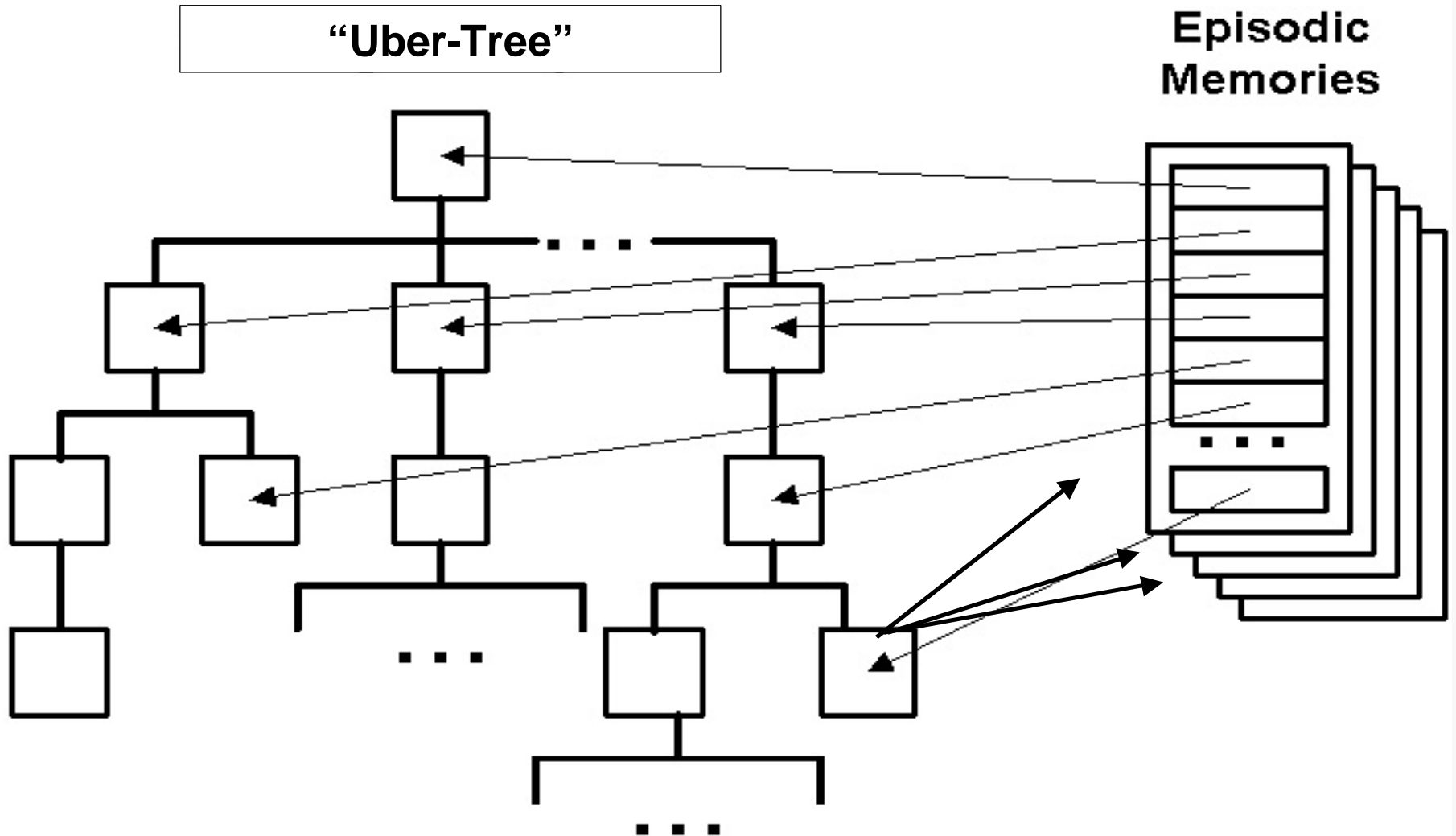
Initiation/Cue

Retrieval



The closest partial match is retrieved.

# Storage of Episodes

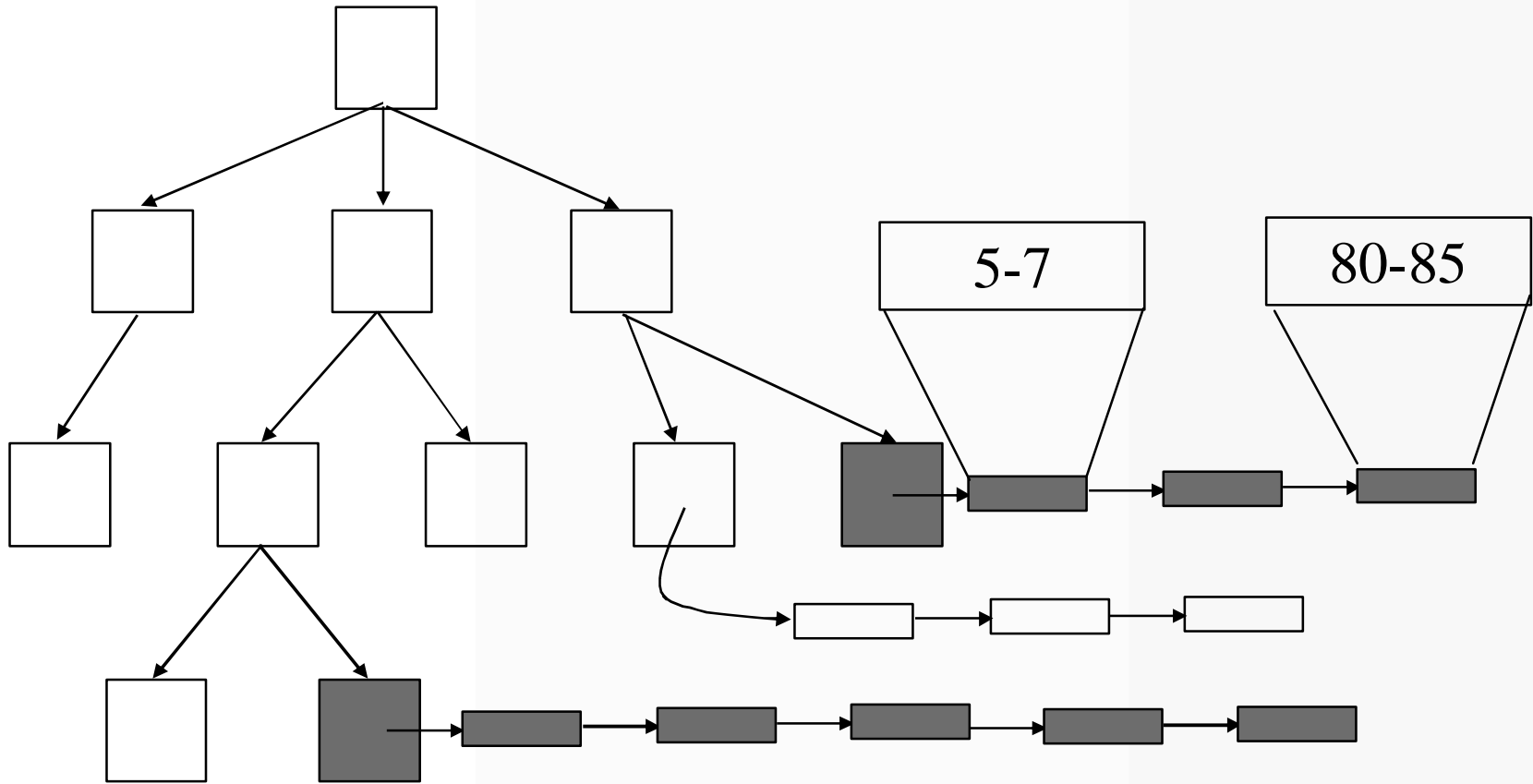


# Alternative Approach

- Observation:
  - Many items don't change from one episode to next
  - Can reconstruct episode from individual facts
  - Eliminate costly episode structure
- New representation
  - For each item, store ranges of when it exists
- New match
  - Trace through Über tree with cue to find all matching ranges
  - Compute score for merged ranges – pick best
  - Reconstruct episode by searching Über with episode number



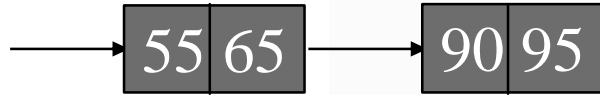
# Retrieval



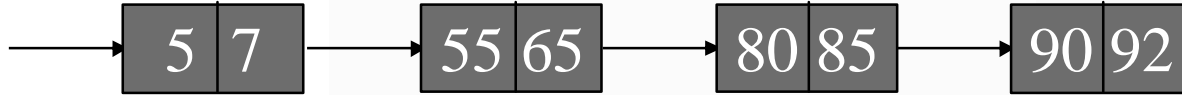
# Merge

Cue  
Activation

34



12



3



3

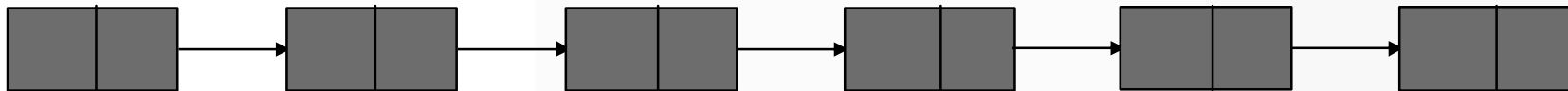
15

46

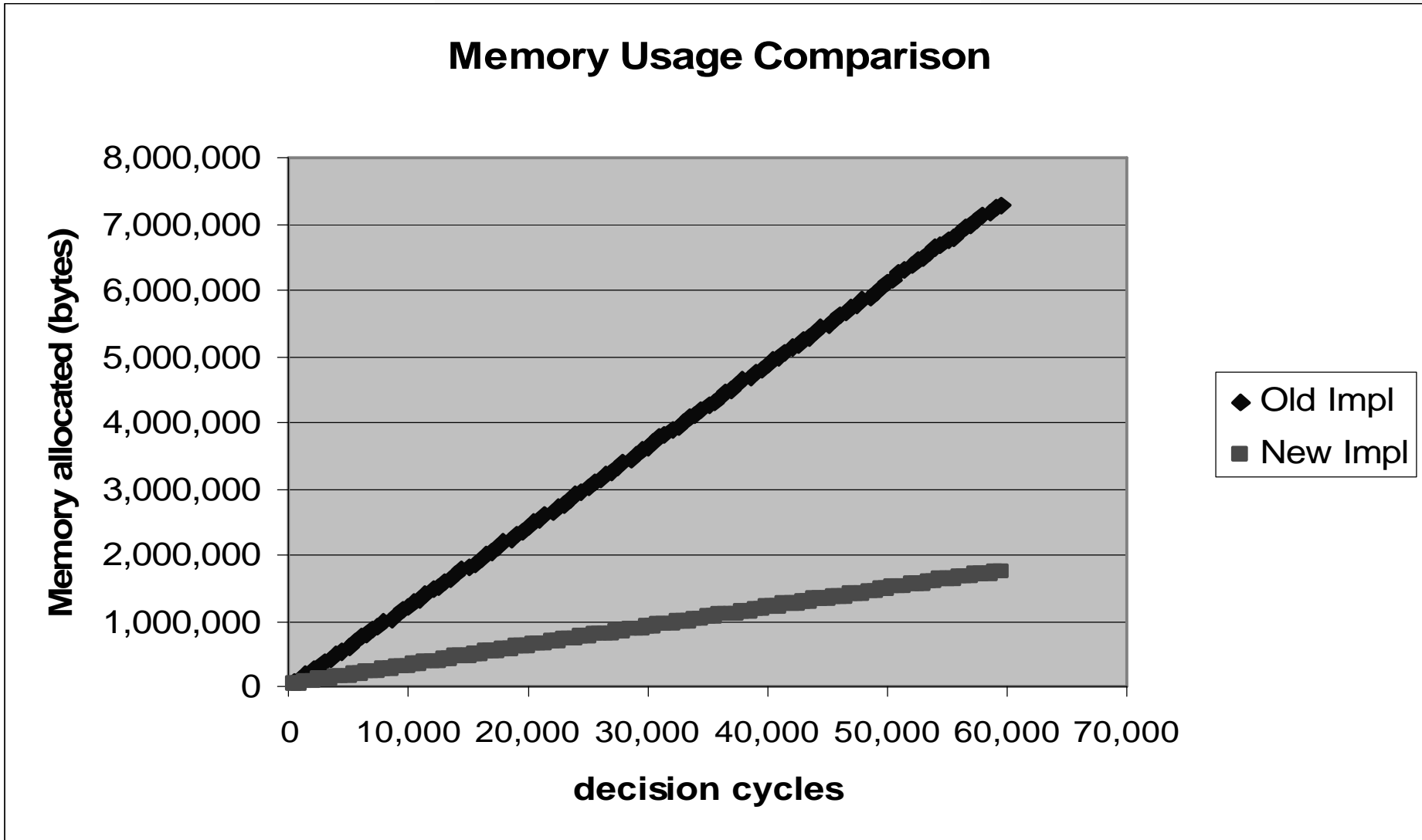
12

49

37



# Memory Usage



# Conclusion

- Explore use of episodic memory as general capability
  - Inspired by psychology
  - Constrained by computation and memory

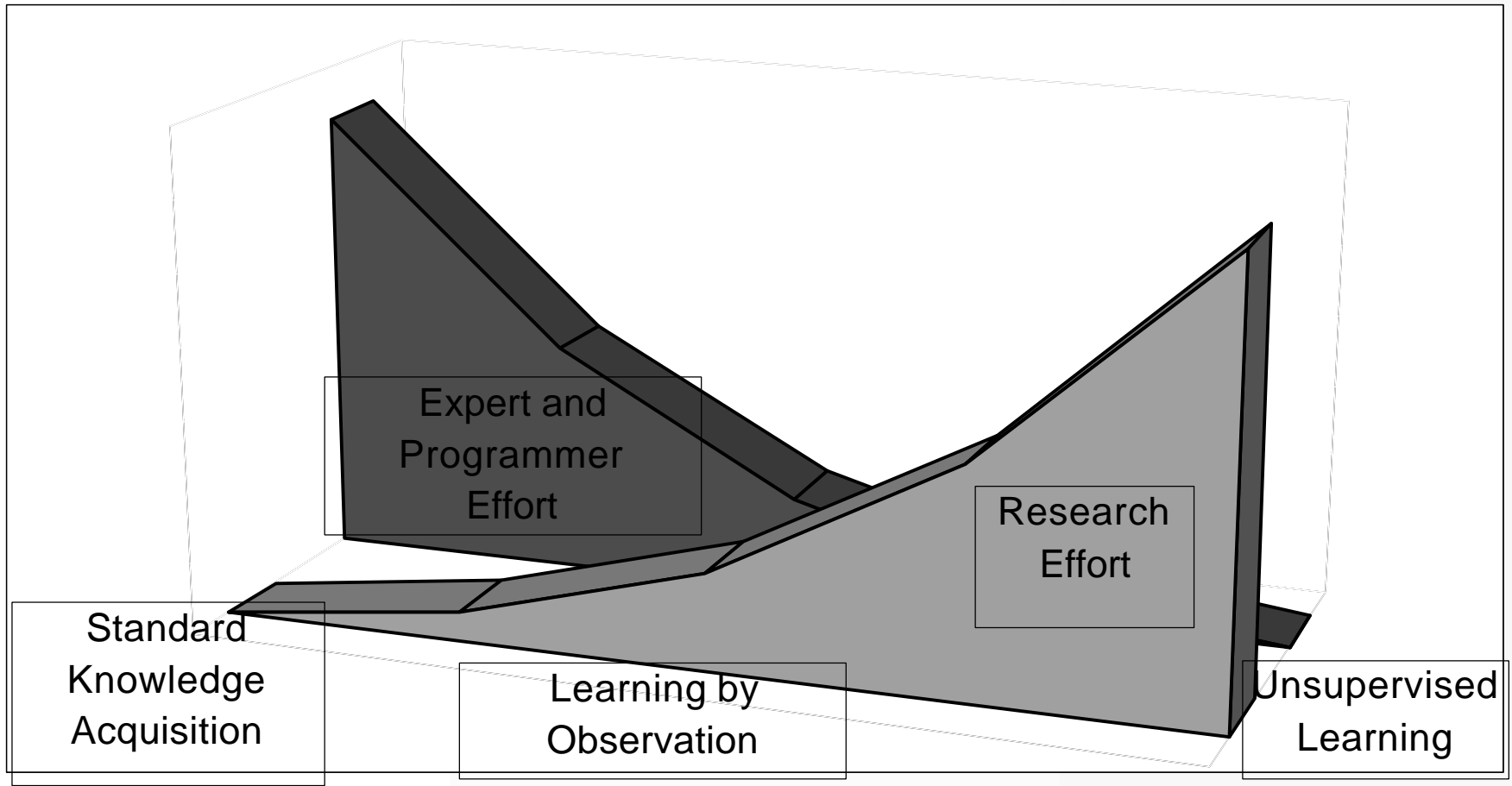
# Learning by Observation

Michael van Lent

# Background

- Goal: Learn rules to perform a task from watching an expert
  - Real time interaction with the game (agent-based approach)
  - Learning what goals to select & how to achieve them
- AI agents require lots of knowledge
  - TacAir-Soar: 8000+ rules
  - Quake II agent: 800+ rules
- Knowledge acquisition for these agents is expensive
  - 15 person/years for TacAir-Soar
- Learning is a cheaper alternative?

# Continuum of Approaches



# The Big Picture

- Problem
  - Task performance cast as classification
- Feedback
  - Supervised learning
- Knowledge Representation
  - Rules
  - Decision trees
- Knowledge Source
  - Observations of an expert
  - Annotations

# Knowledge Representation

- Rules encoding operators
- Operator Hierarchy
- Operator consists of:
  - Pre-conditions (potentially disjunctive)
    - Includes negated test for goal-achieved feature
  - Conditional Actions
    - Action attribute and value (pass-through action values)
  - Goal conditions (potentially disjunctive)
    - Create goal-achieved feature
    - Persistent and non-persistent goal-achieved features
- Task and Domain parameters are widely used to generalize the learned knowledge

# Operator Conditions

- Pre-conditions
  - Positive instance from each observed operator selection
- Action conditions
  - Positive instance from each observed action performance
  - Recent-changes heuristic can be applied
- Goal conditions
  - Positive instance from each observed operator termination
  - Recent-changes heuristic can be applied
- Action attributes and values
  - Attribute taken directly from expert actions
  - Value can be constant or “pass-through”



# Observation Trace

- At each time step record
  - Sensor input changes
    - List of attributes and values
  - Output commands
    - List of attributes and values
  - Operator annotations
    - List of active operators

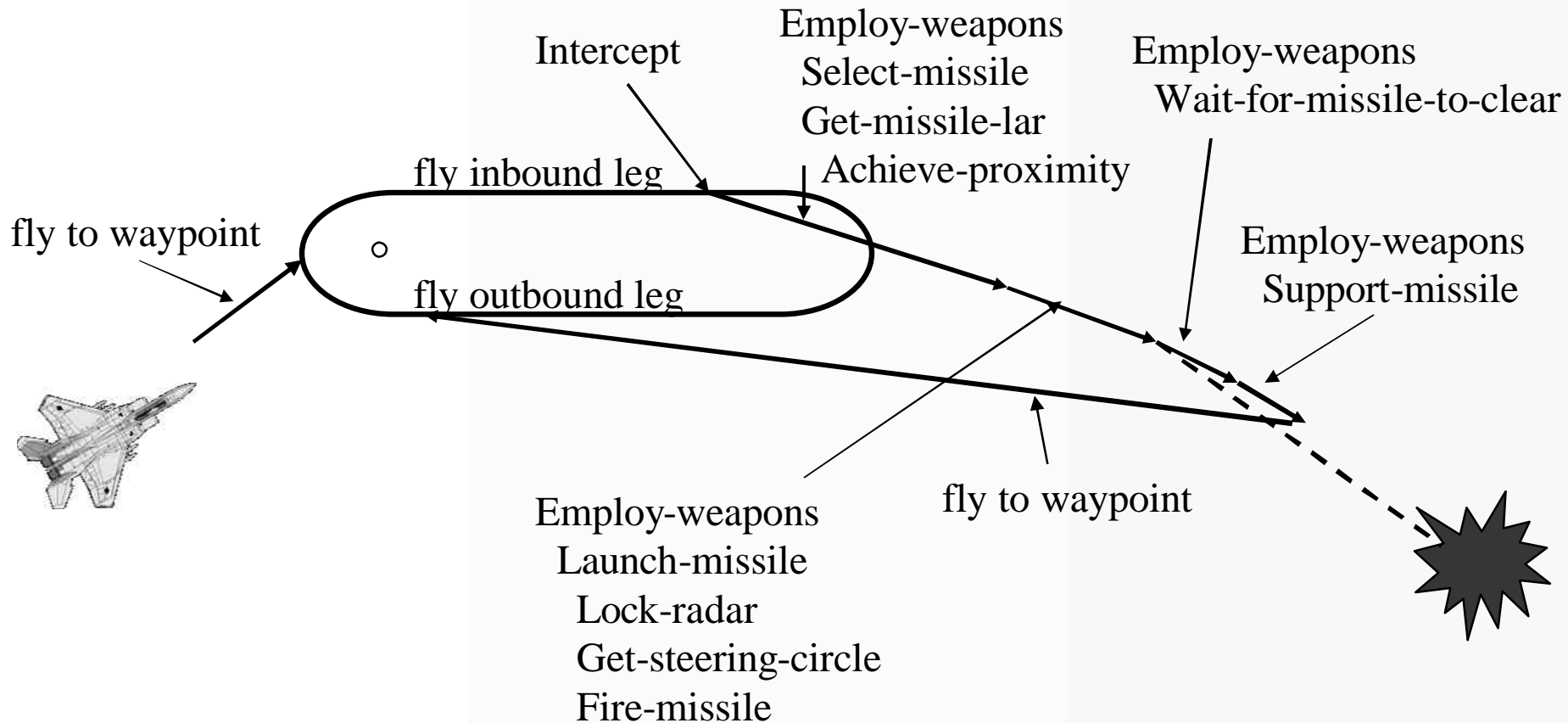
```
# Add Sensor Input for Decision Cycle 2
set Add_Sensor_Input(2,0) [list observe io input-link vehicle radar-mode tws-man ]
set Add_Sensor_Input(2,1) [list observe io input-link vehicle elapsed-time value 5938 ]
set Add_Sensor_Input(2,3) [list observe io input-link vehicle altitude value 1 ]
```

```
# Remove Sensor Input for Decision Cycle 2
set Remove_Sensor_Input(2,0) [list observe io input-link vehicle radar-mode *unknown* ]
set Remove_Sensor_Input(2,1) [list observe io input-link vehicle elapsed-time value 0 ]
set Remove_Sensor_Input(2,3) [list observe io input-link vehicle altitude value 0 ]
```

```
# Expert Actions for Decision Cycle 3
set Expert_Action_List(3) [list [list mvl-load-weapon-bay station-1 ] ]
```

```
# Expert Goal Stack for Decision Cycle 3
set Expert_Goal_Stack(3) [list init-agent station-1 ]
```

# Racetrack & Intercept Behavior



# Learning Example

## First selection of Fly-inbound-leg

- Radar Mode = TWS
- Altitude = 20,102
- Compass = 52
- Wind Speed = 3
- Waypoint Direction = 52
- Waypoint Distance = 1,996
- Near Parameter = 2,000

## Initial pre-conditions

- Radar Mode = TWS
- Altitude = 20,102
- Compass = 52
- Wind Speed = 3
- Waypoint Direction = 52
- Waypoint Distance = 1,996
- Compass == Waypoint Direction
- Waypoint Distance < Near Parameter

# Learning Example

## First instance of Fly-inbound-leg

- Radar Mode = TWS
- Altitude = 20,102
- Compass = 52
- Wind Speed = 3
- Waypoint Direction = 52
- Waypoint Distance = 1,996
- Near Parameter = 2,000

## Initial pre-conditions

- Radar Mode = TWS
- Altitude = 20,102
- Compass = 52
- Wind Speed = 3
- Waypoint Direction = 52
- Waypoint Distance = 1,996
- Compass == Waypoint Direction
- Waypoint Distance < Near Parameter

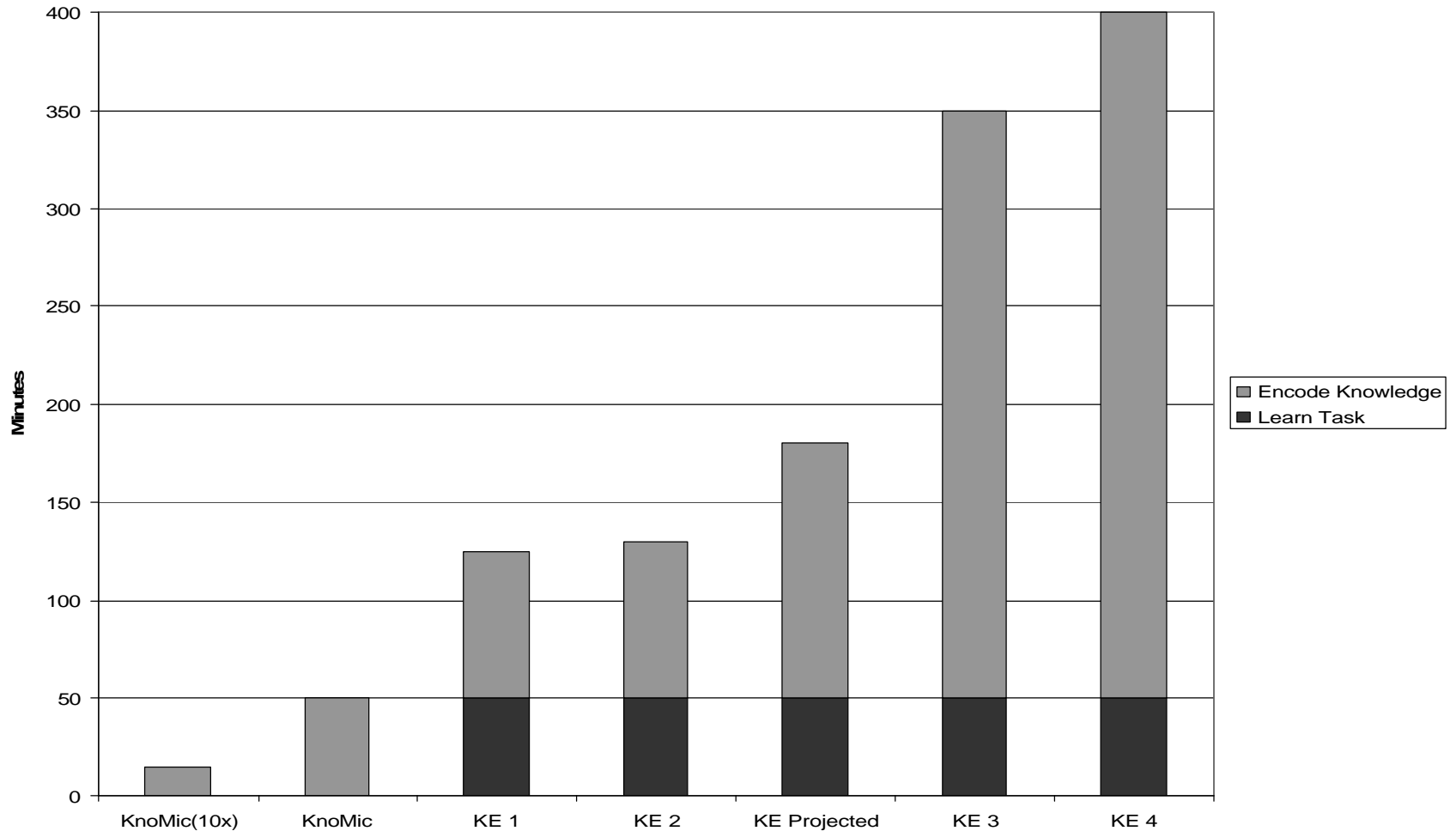
## Second instance of Fly-inbound-leg

- Radar Mode = TWS
- Altitude = 19,975
- Compass = 268
- Waypoint Direction = 270
- Waypoint Distance = 1,987
- Near Parameter = 2,000

## Revised pre-conditions

- Radar Mode = TWS
- Altitude = 19,975 - 20,102
- Waypoint Distance = 1,987 – 1,996
- Compass == Waypoint Direction
- Waypoint Distance < Near Parameter

# Results 2: Efficiency



# Evaluation

- Pros
  - Observations are fairly easy to get
  - Suitable for online learning (learn after each session)
  - AI can learn to imitate players
- Cons
  - Only more efficient for large rule sets?
  - Experts need to annotate the observation logs
- Challenges
  - Identifying the right features
  - Making sure you have enough observations

# References

- Learning Task Performance Knowledge by Observation
  - University of Michigan dissertation
- Knowledge Capture Conference (K-CAP).
- IJCAI Workshop on Modeling Others from Observation.
- AI Game Programming Wisdom.

# Learning Player Models

John Laird

# Learning Player Model

- Create an internal model of what player might do
- Allows AI to adapt to player's tactics & strategy
- Tactics
  - Player is usually found in room b, c, & f
  - Player prefers using the rocket launcher
  - Patterns of players' moves
    - When they block, attack, retreat, combinations of moves, etc.
- Strategy
  - Likelihood of player attacking from a given direction
  - Enemy tends to concentrate on technology and defense vs. exploration and attack

# Two Parts to Player Model

- Representation of player's behavior
  - Built up during playing
- Tactics that test player model and generate AI behavior

*Multiple approaches for each of these*

# Simple Representation of Behavior

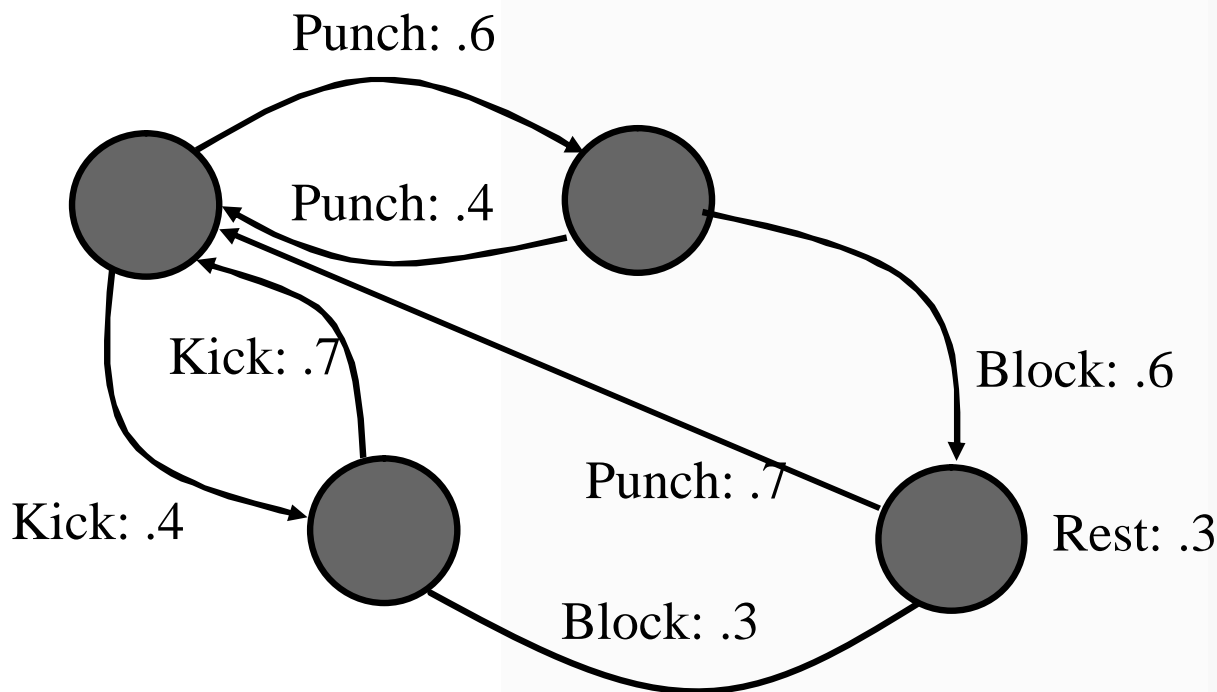
- Predefine set of *traits*
  - Always runs
  - Prefers dark rooms
  - Never blocks
- Simply count during game play
  - Doesn't track changes in style
- Limited horizon of past values
  - Frequency of using attack – range, melee, ...
  - $Traitvalue = a * ObservedValue + (1-a) * oldTraitValue$
  - $a$  = learning rate which determines influence of each observation

# Using Traits

- Pick traits your AI tactics code can use (or create tactics that can use the traits you gather).
- Tradeoff: level of detail vs. computation/complexity
  - Prefers dark rooms that have one entrance
  - More specialized better prediction, but more complex and less data

# Markov Decision Process (MDP) or N-Grams

- Build up a probabilistic state transition network that describes player's behavior

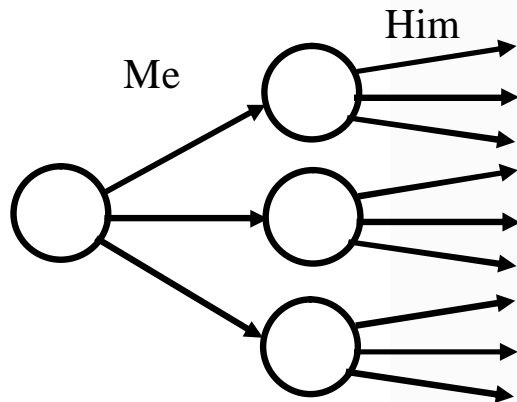


# Other Models

- Any decision making system:
  - Neural networks
  - Decision tree
  - Rule-based system
- Train with situation/action pairs
- Use AI's behavior as model of opponent
  - Chess, checkers, ...

# Using Player Model

- Tests for values and provide direct response
  - If player is likely to kick then block.
  - If player attacks very late, don't build defenses early on.
- Predict players behavior and search for best response
  - Can use general look-ahead/mini-max/alpha-beta search
  - Doesn't work with highly random games (Backgammon, Sorry)



# Anticipation

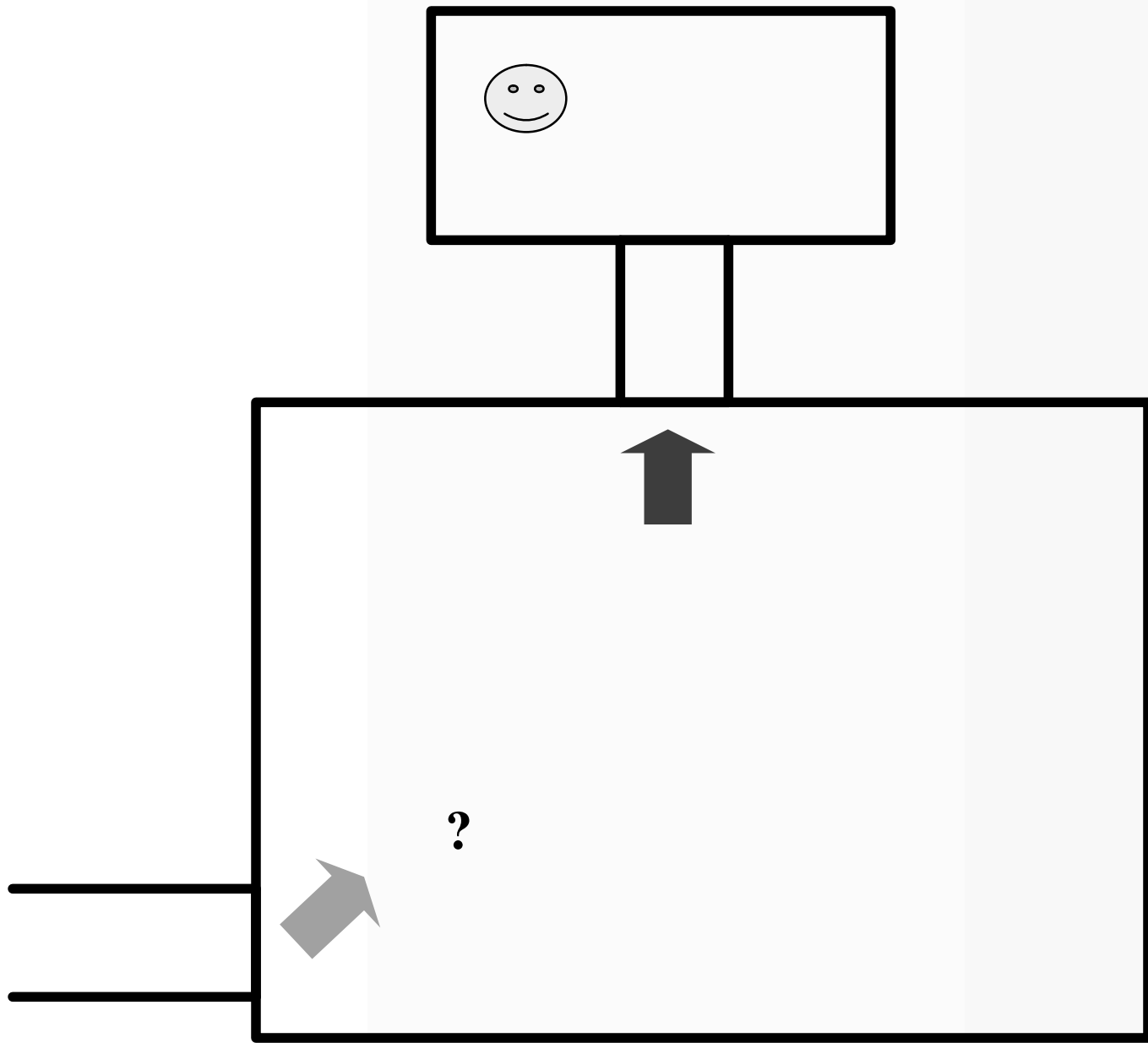
Dennis (Thresh) Fong:

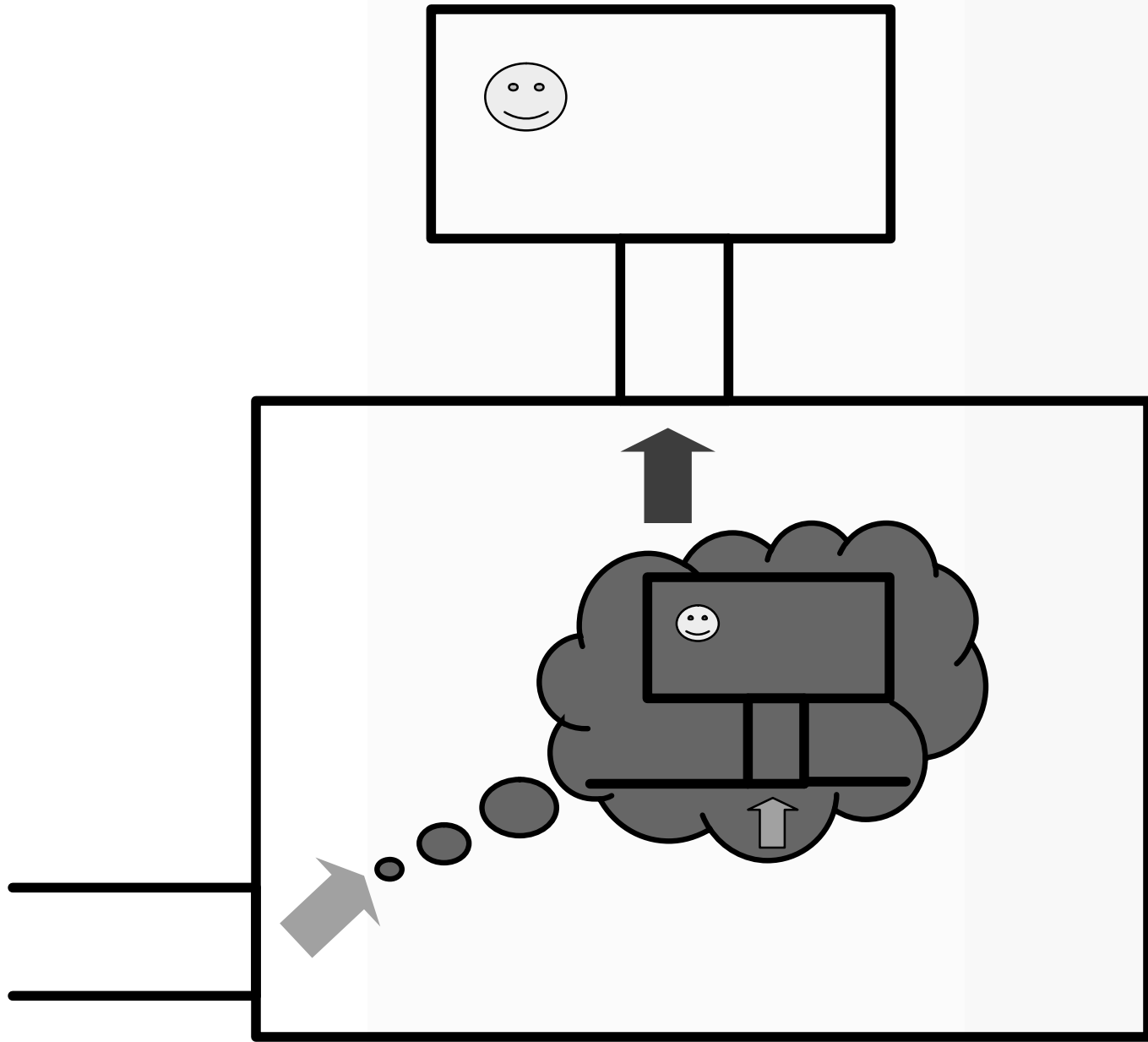
“Say my opponent walks into a room. I'm visualizing him walking in, picking up the weapon. On his way out, I'm waiting at the doorway and I fire a rocket two seconds before he even rounds the corner. A lot of people rely strictly on aim, but everybody has their bad aim days. So even if I'm having a bad day, I can still pull out a win. That's why I've never lost a tournament.”

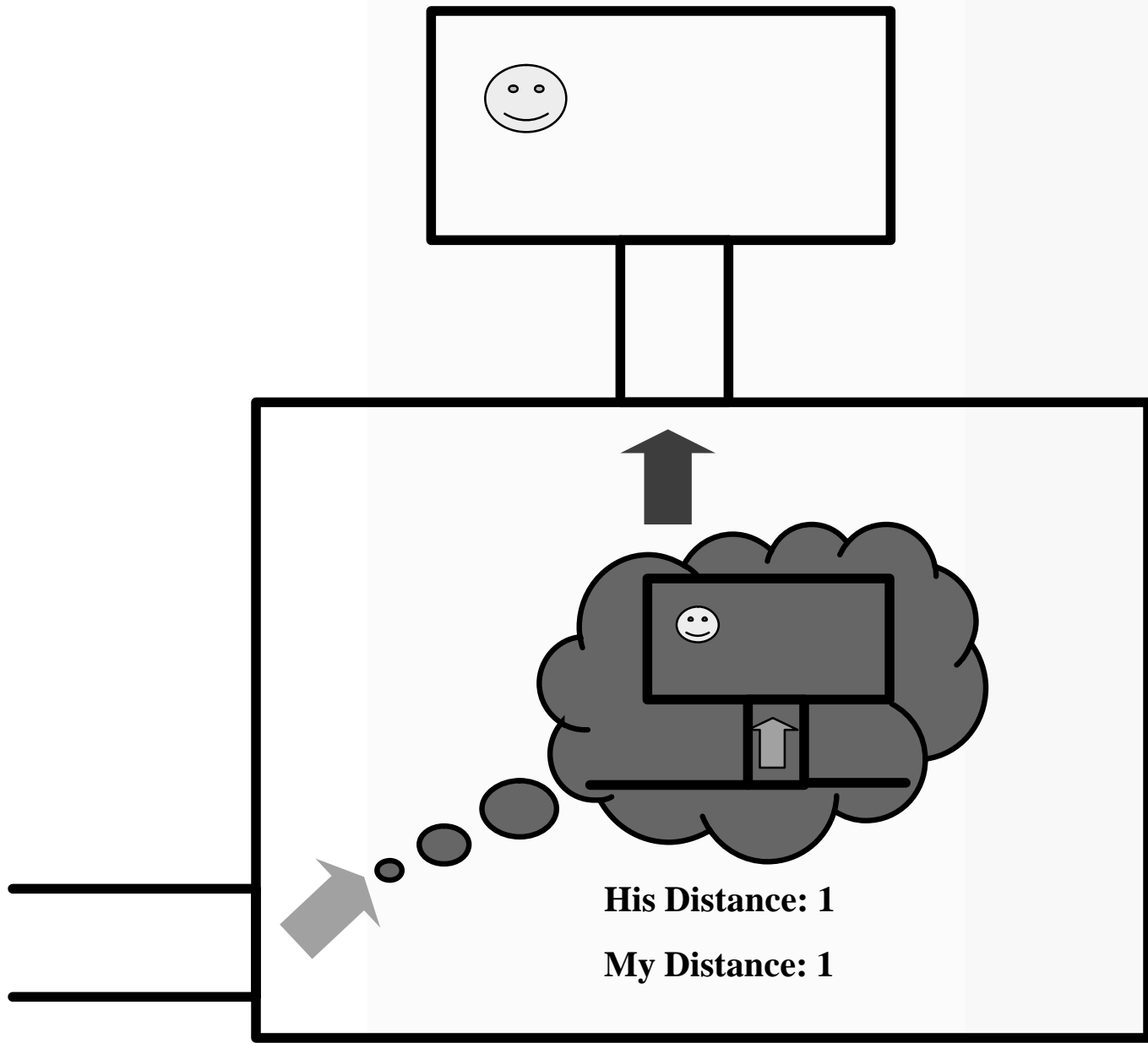
Newsweek, 11/21/99

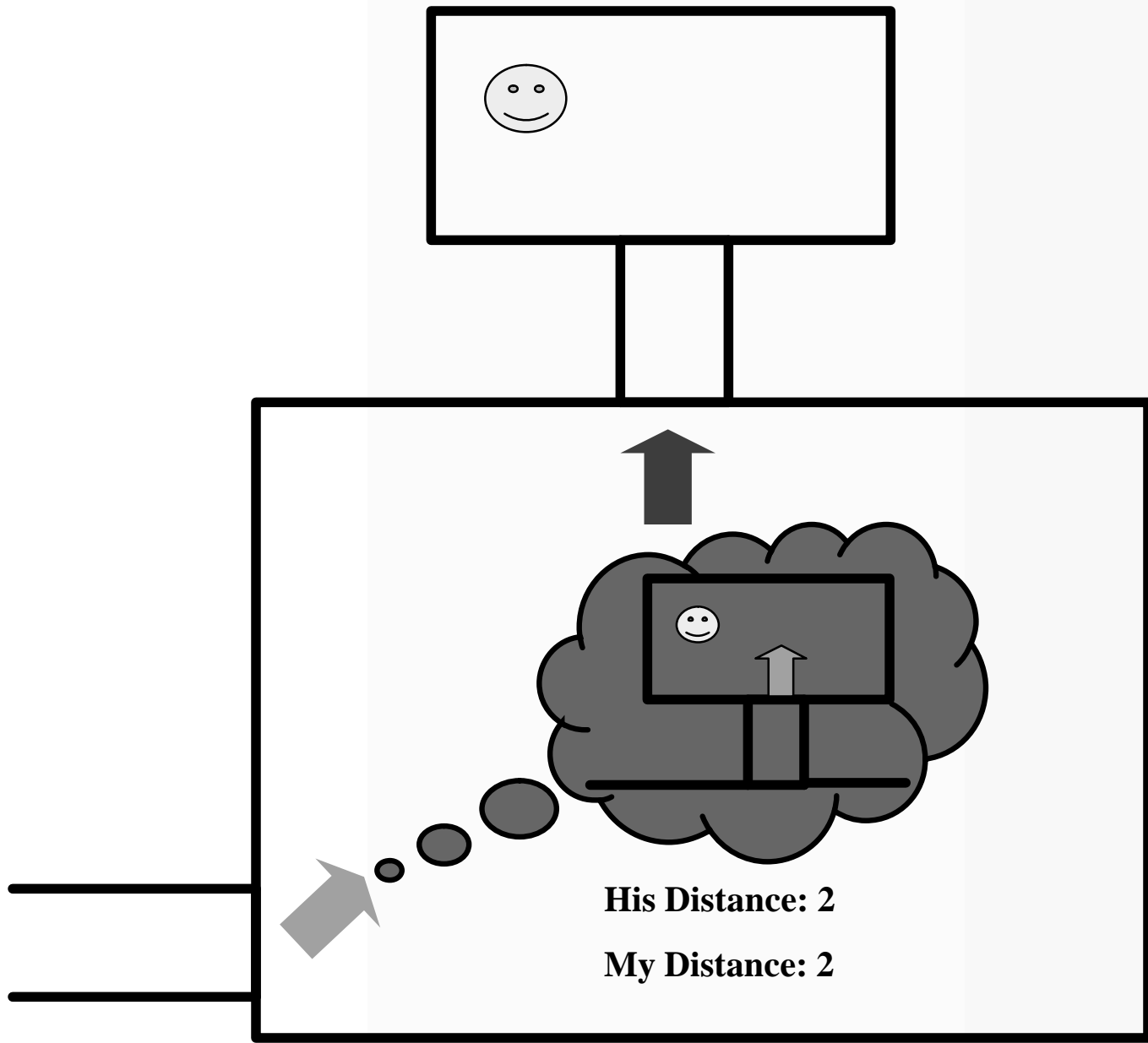
*Wayne Gretzky:*

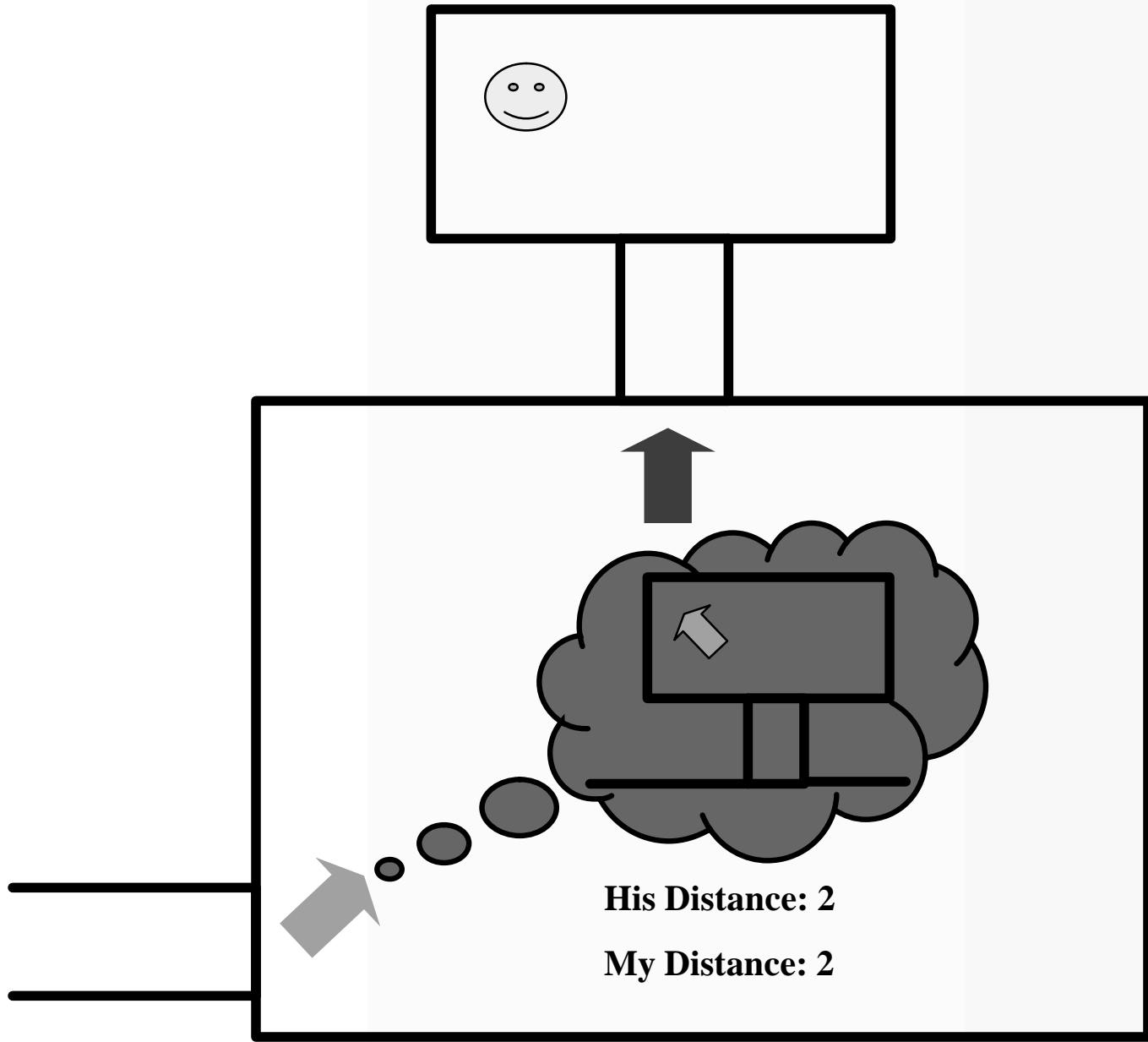
“Some people skate to the puck. I skate to where the puck is going to be.”

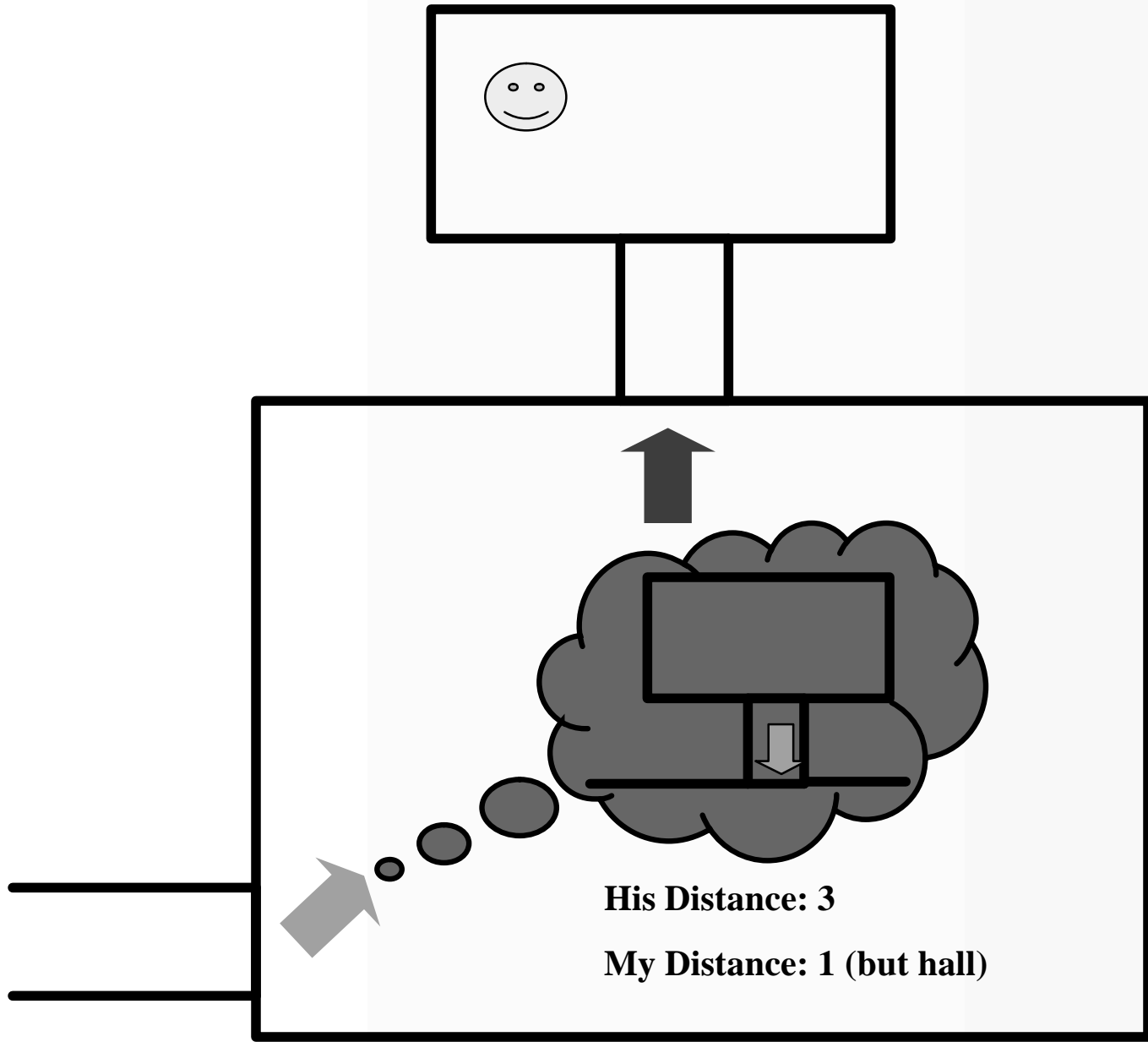


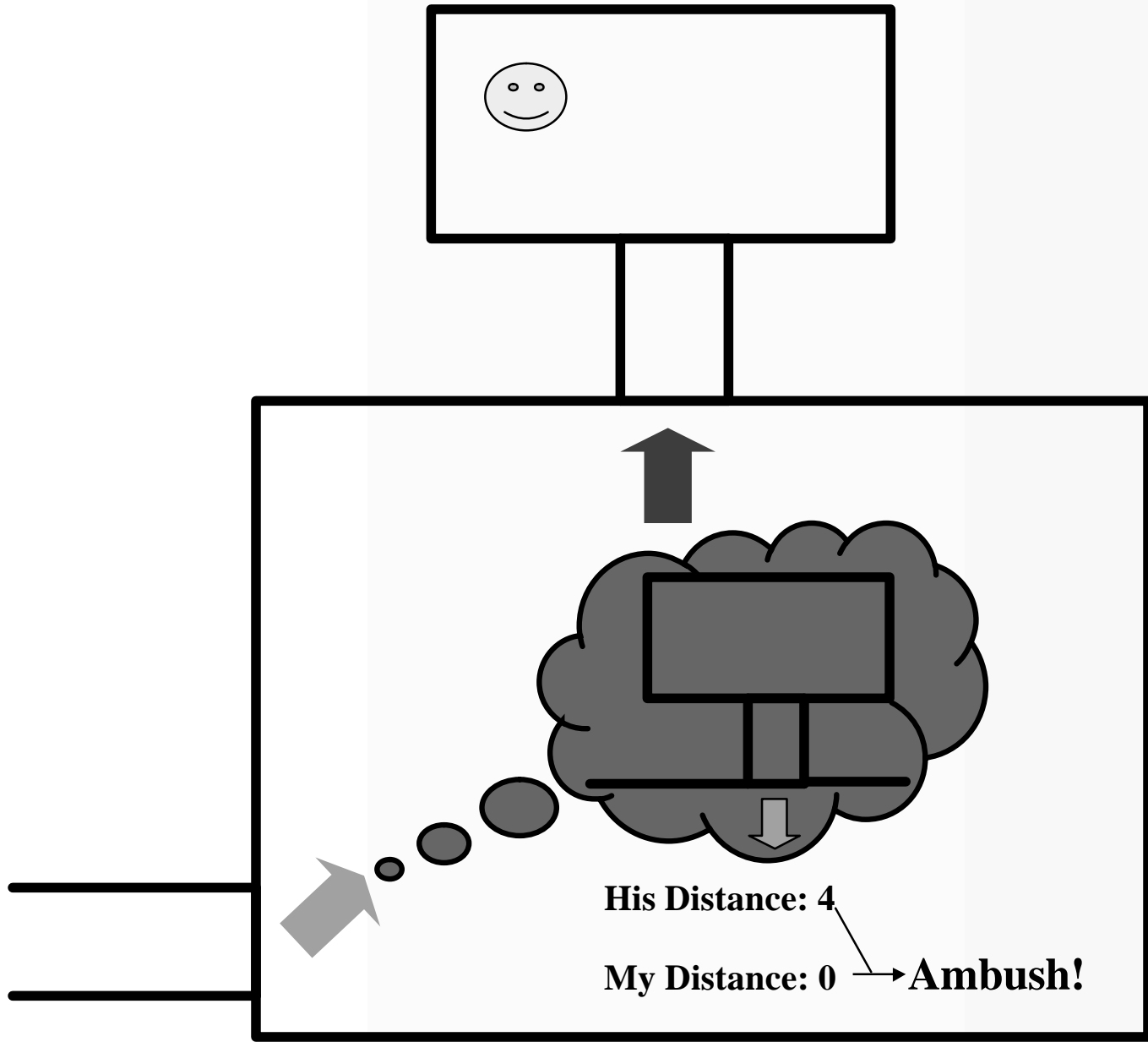


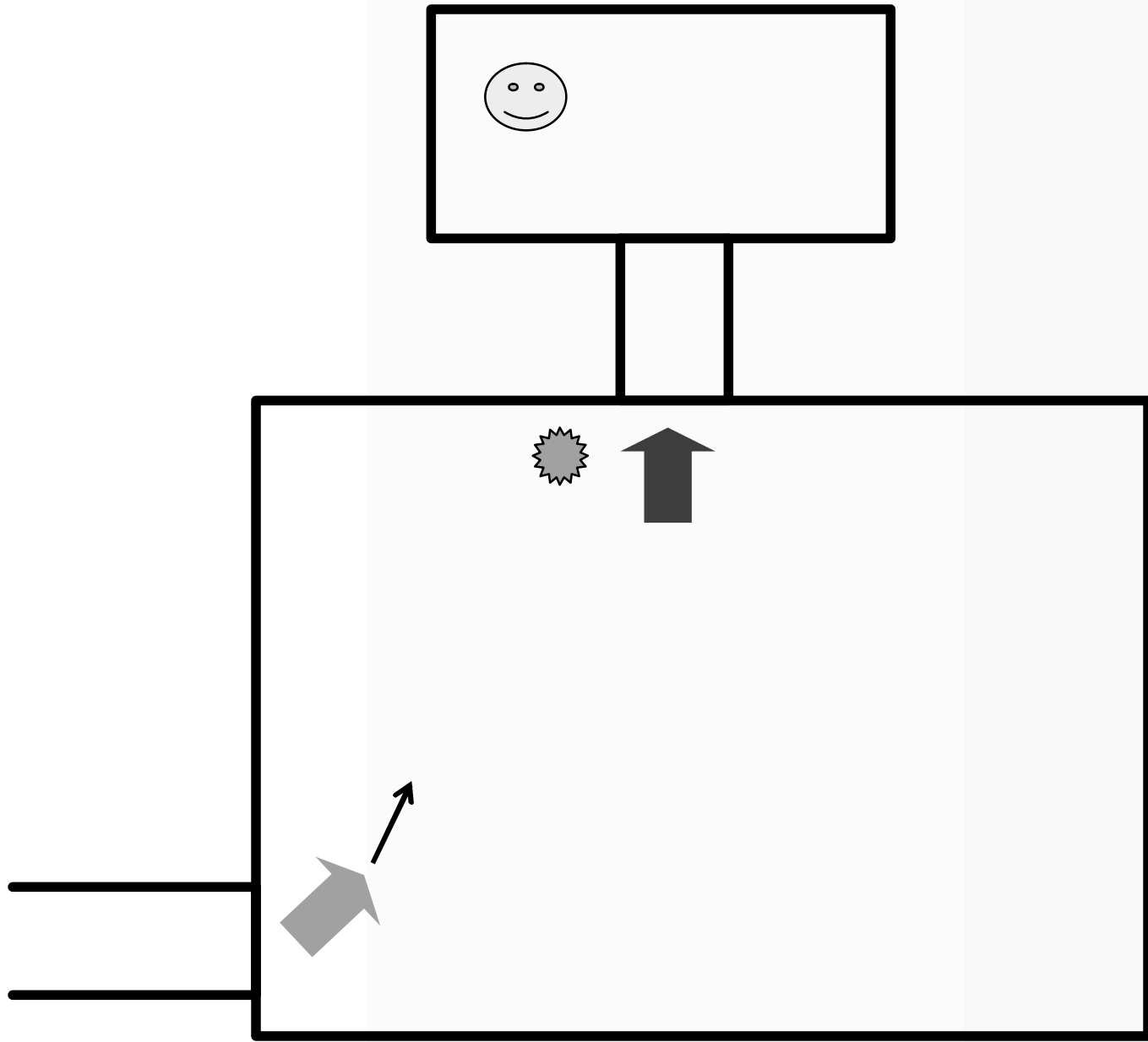


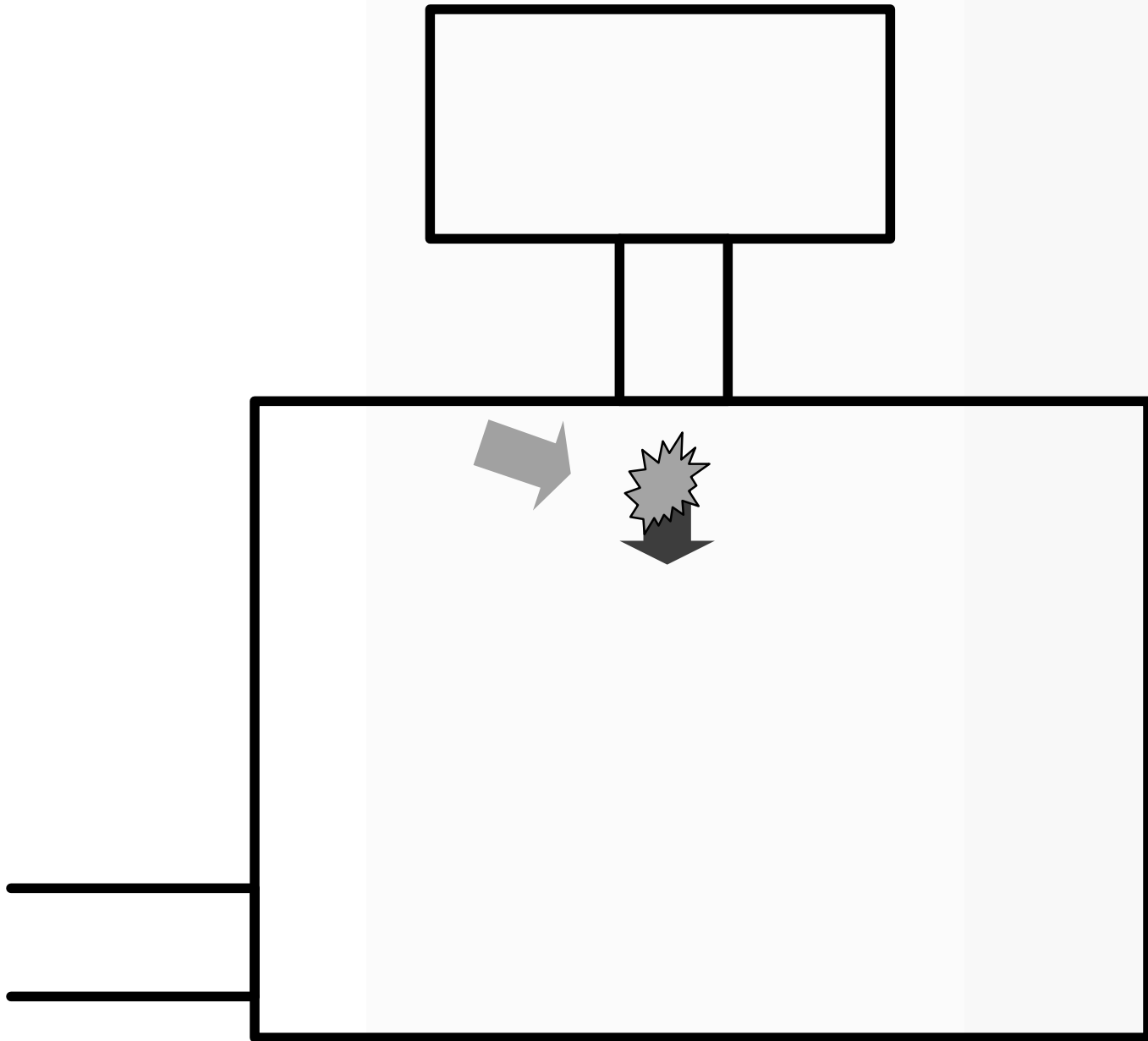












# Adaptive Anticipation

- Opponent might have different weapon preferences
  - Influences which weapons he pursues, which rooms he goes to
- Gather data on opponent's weapon preferences
  - Quakebot notices when opponent changes weapons
  - Use derived preferences for predicting opponent's behavior
  - Dynamically modifies anticipation with experience

# References

- Ryan Houlette: Player Modeling for Adaptive Games: AI Programming Wisdom 2, p. 557
- John Manslow: Learning and Adaptation: AI Programming Wisdom, p. 559
- Francois Laramée: Using N-Gram Statistical Models to Predict Play Behavior: AI Programming Wisdom, p. 596
- John Laird, It Knows What You're Going to Do: Adding Anticipation to a Quakebot. Agents 2001 Conference.

# Tutorial Overview

- I. Introduction to learning and games [.75 hour] {JEL}
- II. Overview of machine learning field [.75 hour] {MvL}
- III. Analysis of specific learning mechanisms [3 hours total]
  - Decision Trees [.5 hour] {MvL}
  - Neural Networks [.5 hour] {JEL}
  - Genetic Algorithms [.5 hour] {MvL}
  - Bayesian Networks [.5 hour] {MvL}
  - Reinforcement Learning [1 hour] {JEL}
- IV. Advanced Techniques [1 hour]
  - Episodic Memory [.3 hour] {JEL}
  - Behavior capture [.3 hour] {MvL}
  - Player modeling [.3 hour] {JEL}
- V. Questions and Discussion [.5 hour] {MvL & JEL}