CS 378: Computer Game Technology

AI – Decision Trees and Rule Systems

Spring 2012

Today

- AI
  - Decision trees
  - Rule-based systems

Classification

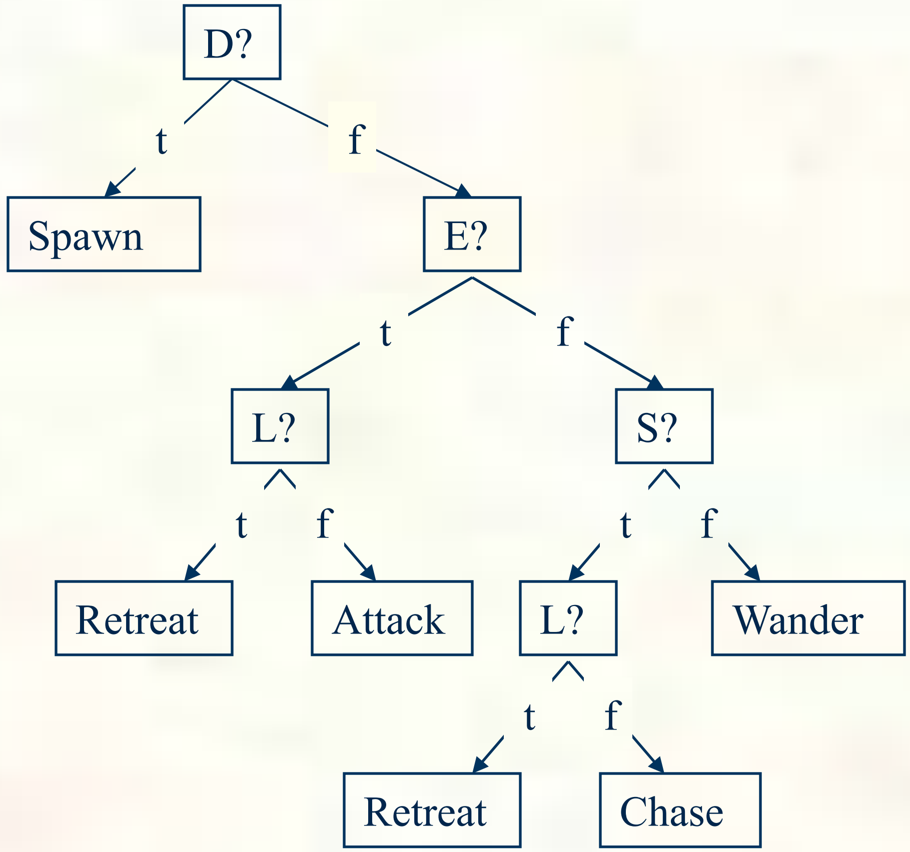
- Our aim is to decide which action to take given the world state
- Convert this to a classification problem:
  - The state of the world is a set of *attributes* (or *features*)
    - Who I can see, how far away they are, how much energy, …
  - Given any state, there is one appropriate action
    - Extends to multiple actions at the same time
  - The action is the *class* that a world state belongs to
    - Low energy, see the enemy means I should be in the retreat state
- Classification problems are *very* well studied

Decision Trees

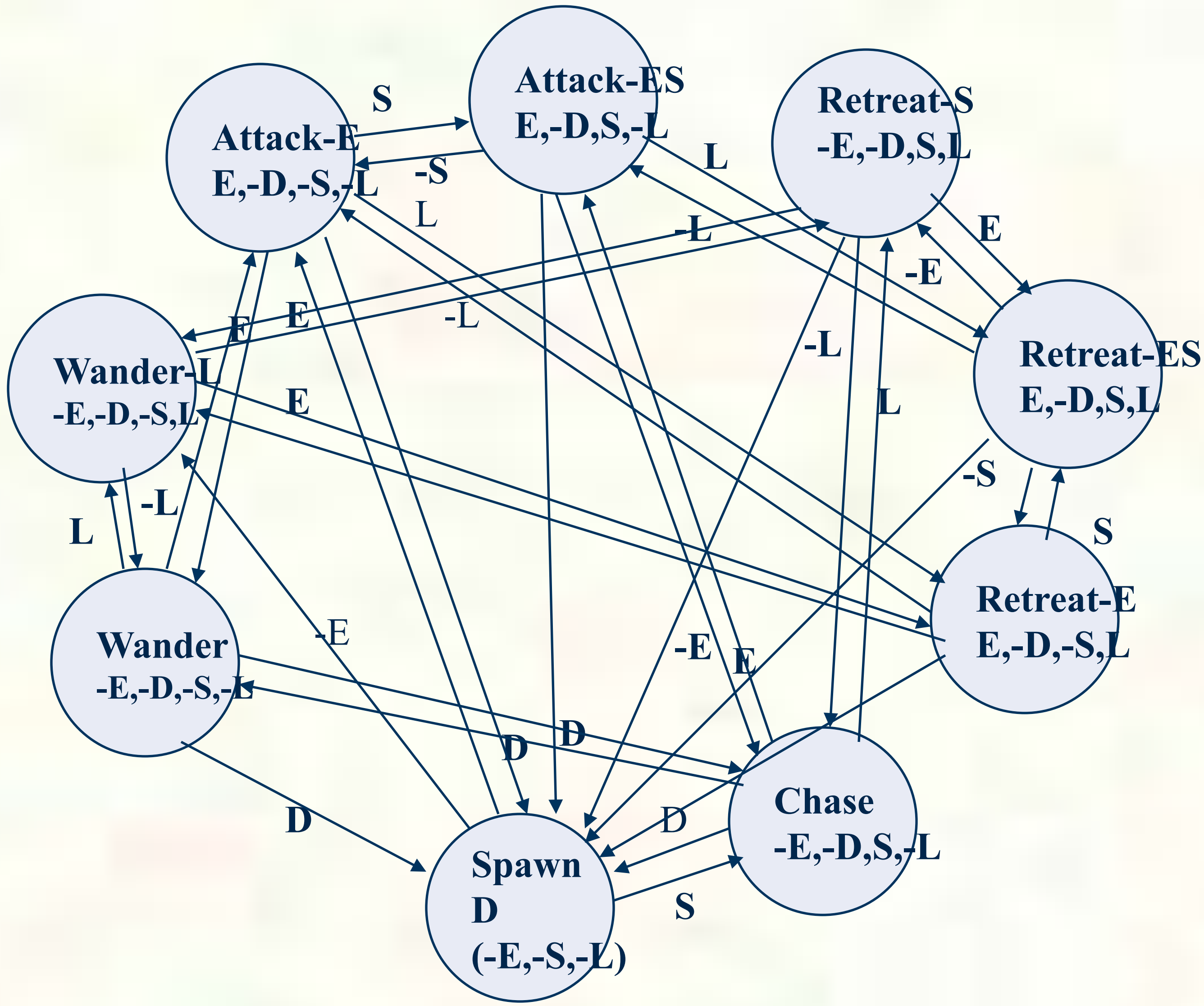
- Nodes represent attribute tests
  - One child for each possible outcome of the test
- Leaves represent classifications
  - Can have the same classification for several leaves
- Classify by descending from root to a leaf
  - At each node perform the test and descend the appropriate branch
  - When a leaf is reached return the classification (action) of that leaf
- Decision tree is a "disjunction of conjunctions of constraints on the attribute values of an instance"
  - Action if (A and B and C) or (A and ~B and D) or (…) …
  - Retreat if (low health and see enemy) or (low health and hear enemy) or (…) …

Decision Tree for Quake

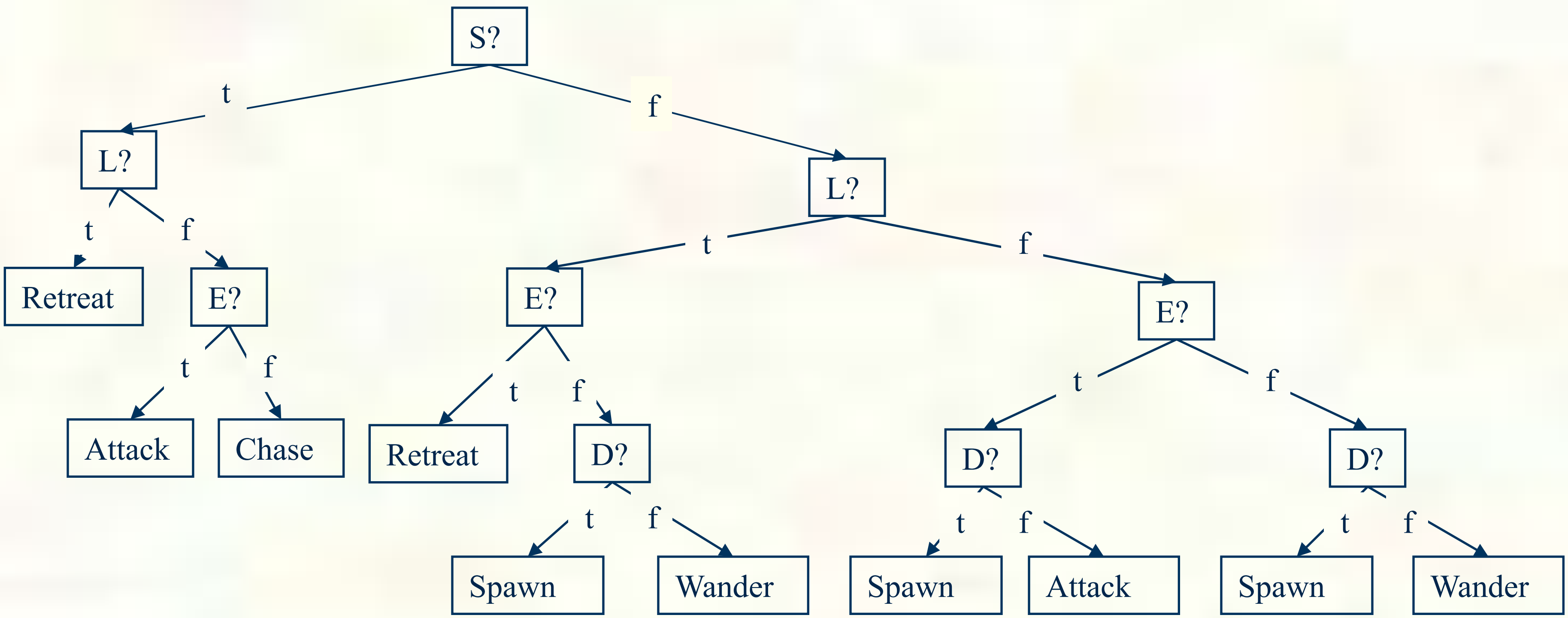
- Just one tree
- Attributes: Enemy=<t,f> Low=<t,f> Sound=<t,f> Death=<t,f>
- Actions: Attack, Retreat, Chase, Spawn, Wander
- Could add additional trees:
  - If I'm attacking, which weapon should I use?
  - If I'm wandering, which way should I go?
  - Can be thought of as just extending given tree (but easier to design)
  - Or, can share pieces of tree, such as a Retreat sub-tree



Compare and Contrast



Different Trees – Same Decision



Handling Simultaneous Actions

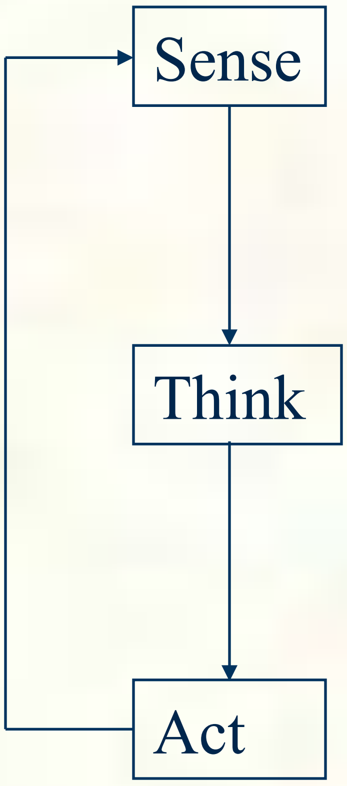
- Treat each output command as a separate classification problem
  - Given inputs should walk => <forward, backward, stop>
  - Given inputs should turn => <left, right, none>
  - Given inputs should run => <yes, no>
  - Given inputs should weapon => <blaster, shotgun…>
  - Given inputs should fire => <yes, no>
- Have a separate tree for each command
- If commands are not independent, two options:
  - Have a general conflict resolution strategy
  - Put dependent actions in one tree

Deciding on Actions

- Each time the AI is called:
  - Poll each decision tree for current output
  - Event driven - only call when state changes
- Need current value of each input attribute
  - All sensor inputs describe the state of the world
- Store the state of the environment
  - Most recent values for all sensor inputs
  - Change state upon receipt of a message
  - Or, check validity when AI is updated
  - Or, a mix of both (polling and event driven)

Sense, Think, Act Cycle

- Sense
  - Gather input sensor changes
  - Update state with new values
- Think
  - Poll each decision tree
- Act
  - Execute any changes to actions



Building Decision Trees

- Decision trees can be constructed by hand
  - Think of the questions you would ask to decide what to do
  - For example: Tonight I can study, play games or sleep. How do I make my decision?
- But, decision trees are typically *learned*:
  - Provide examples: many sets of attribute values and resulting actions
  - Algorithm then constructs a tree from the examples
  - Reasoning: We don't know how to decide on an action, so let the computer do the work
  - Whose behavior would we wish to learn?

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 that results in action A, then X must always be true for action A
  - More examples are better
  - Errors in examples cause difficulty
    - If X is true in most examples X must always be true
    - ID3 does a good job of handling errors (noise) in examples
  - Note that induction can result in errors
    - It may just be coincidence that X is true in all the examples
- Typical decision tree learning determines what tests are always true for each action
  - Assumes that if those things are true again, then the same action should result

Learning Algorithms

- Recursive algorithms
  - Find an attribute test that separates the actions
  - Divide the examples based on the test
  - Recurse on the subsets
- What does it mean to separate?
  - Ideally, there are no actions that have examples in both sets
  - Failing that, most actions have most examples in one set
  - The thing to measure is entropy - the degree of homogeneity (or lack of it) in a set
    - Entropy is also important for compression
- What have we seen before that tries to separate sets?
  - Why is this different?

Induction requires Examples

- Where do examples come from?
  - Programmer/designer provides examples
  - Capture an expert player's actions, and the game state, while they play
- # of examples needed depends on difficulty of concept
  - Difficulty: Number of tests needed to determine the action
  - 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 by estimating how well the tree does on examples it hasn't seen

Decision Tree Advantages

- Simpler, more compact representation
- State is recorded in a memory
  - Create “internal sensors” – Enemy-Recently-Sensed
- Easy to create and understand
  - Can also be represented as rules
- Decision trees can be learned

Decision Tree Disadvantages

- Decision tree engine requires more coding than FSM
  - Each tree is “unique” sequence of tests, so little common structure
- Need as many examples as possible
- Higher CPU cost - but not much higher
- Learned decision trees may contain errors

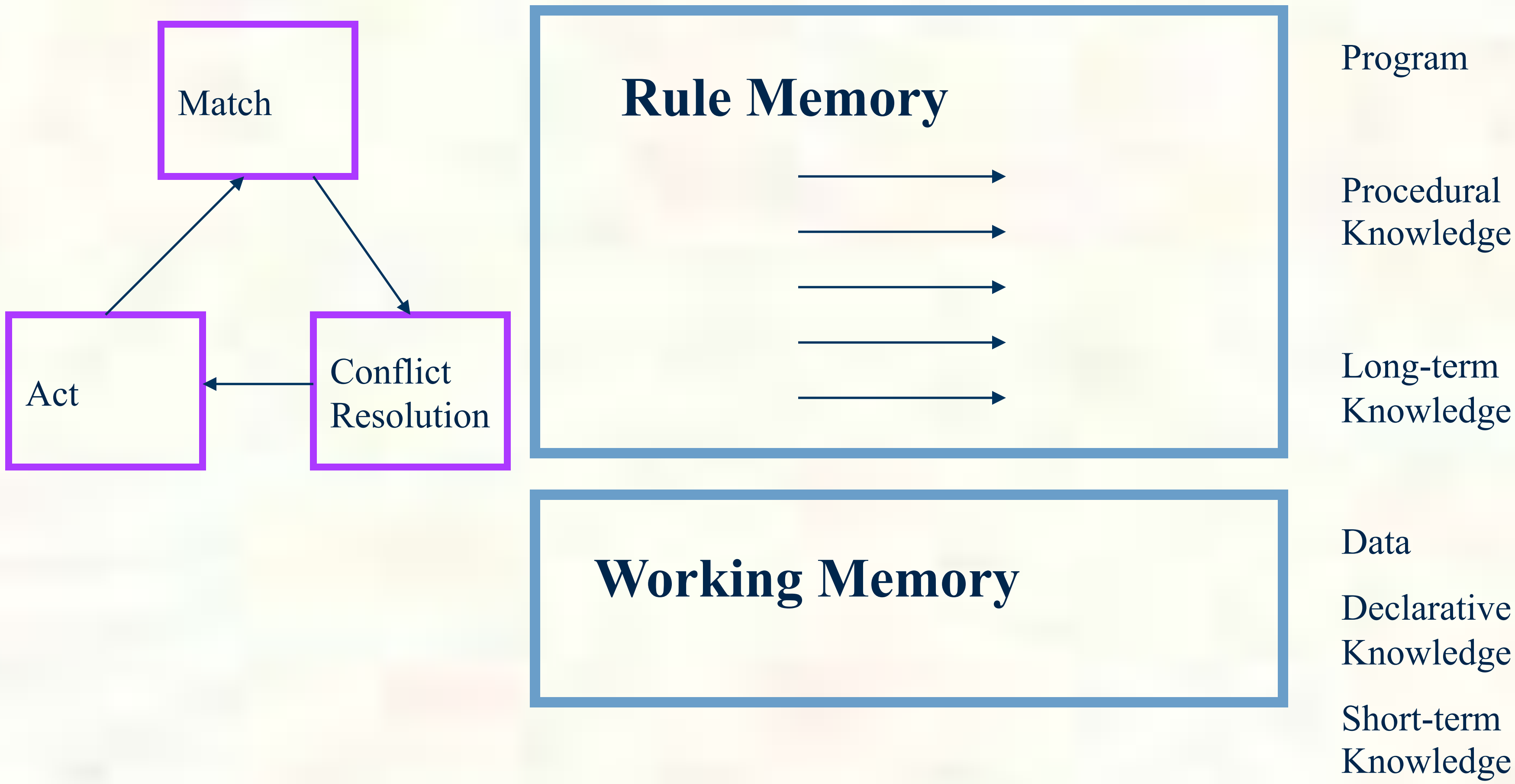
References

- 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,10th International Conference on Machine Learning, 1993

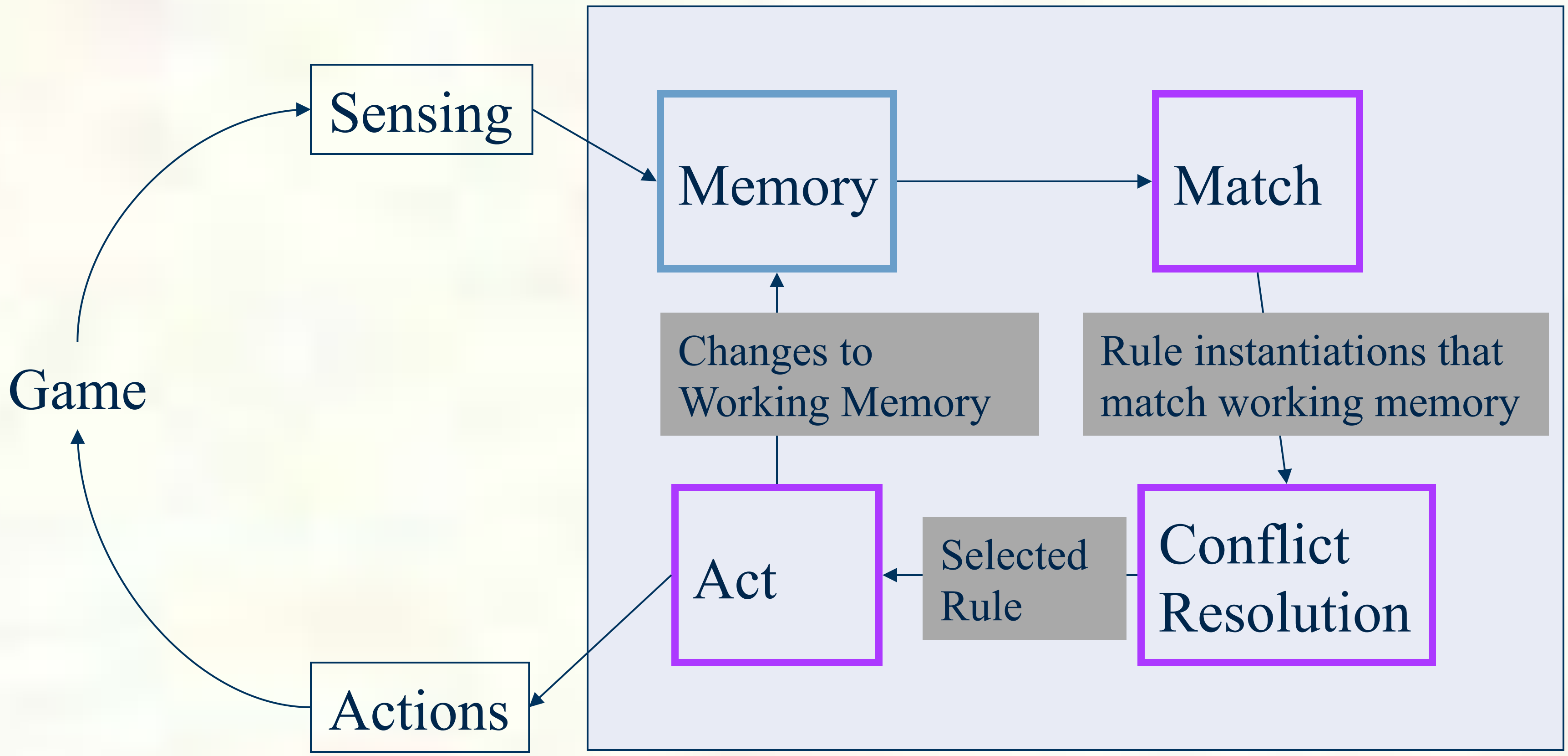
Rule-Based Systems

- Decision trees can be converted into rules
  - Just test the disjunction of conjunctions for each leaf
- More general rule-based systems let you write the rules explicitly
- System consists of:
  - A rule set - the rules to evaluate
  - A working memory - stores state
  - A matching scheme - decides which rules are applicable
  - A conflict resolution scheme - if more than one rule is applicable, decides how to proceed
- What types of games make the most extensive use of rules?

Rule-Based Systems Structure



AI Cycle


Sensing
Memory
Match
Game
Changes to Working Memory
Rule instantiations that match working memory
Act
Selected Rule
Conflict Resolution
Actions

Age of Kings

```
; The AI will attack once at 1100 seconds and then again
; every 1400 sec, provided it has enough defense soldiers.

(defrule
        (game-time > 1100)
=>
        (attack-now)
        (enable-timer 7 1400))

(defrule
        (timer-triggered 7)
        (defend-soldier-count >= 12)
=>
        (attack-now)
        (disable-timer 7)
        (enable-timer 7 1400))
```

Rule

Action

Age of Kings

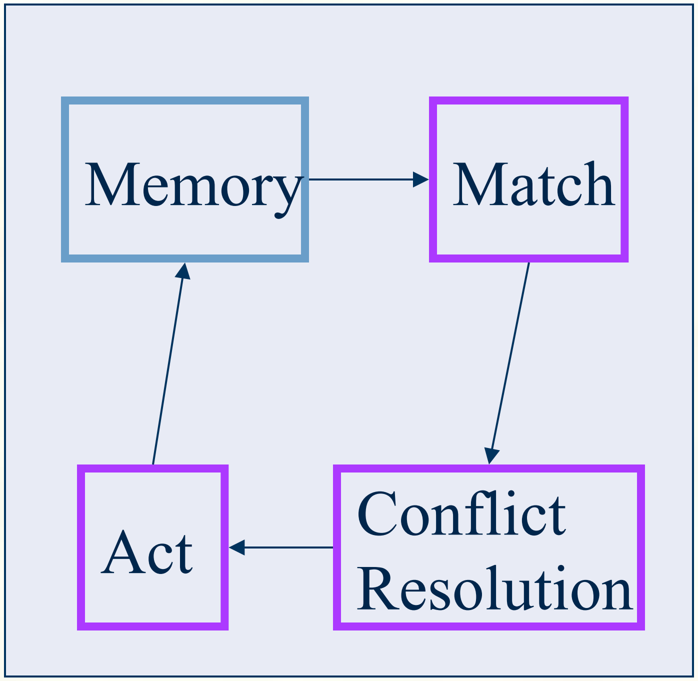
```
(defrule
        (true)
=>
        (enable-timer 4 3600)
        (disable-self))

(defrule
        (timer-triggered 4)
=>
        (cc-add-resource food 700)
        (cc-add-resource wood 700)
        (cc-add-resource gold 700)
        (disable-timer 4)
        (enable-timer 4 2700)
        (disable-self))
```

- What is it doing?

Implementing Rule-Based Systems

- Where does the time go?
  - 90-95% goes to Match
- Matching all rules against all of working memory each cycle is way too slow
- Key observation
  - # of changes to working memory each cycle is small
  - If conditions, and hence rules, can be associated with changes, then we can make things fast (event driven)

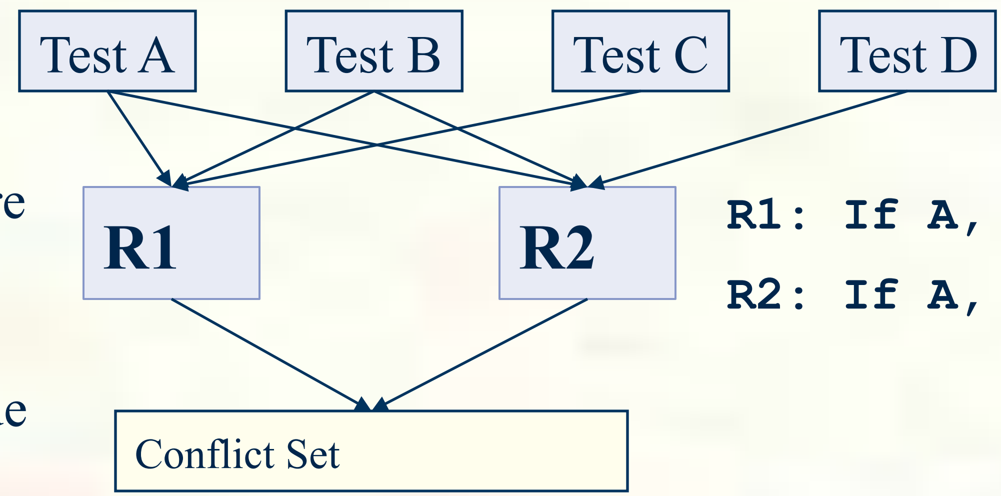


Efficient Special Case

- If only simple tests in conditions, compile rules into a *match net*
  - Simple means: Can map changes in state to rules that must be reevaluated
- Process changes to working memory
- Associate changes with tests
- Expected cost: Linear in the number of changes to working memory

Rules: Bit vectors store which tests are true

Rules with all tests true go in conflict set



R1: If A, B, C, then …

R2: If A, B, D, then …

General Case

- Rules can be arbitrarily complex
  - In particular: function calls in conditions and actions
- If we have arbitrary function calls in conditions:
  - Can't hash based on changes
  - Run through rules one at a time and test conditions
  - Pick the first one that matches (or do something else)
  - Time to match depends on:
    - Number of rules
    - Complexity of conditions
    - Number of rules that don't match

Baldurs Gate

```
IF
  Heard([PC],UNDER_ATTACK)
  !InParty(LastAttackerOf(LastHeardBy(Myself)))
  Range(LastAttackerOf(LastHeardBy(Myself)),5)
  !StateCheck(LastAttackerOf(LastHeardBy(Myself)),
              STATE_PANIC)
  !Class(Myself,FIGHTER_MAGE_THIEF)
THEN
  RESPONSE #100
  EquipMostDamagingMelee()
  AttackReevaluate(LastAttackerOf(LastHeardBy(Myself)),30)
END
```

Research Rule-based Systems

- Allow complex conditions with multiple variables
  - Function calls in conditions and actions
  - Can compute many relations using rules
- Examples:
  - OPS5, OPS83, CLIPS, ART, ECLIPS, …
- Laird: "Might be overkill for most of today's computer game AIs"

Conflict Resolution Strategies

- What do we do if multiple rules match?

Conflict Resolution Strategies

- What do we do if multiple rules match?
- Rule order – pick the first rule that matches
  - Makes order of loading important – not good for big systems
- Rule specificity - pick the most specific rule
- Rule importance – pick rule with highest priority
  - When a rule is defined, give it a priority number
  - Forces a total order on the rules – is right 80% of the time
  - Decide Rule 4 [80] is better than Rule 7 [70]
  - Decide Rule 6 [85] is better than Rule 5 [75]
  - Now have ordering between all of them – even if wrong

Basic Idea of Efficient Matching

- How do we reduce the cost of matching?
- Save intermediate match information (RETE)
  - Share intermediate match information between rules
  - Recompute intermediate information for changes
  - Requires extra memory for intermediate match information
  - Scales well to large rule sets
- Recompute match for rules affected by change (TREAT)
  - Check changes against rules in conflict set
  - Less memory than Rete
  - Doesn't scale as well to large rule sets
- Make extensive use of hashing (mapping between memory and tests/rules)

Rule-based System: Good and Bad

- Advantages
  - Corresponds to way people often think of knowledge
  - Very expressive
  - Modular knowledge
    - Easy to write and debug compared to decision trees
    - More concise than FSM
- Disadvantages
  - Can be memory intensive
  - Can be computationally intensive
  - Sometimes difficult to debug

References

- RETE:
  - Forgy, C. L. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1) 1982, pp. 17-37
- TREAT:
  - Miranker, D. TREAT: A new and efficient match algorithm for AI production systems. Pittman/Morgan Kaufman, 1989