the gamedesigninitiative at cornell university

Lecture 20

Character AI: Thinking and Acting

Take Away for Today

- Review the **sense-think-act** cycle
  - How do we separate actions and thinking?
  - Delay the sensing problem to next time
- What is **rule-based** character AI?
  - How does it relate to sense-think-act?
  - What are its advantages and disadvantages?
- What **alternatives** are there to rule-based AI?
  - What is our motivation for using them?
  - How do they affect the game architecture?

Classical AI vs. Game AI

- **Classical**: Design of *intelligent agents*
  - Perceives environment, maximizes its success
  - Established area of computer science
  - Subtopics: planning, machine learning
- **Game**: Design of *rational behavior*
  - Does not need to optimize (and often will not)
  - Often about "scripting" a personality
  - More akin to cognitive science

Role of AI in Games

- **Autonomous Characters** (NPCs)
  - Mimics the "personality" of the character
  - May be opponent or support character
- **Strategic Opponents**
  - AI at the "player level"
  - Closest to classical AI
- **Character Dialog**
  - Intelligent commentary
  - Narrative management (e.g. Façade)

Review: Sense-Think-Act

- **Sense**:
  - Perceive the world
  - Reading the game state
  - **Example**: enemy near?
- **Think**:
  - Choose an action
  - Often merged with sense
  - **Example**: fight or flee
- **Act**:
  - Update the state
  - Simple and fast
  - **Example**: reduce health

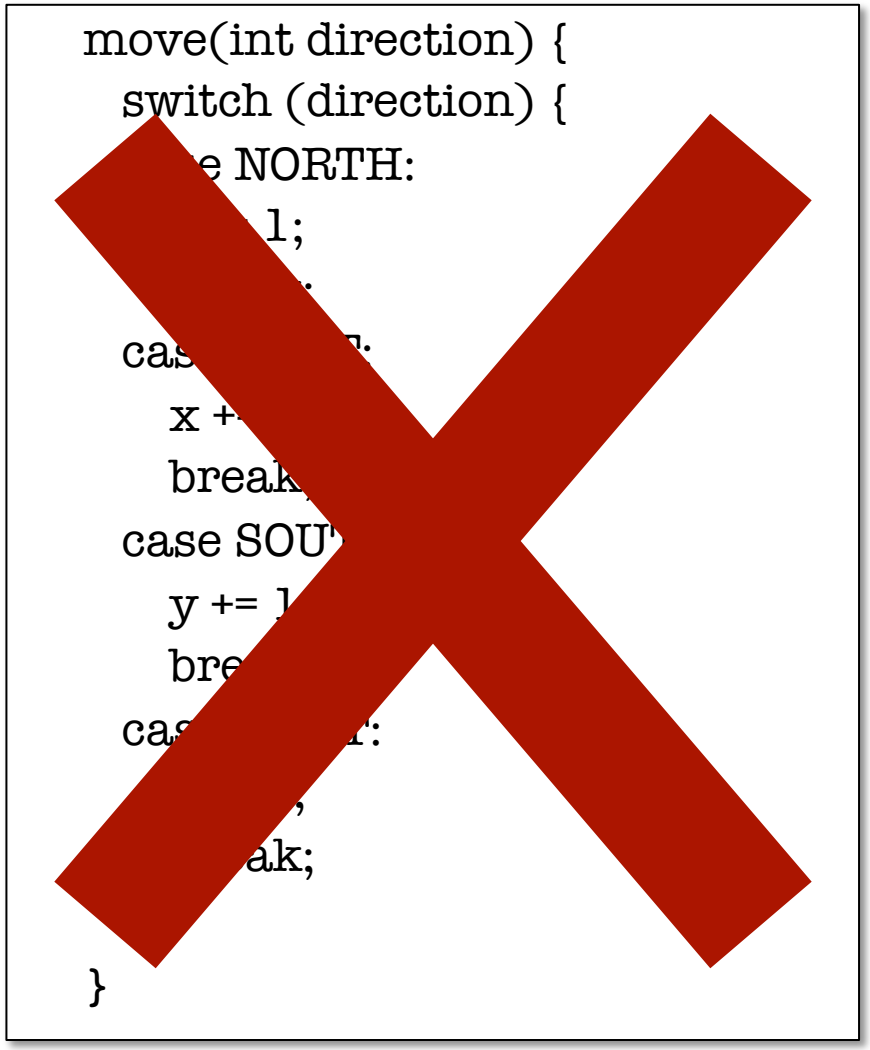
S-T-A: Separation of Logic

- **Loops** = sensing
  - Read other objects
  - *Aggregate* for thinking
  - **Example**: nearest enemy
- **Conditionals** = thinking
  - Use results of sensing
  - Switch between possibilities
  - **Example**: attack or flee
- **Assignments** = actions
  - Rarely need loops
  - Avoid conditionals

```
move(int direction) {
  switch (direction) {
  case NORTH:
    y -= 1;
    break;
  case EAST:
    x += 1;
    break;
  case SOUTH:
    y += 1;
    break;
  case WEST:
    x -= 1;
    break;
  }
}
```

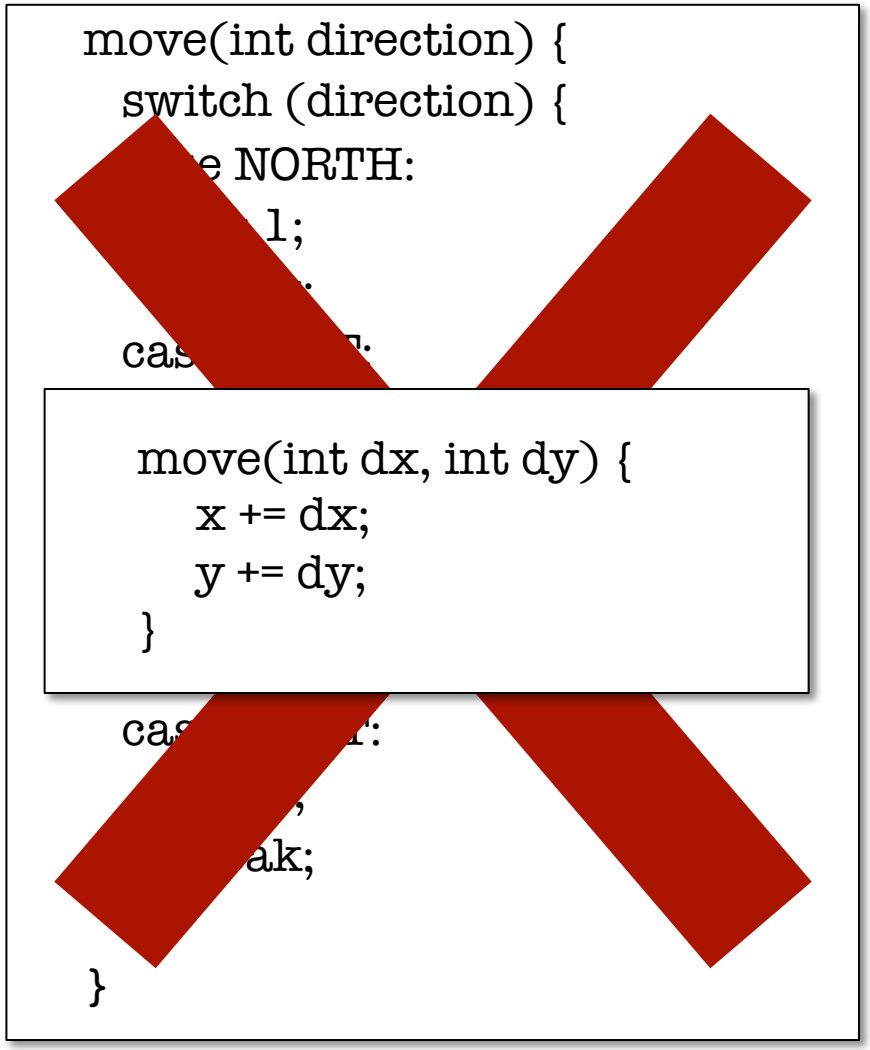
S-T-A: Separation of Logic

- **Loops** = sensing
  - Read other objects
  - *Aggregate* for thinking
  - **Example**: nearest enemy
- **Conditionals** = thinking
  - Use results of sensing
  - Switch between possibilities
  - **Example**: attack or flee
- **Assignments** = actions
  - Rarely need loops
  - Avoid conditionals



S-T-A: Separation of Logic

- **Loops** = sensing
  - Read other objects
  - *Aggregate* for thinking
  - **Example**: nearest enemy
- **Conditionals** = thinking
  - Use results of sensing
  - Switch between possibilities
  - **Example**: attack or flee
- **Assignments** = actions
  - Rarely need loops
  - Avoid conditionals

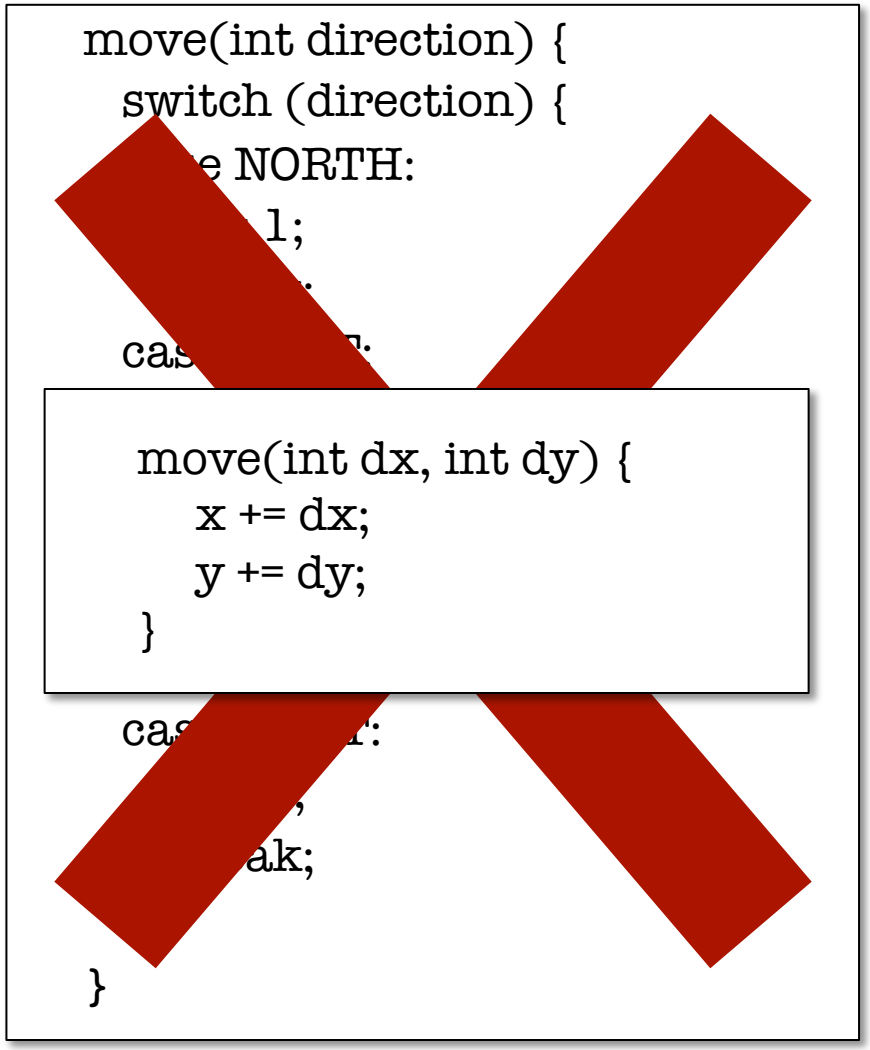


Review: Sense-Think-Act

- **Sense**:
  - Perceive the world
  - Reading the game state
  - **Example**: enemy near?
- **Think**:
  - Choose an action
  - Often merged with sense
  - **Example**: fight or flee
- **Act**:
  - Update the state
  - Simple and fast
  - **Example**: reduce health

Actions: Short and Simple

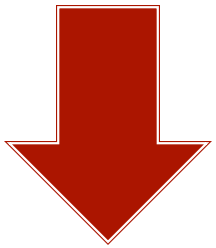
- Mainly use **assignments**
  - Avoid loops, conditionals
  - Similar to getters/setters
  - Complex code in thinking
- Helps with **serializability**
  - Record and undo actions
- Helps with **networking**
  - Keep doing last action
  - Recall: *dead reckoning*



Making Actions Serializable

Actions as Data

method(a_0, ..., a_n)



("method", a_0, ..., a_n)

or

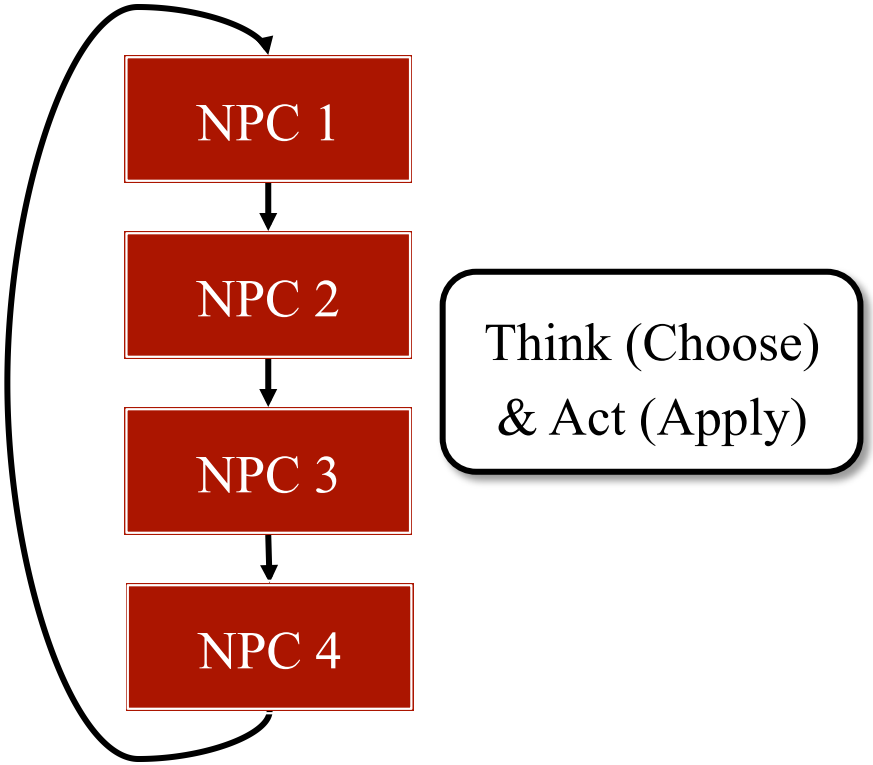
(*method, a_0, ..., a_n)

Applications

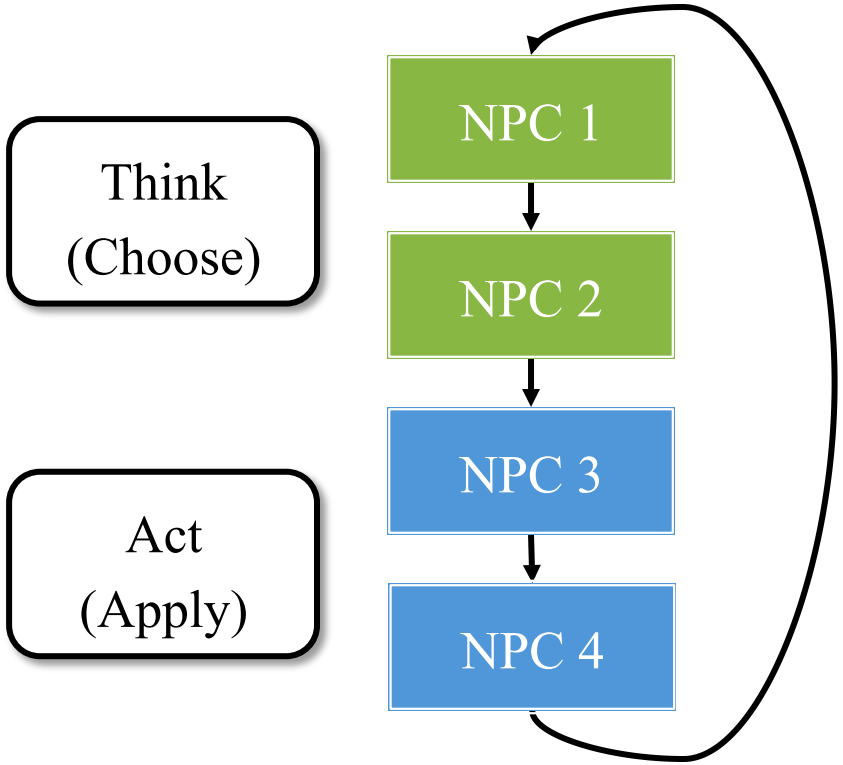
- **Cut-Scenes**
  - Sequence of actions
  - Stored in data file
- **Recorded Gameplay**
  - Repeat user actions
  - Time-travel games
- **Delayed Actions**
  - All think, then all act

Delaying Actions

Sequential Actions are Bad

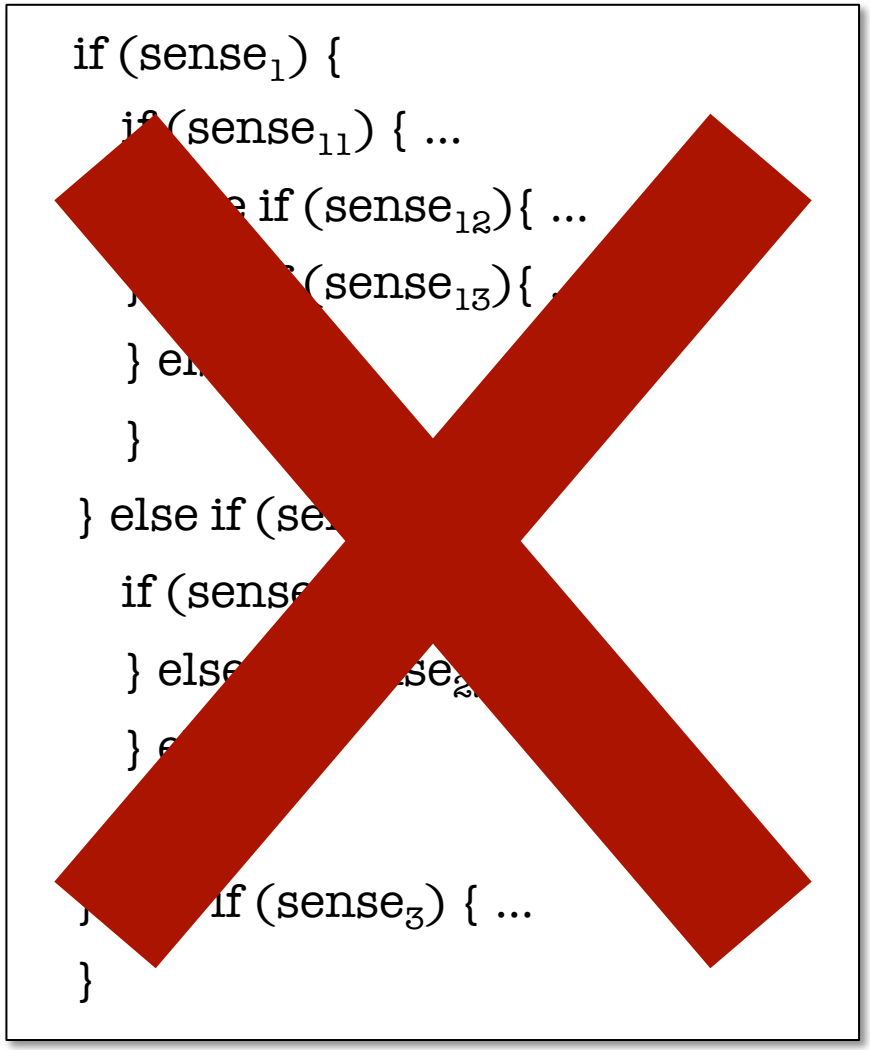


Choose Action; Apply Later



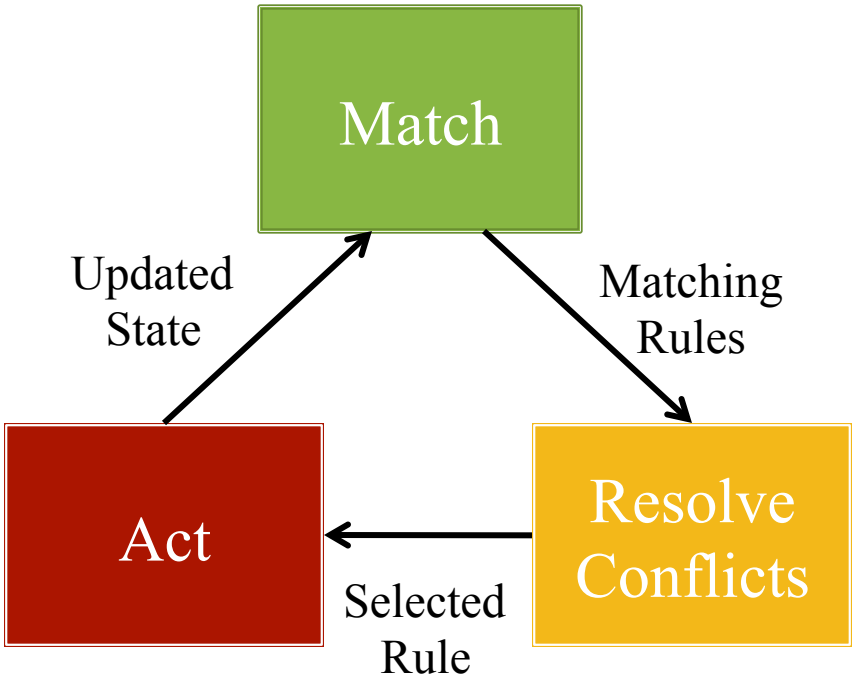
Thinking: Primary Challenge

- A mess of conditionals
  - "Spaghetti" code
  - Difficult to modify
- Abstraction requirements:
  - Easy to visualize models
  - Mirror "cognitive thought"
- Want to separate talent
  - **Sensing**: Programmers
  - **Thinking**: *Designers*
  - **Actions**: Programmers



Rule-Based AI

If *X* is true, Then do *Y*



Three-Step Process

- **Match**
  - For each rule, check **if**
  - Return *all* matches
- **Resolve**
  - Can only use one rule
  - Use metarule to pick one
- **Act**
  - Do **then**-part

Rule-Based AI

If ***X*** is true, Then do ***Y***

- **Thinking**: Providing a list of several rules
  - But what happens if there is more than one rule?
  - Which rule do we choose?

Rule-Based AI

- **Thinking**: Providing a list of several rules
  - But what happens if there is more than one rule?
  - Which rule do we choose?

Conflict Resolution

- Often **resolve by order**
  - Each rule has a priority
  - Higher priorities go first
  - "Flattening" conditionals

R_1: `if` $event_1$ `then` act_1

R_2: `if` $event_2$ `then` act_2

R_3: `if` $event_3$ `then` act_3

R_4: `if` $event_4$ `then` act_4

R_5: `if` $event_5$ `then` act_5

R_6: `if` $event_6$ `then` act_6

R_7: `if` $event_7$ `then` act_7

Conflict Resolution

- Often **resolve by order**
  - Each rule has a priority
  - Higher priorities go first
  - "Flattening" conditionals
- **Problems**:
  - Predictable
    Same events = same rules
  - Total order
    Sometimes no preference
  - Performance
    On average, go far down list

R_1: `if` $event_1$ `then` act_1

R_2: `if` $event_2$ `then` act_2

R_3: `if` $event_3$ `then` act_3

R_4: `if` $event_4$ `then` act_4

R_5: `if` $event_5$ `then` act_5

R_6: `if` $event_6$ `then` act_6

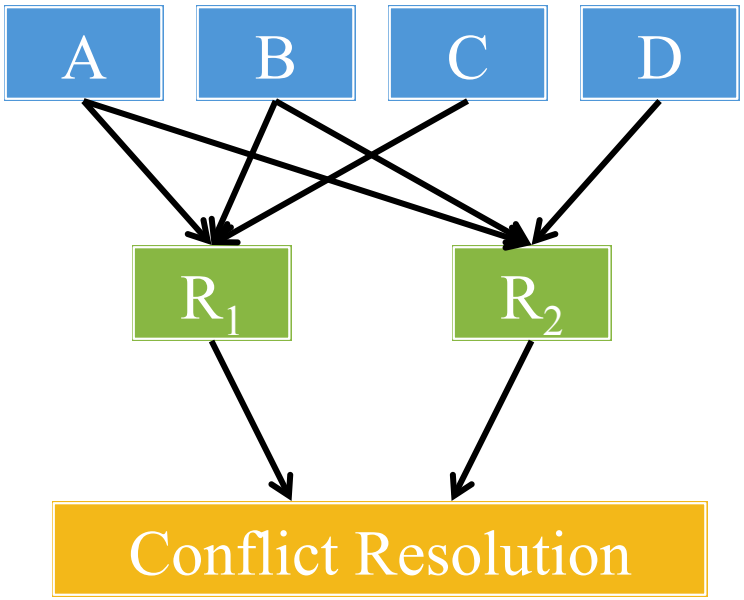
R_7: `if` $event_7$ `then` act_7

Conflict Resolution

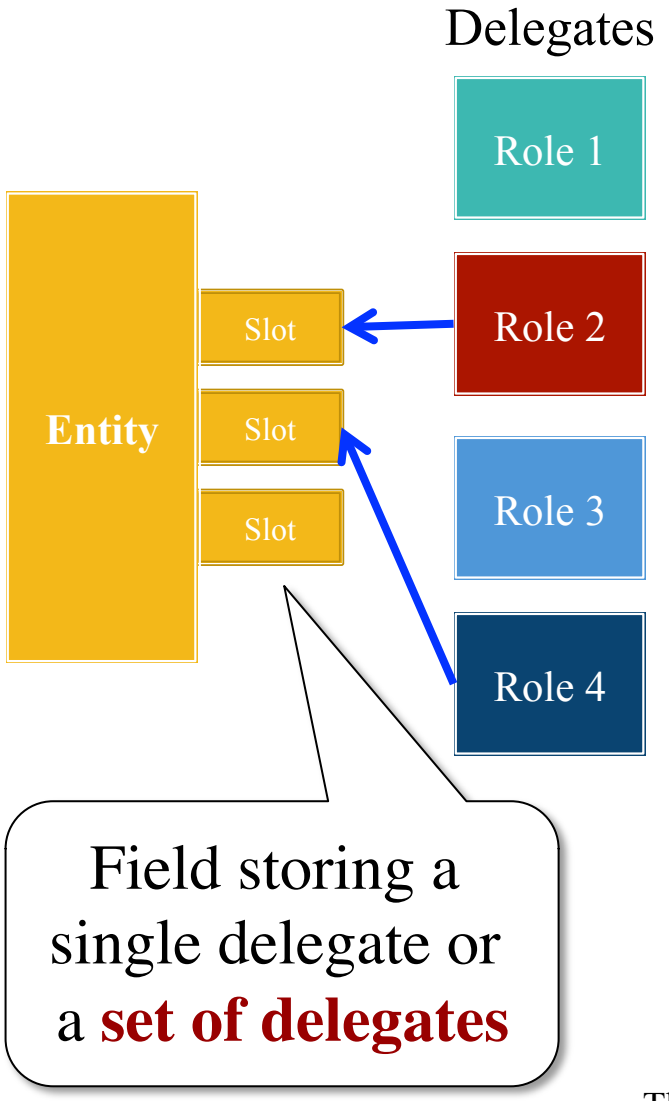
- **Specificity**:
  - Rule w/ most "components"
- **Random**:
  - Select randomly from list
  - May "weight" probabilities
- **Refractory Inhibition**:
  - Do not repeat recent rule
  - Can combine with ordering
- **Data Recency**:
  - Select most recent update

R_1: `if A, B, C, then`

R_2: `if A, B, D, then`



Components and AI



- **Act** stored in components
  - Capabilities given by roles
  - Simple functionality
- Where are **sense** & **think**?
  - In the component/role?
  - Think needs to know *all* roles to pick a single action
- How do roles interact?
  - Which action is chosen?

Treat Components Like Rules

- Each component is a rule
  - Choses one (or no) action
  - Embodies a type of sensing
  - More complex system than simple if-then rules
- **Conflict Resolution**
  - Priorities to components
  - (or priorities to actions)
  - Conflicts resolved *globally*

C_1: warrior component

C_2: archer component

C_3: human component

C_4: orc component

Implies a global AI **subsytem**

Non-Conflict Resolution?

- Some actions do *not* conflict
  - **Example**: run & shoot
  - Can we apply them both?
- Only if both **commutative**
  - Treat action as f: $\mathbb{S} \rightarrow \mathbb{S}$
  - Require $f(g(s)) = g(f(s))$
  - Easy if state is disjoint
- *Animation* is a big problem
  - Each action has animation
  - Same solution as state?

- Commutative, disjoint:
  - `move(dx,dy)`:

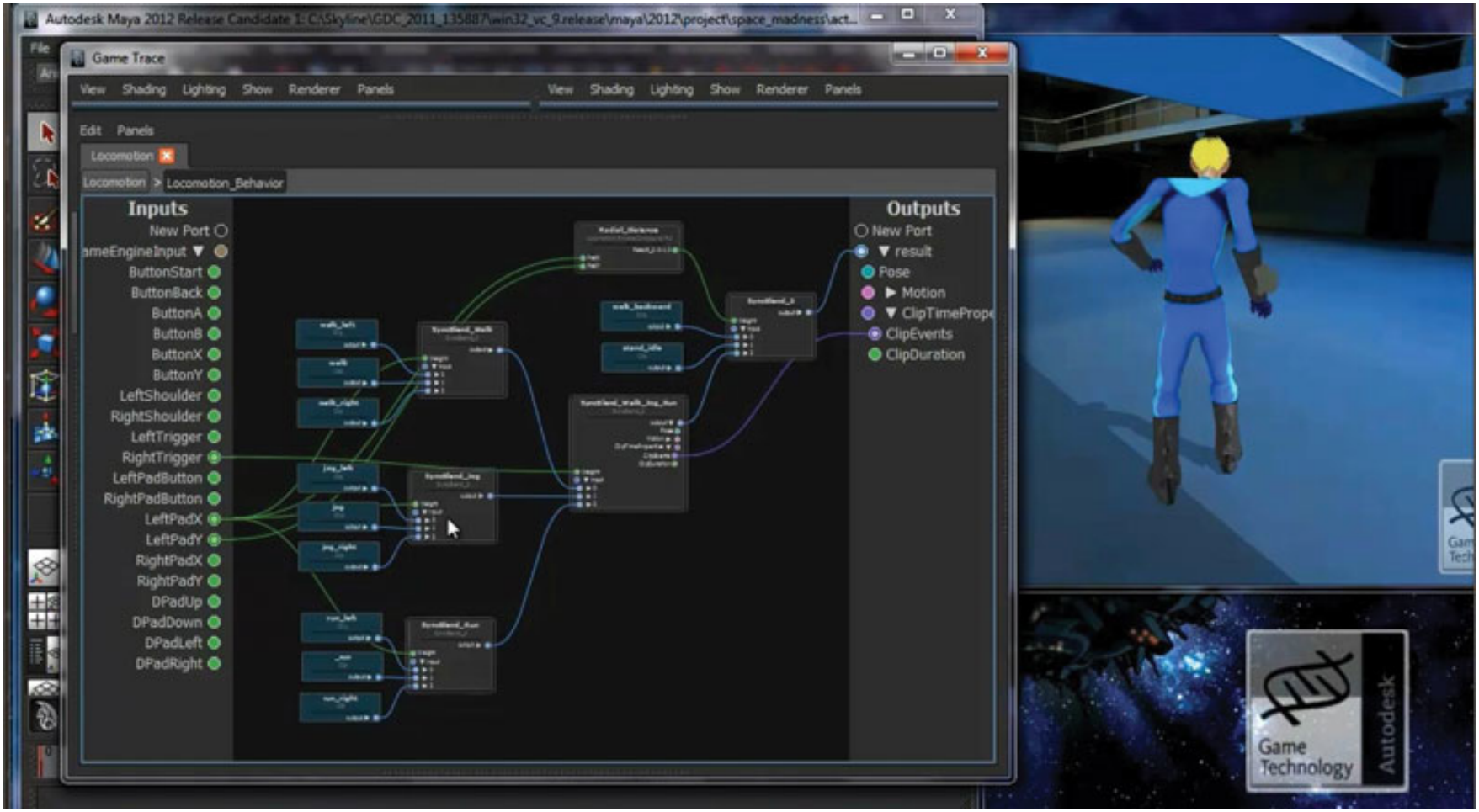
    ```
    x = x + dx
    y = y + dy
    ```

  - `damage(d)`:

    ```
    hp = hp - d
    ```

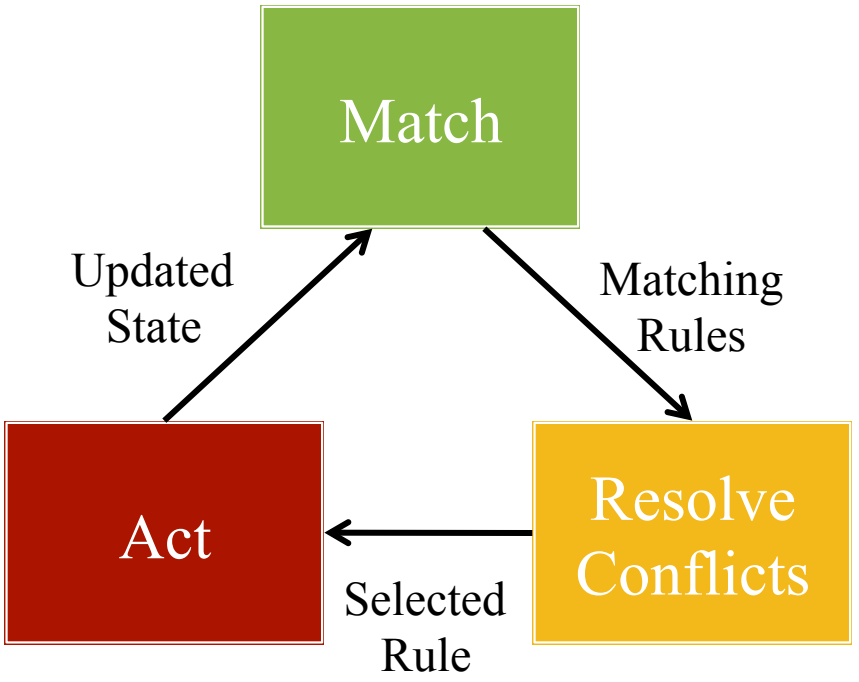
- Commutative, not disjoint:
  - 2 `move` actions
  - Addition commutes
  - **Example**: walk, push

Animation: Blend Trees



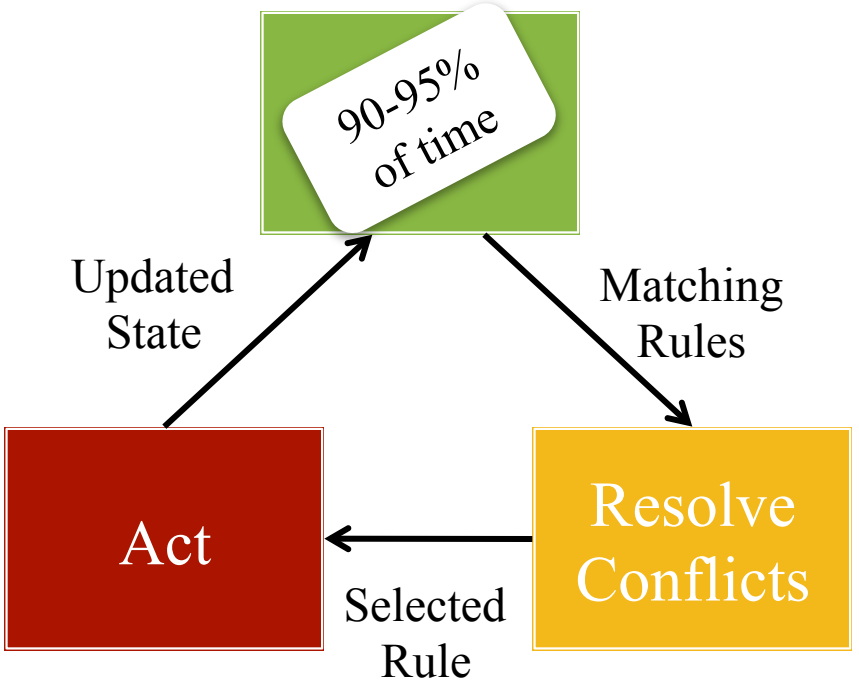
Rule-Based AI: Performance

- Matching = **sensing**
  - If-part is expensive
  - Test *every* condition
  - Many unmatched rules
- Improving performance
  - Optimize sensing (make if-part cheap)
  - Limit number of rules
  - Other solutions?
- Most games limit rules
  - Reason for *state machines*

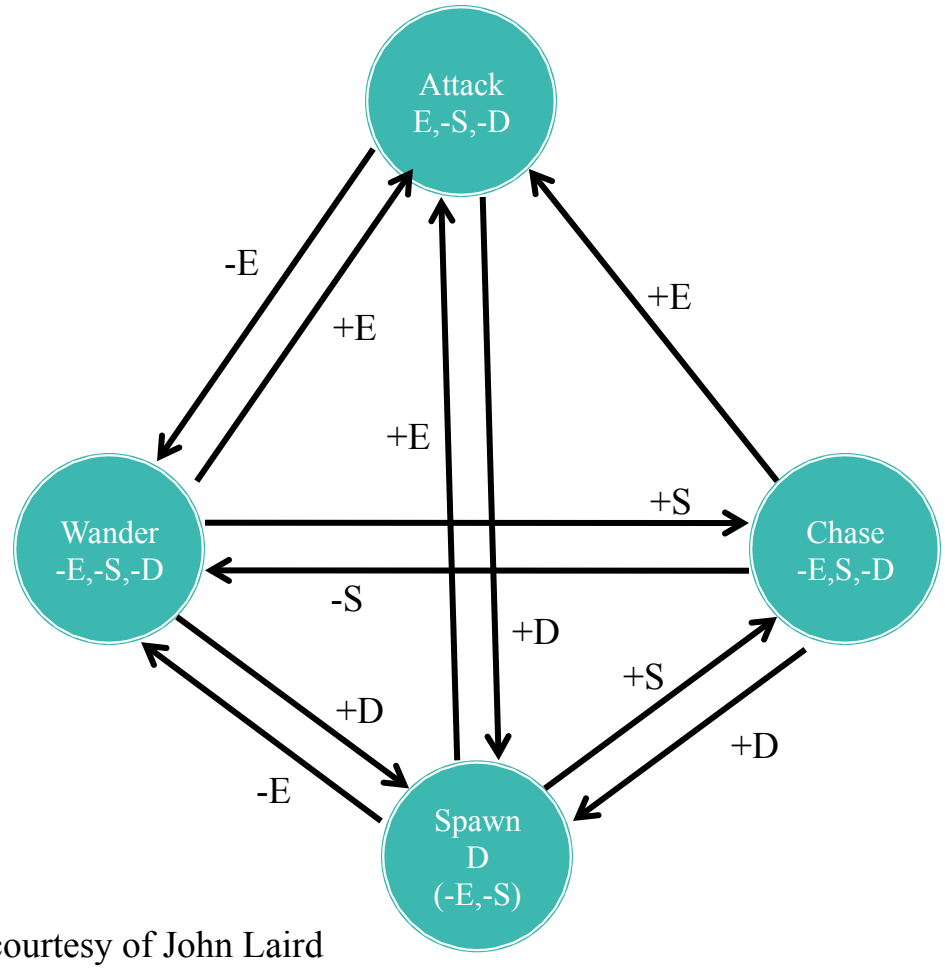


Rule-Based AI: Performance

- Matching = **sensing**
  - If-part is expensive
  - Test *every* condition
  - Many unmatched rules
- Improving performance
  - Optimize sensing (make if-part cheap)
  - Limit number of rules
  - Other solutions?
- Most games limit rules
  - Reason for *state machines*



Finite State Machines

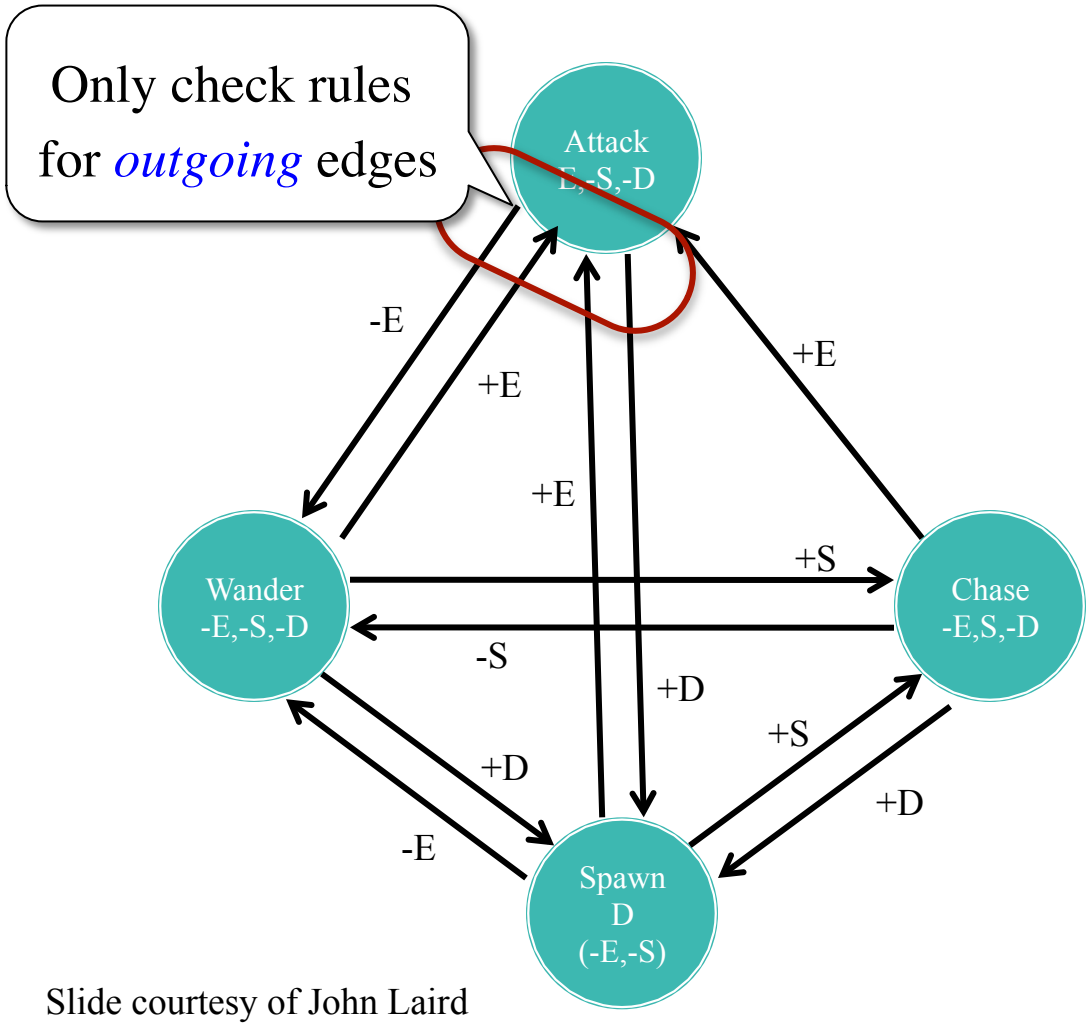


Events

- **E**=Enemy Seen
- **S**=Sound Heard
- **D**=Die

Slide courtesy of John Laird

Finite State Machines



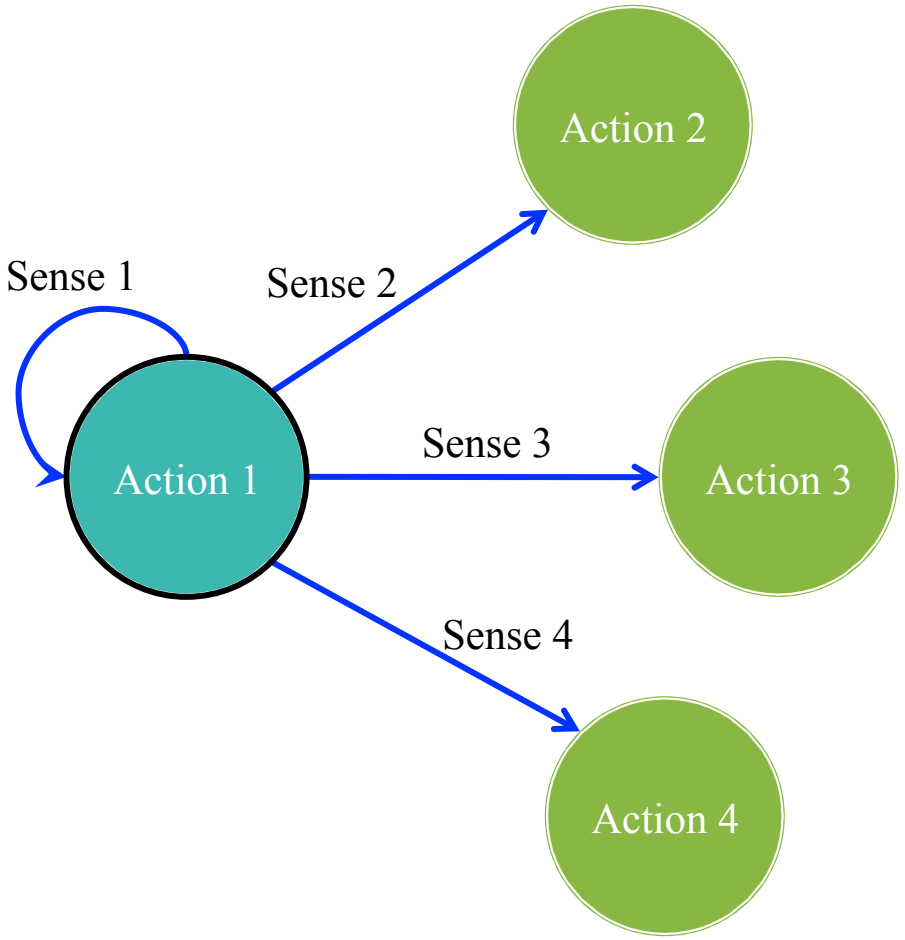
Events

- **E**=Enemy Seen
- **S**=Sound Heard
- **D**=Die

Slide courtesy of John Laird

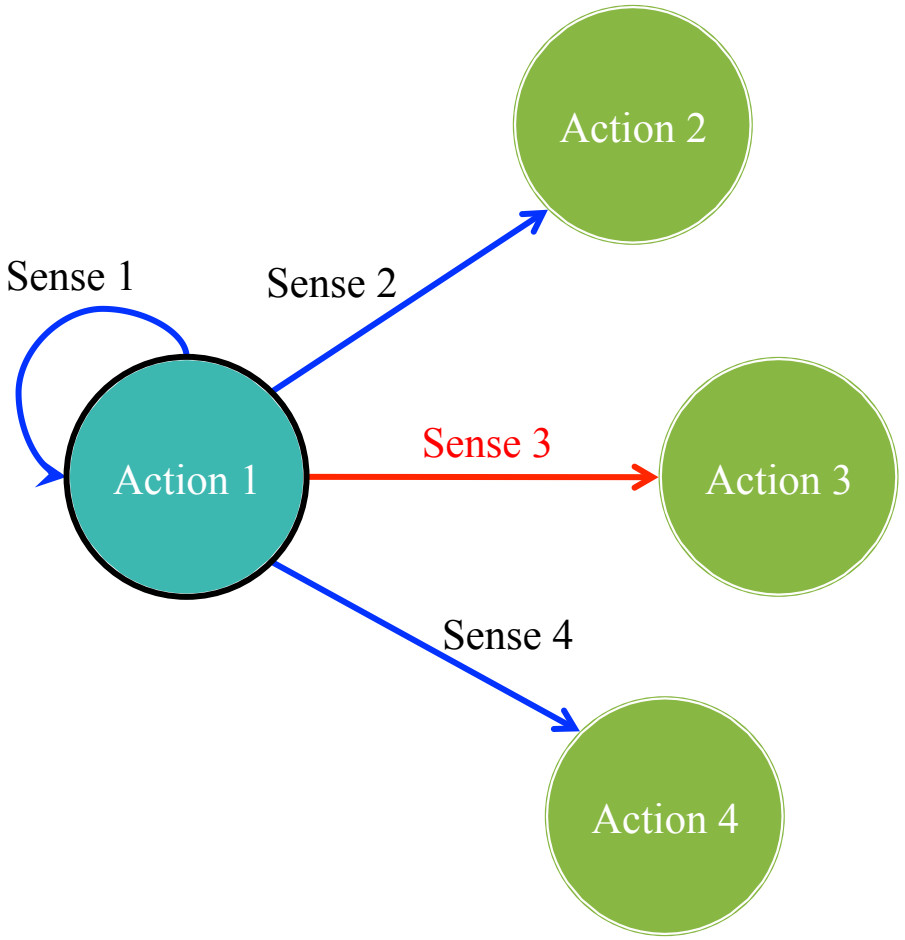
Finite State Machines

- **Transitions**: Senses
  - Each edge has test condition
  - Traverse edge that is true
  - Conflict resolution for ties
  - Stop once reach new state
- **States**: Actions
  - Action of past frame
  - Traverse to next state
  - Defines action this frame
- **Localized** form of rule AI



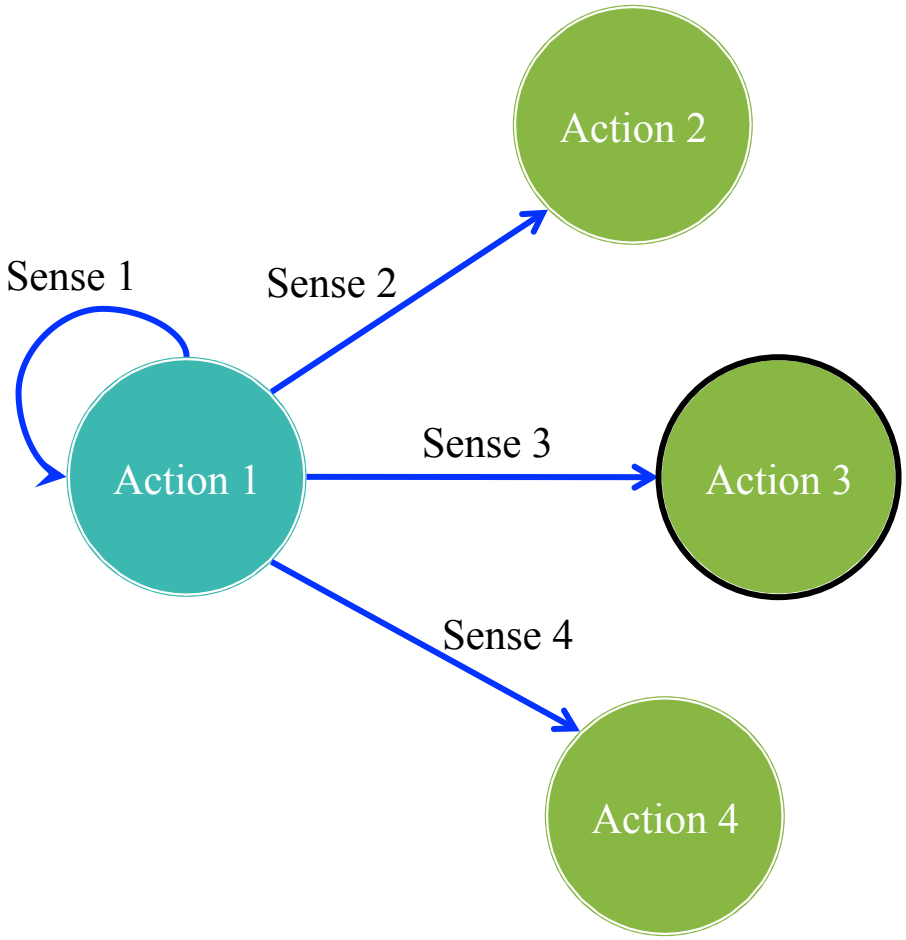
Finite State Machines

- **Transitions**: Senses
  - Each edge has test condition
  - Traverse edge that is true
  - Conflict resolution for ties
  - Stop once reach new state
- **States**: Actions
  - Action of past frame
  - Traverse to next state
  - Defines action this frame
- **Localized** form of rule AI



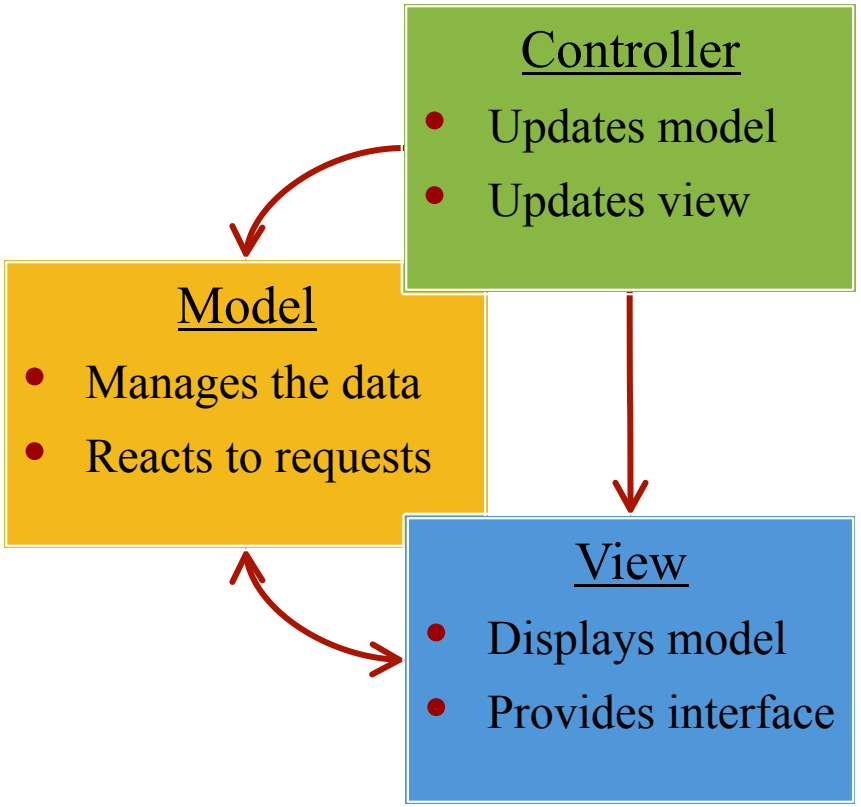
Finite State Machines

- **Transitions**: Senses
  - Each edge has test condition
  - Traverse edge that is true
  - Conflict resolution for ties
  - Stop once reach new state
- **States**: Actions
  - Action of past frame
  - Traverse to next state
  - Defines action this frame
- **Localized** form of rule AI



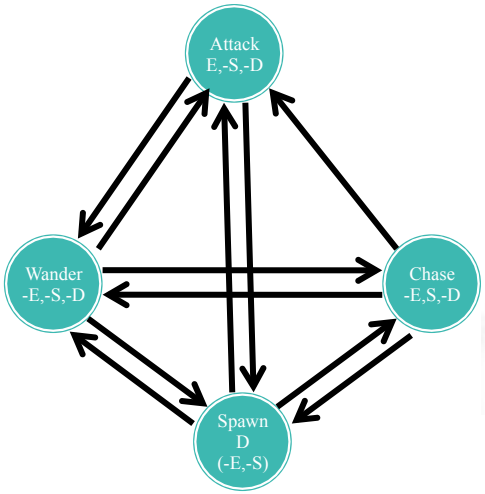
Implementation: Model-View-Controller

- Games have **thin** models
  - Methods = get/set/update
  - Controllers are heavyweight
- AI is a **controller**
  - Uniform process over NPCs
- But behavior is *personal*
  - Diff. NPCs = diff. behavior
  - Do not want unique code
- What can we do?
  - Data-Driven Design



Implementation: Model-View-Controller

- **Actions** go in the model
  - Lightweight updates
  - Specific to model or role
- Controller is framework for general **sensing**, **thinking**
  - Standard FSM engine
  - Or FSM alternatives (later)
- **Process** stored in a model
  - Represent thinking as *graph*
  - Controller processes graph



wander

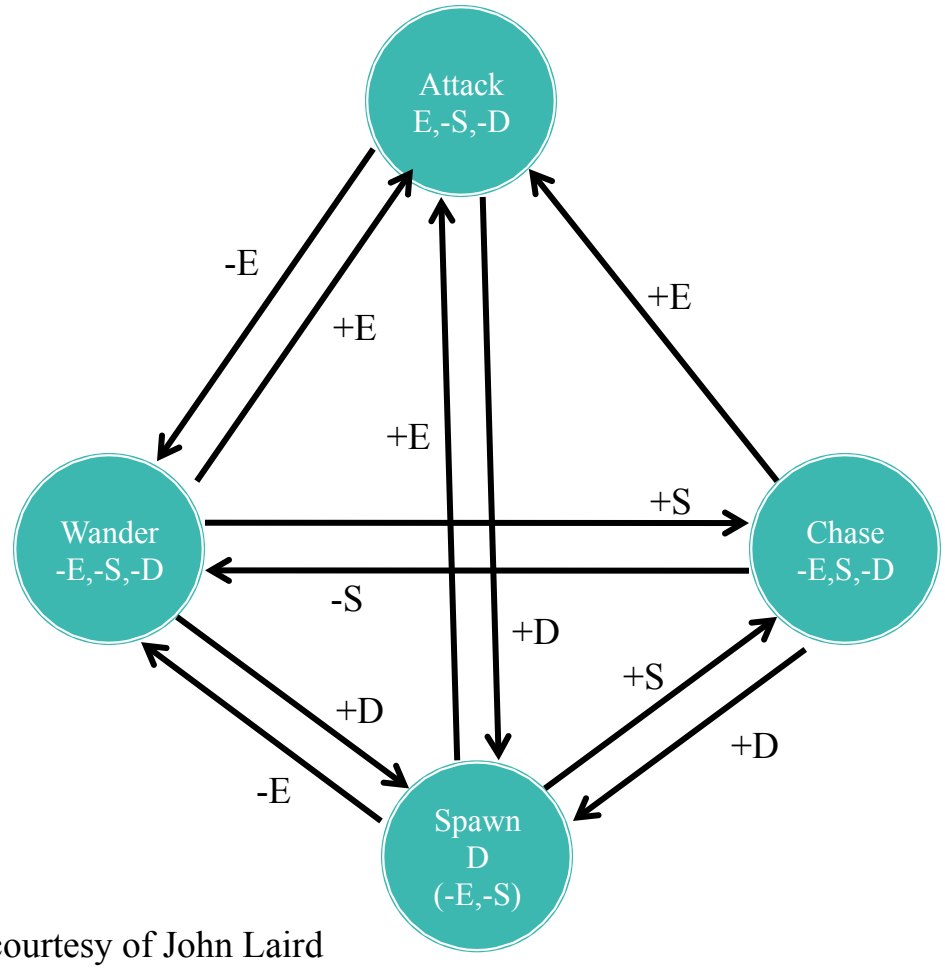
chase

attack

spawn

AI Controller

Problems with FSMs



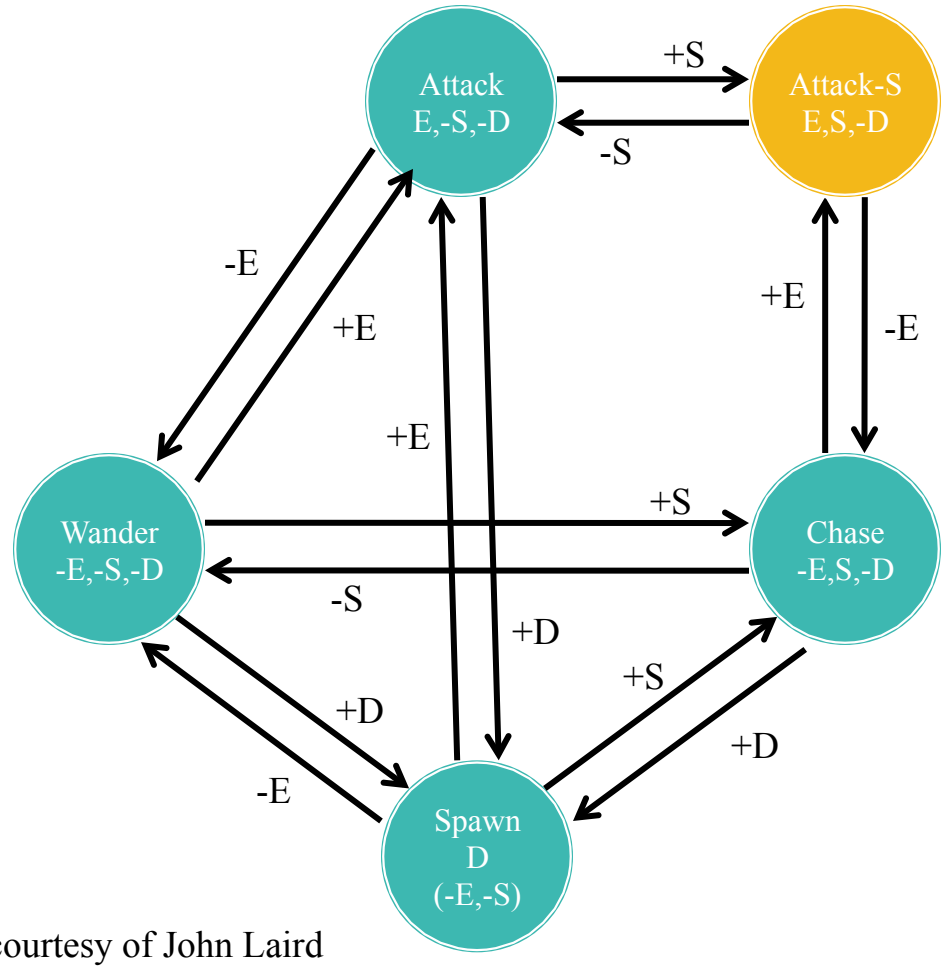
Events

- **E**=Enemy Seen
- **S**=Sound Heard
- **D**=Die

No edge from **Attack** to **Chase**

Slide courtesy of John Laird

Problems with FSMs



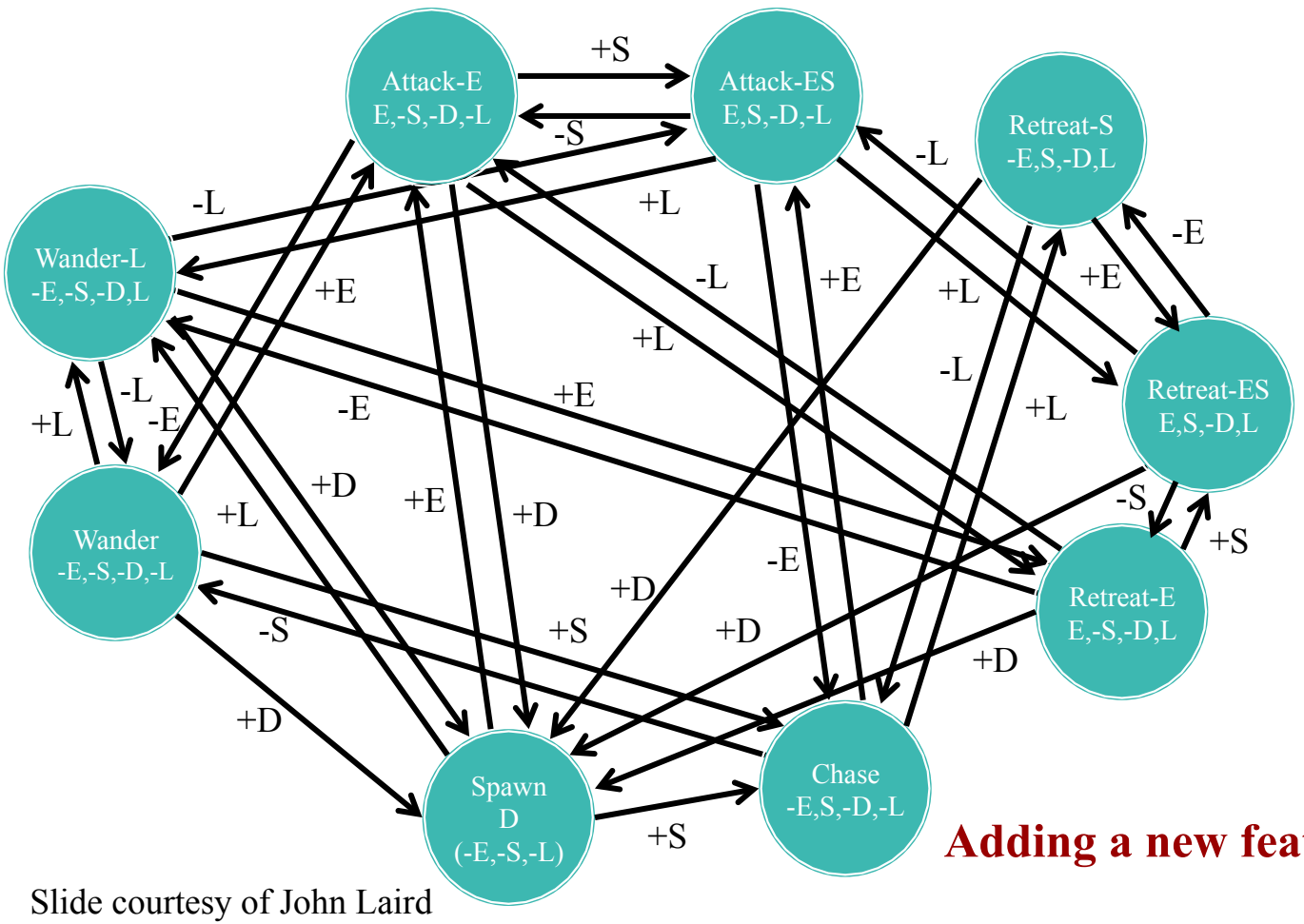
Events

- **E**=Enemy Seen
- **S**=Sound Heard
- **D**=Die

Requires a *redundant* state

Slide courtesy of John Laird

Problems with FSMs



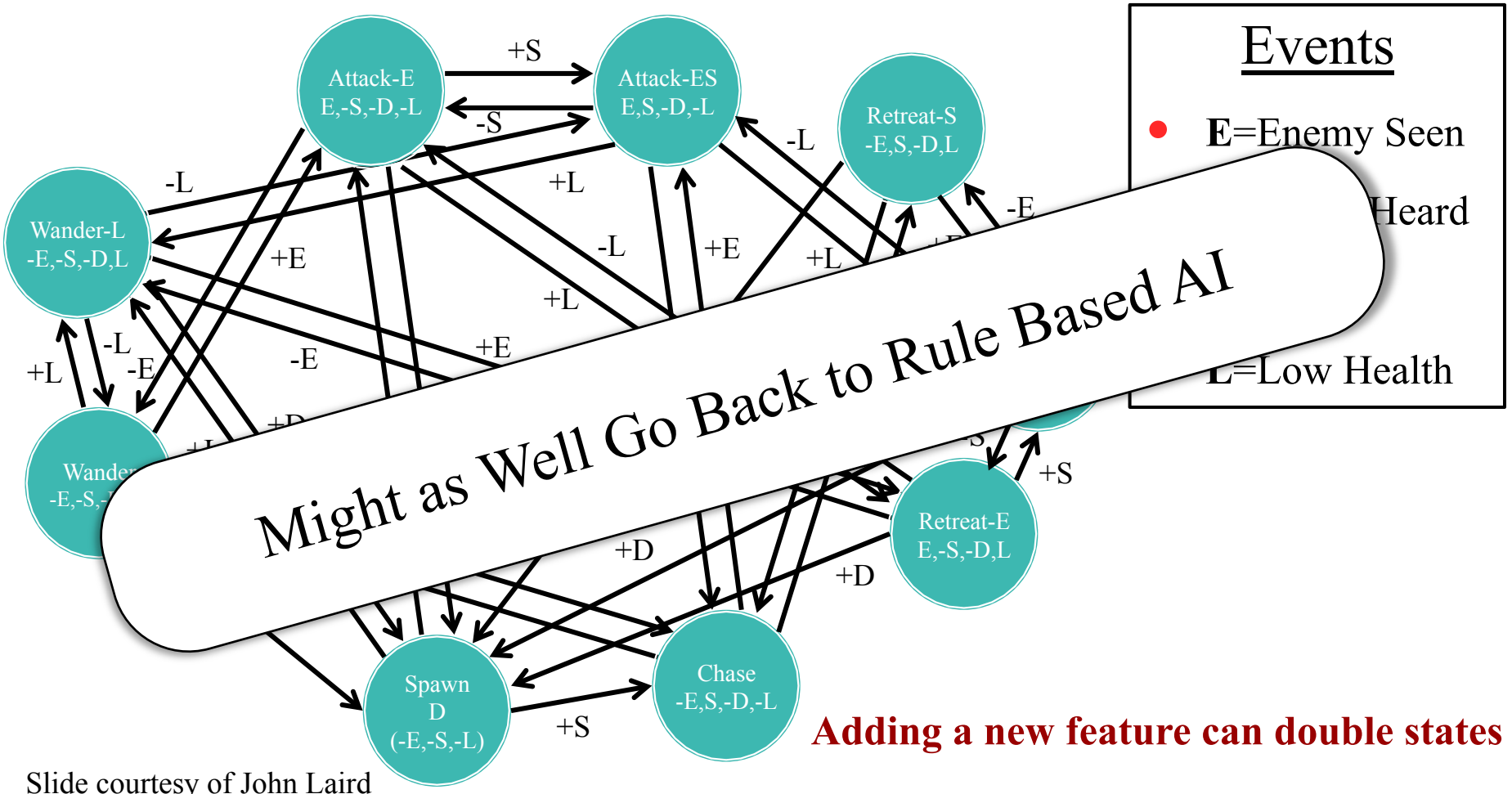
Events

- **E**=Enemy Seen
- **S**=Sound Heard
- **D**=Die
- **L**=Low Health

Adding a new feature can double states

Slide courtesy of John Laird

Problems with FSMs



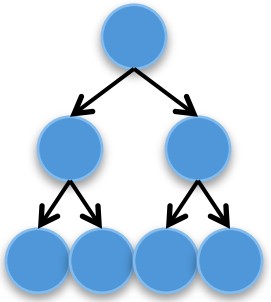
Slide courtesy of John Laird

An Observation

- Each state has a set of **global attributes**
  - Different attributes may have same actions
  - Reason for redundant behavior
- Currently just cared about attributes
  - Not really using the full power of a FSM
  - Why don't we just check attributes directly?
- Attribute-based selection: *decision trees*

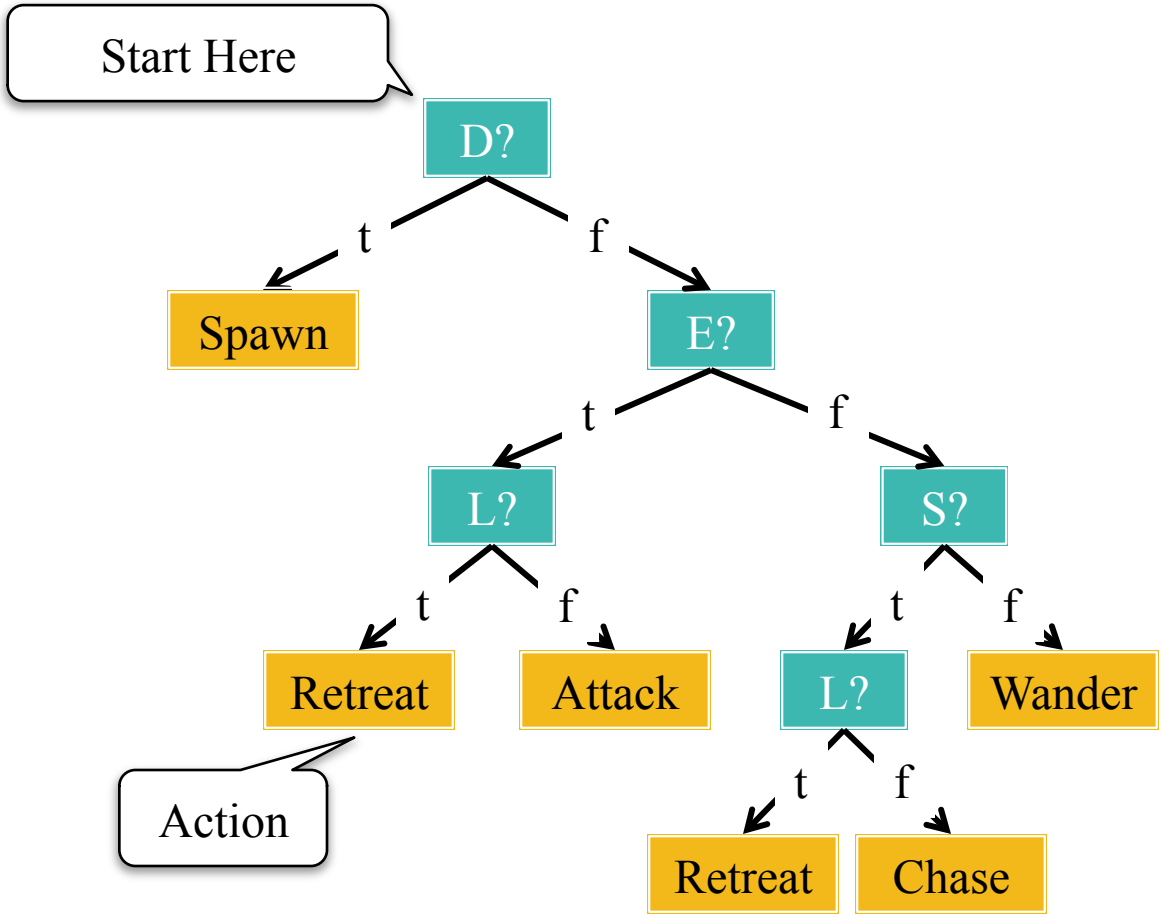
Decision Trees

- Thinking **encoded as a tree**
  - Attributes = tree nodes
  - Left = true, right = false
  - Actions = leaves (reach from the root)



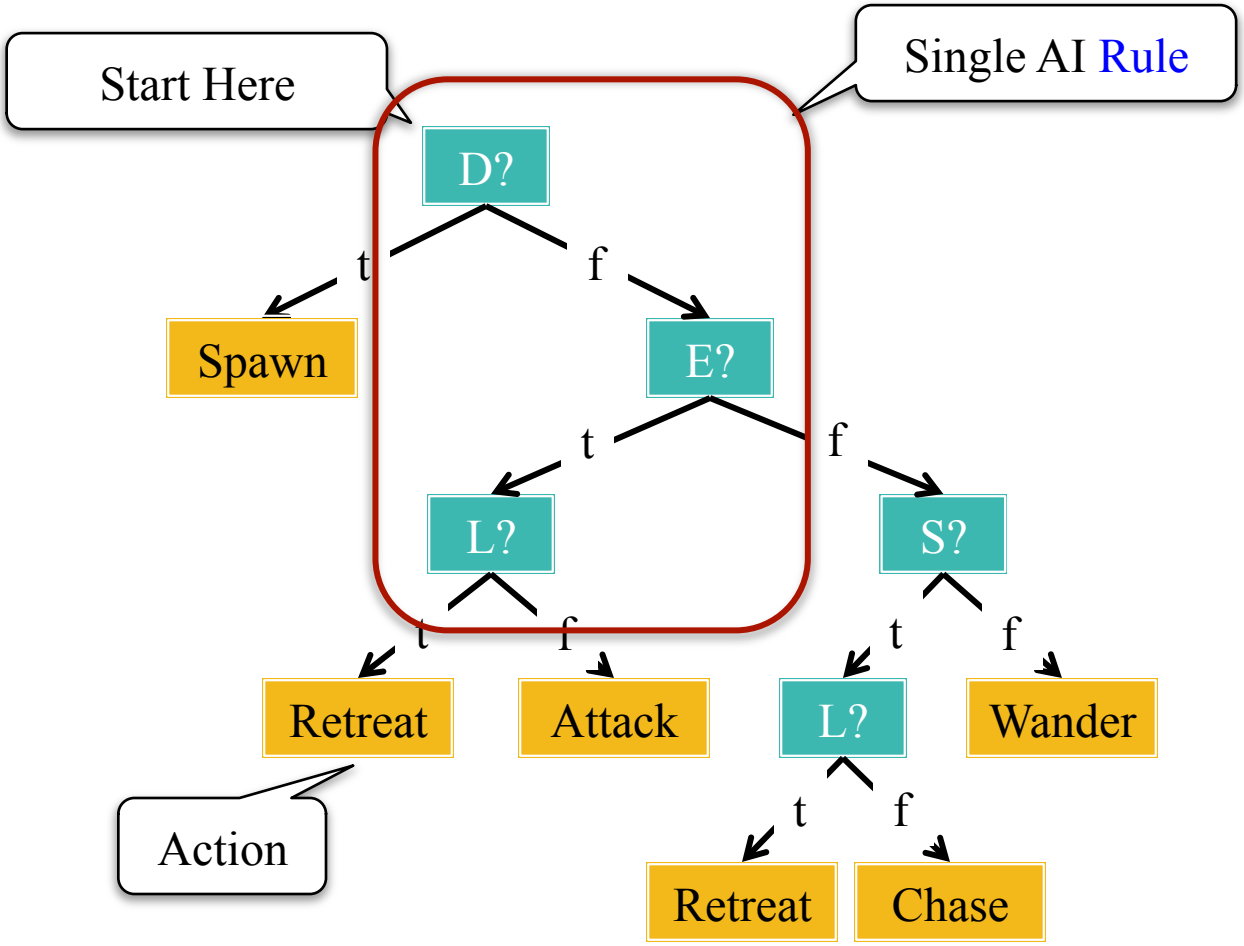
- Classify by **descending** from root to a leaf
  - Start with the test at the root
  - Descend the branch according to the test
  - Repeat until a leaf is reached

Decision Tree Example



Slide courtesy of John Laird

Decision Tree Example

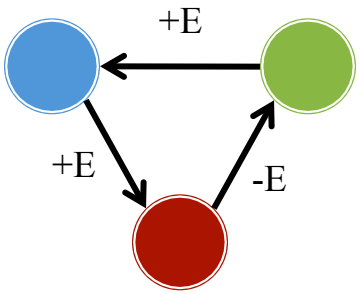


Slide courtesy of John Laird

FSMs vs. Decision Trees

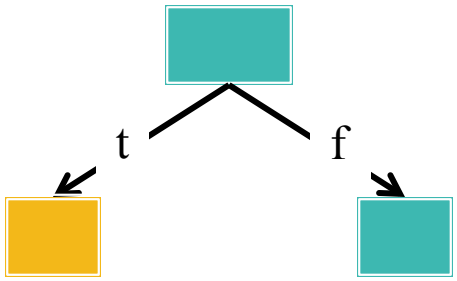
Finite State Machines

- Not limited to attributes
- Allow "arbitrary" behavior
- Explode in size very fast

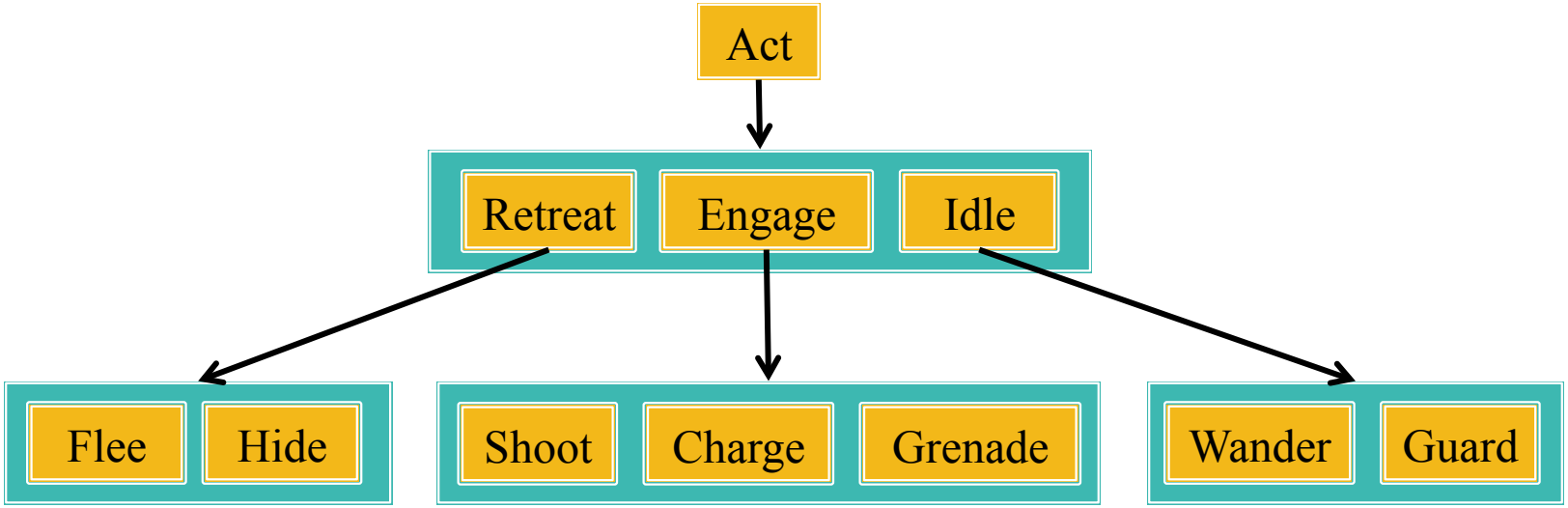


Decision Trees

- Only attribute selection
- Much more manageable
- Mixes w/ machine learning

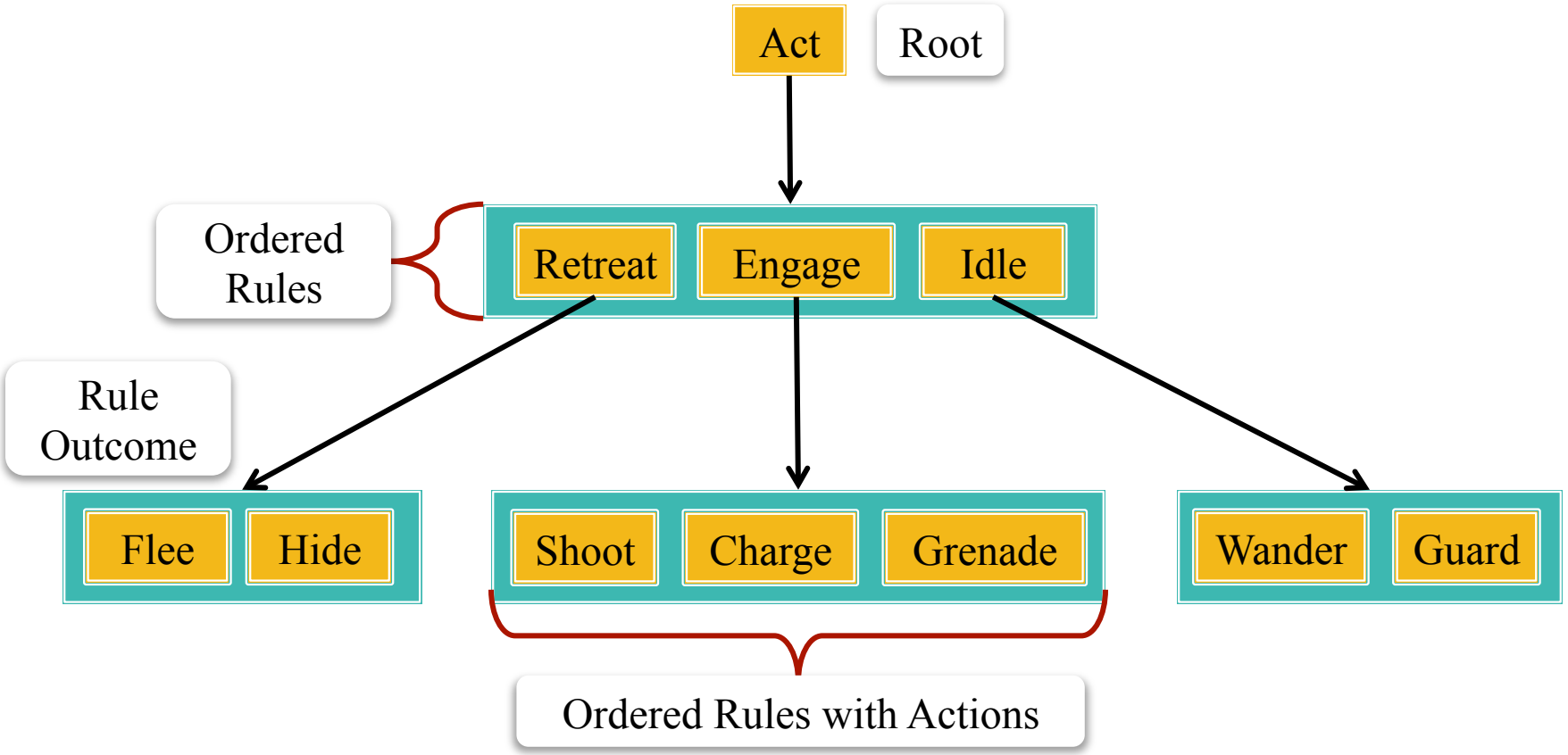


Behavior Trees

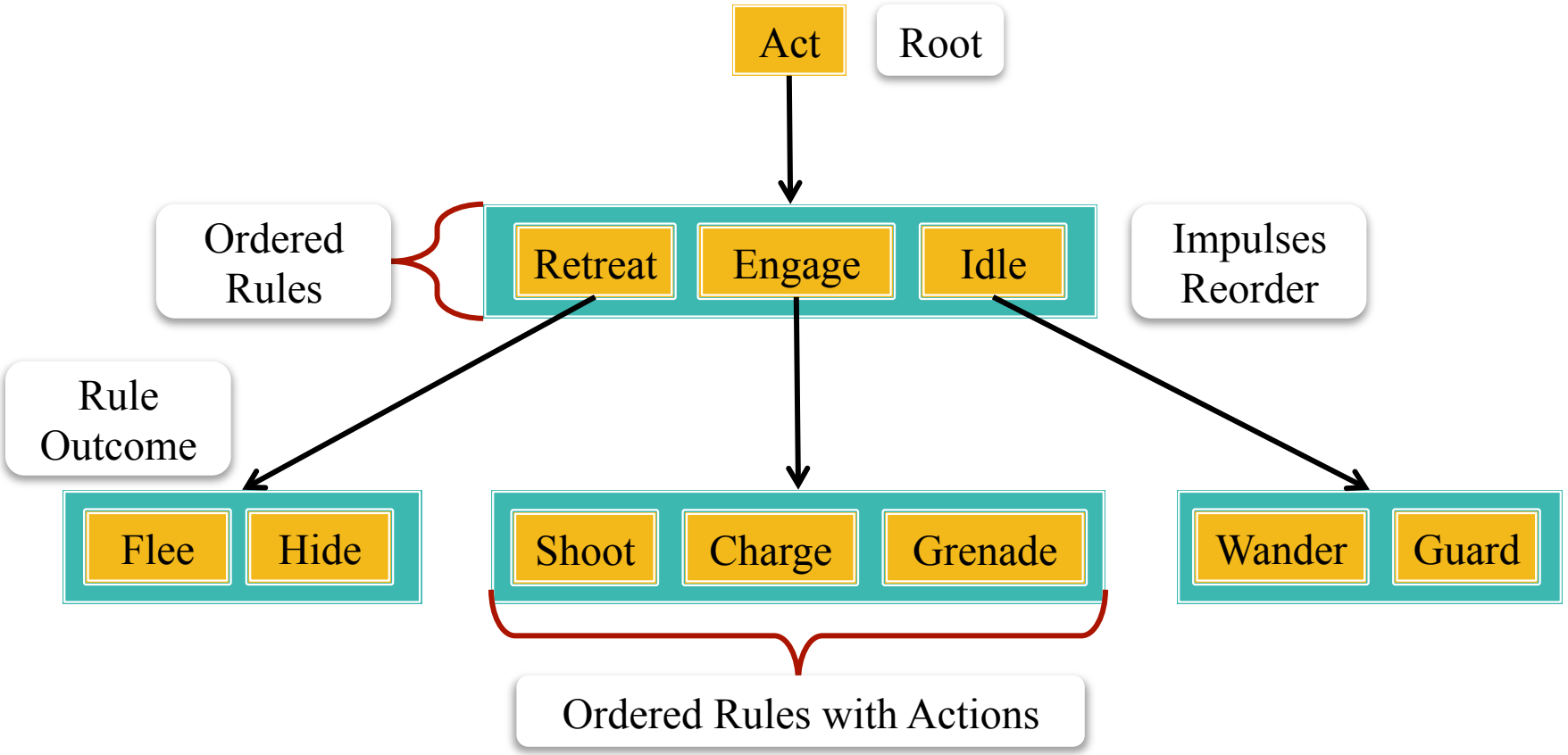


- Part rule-based
- Part decision tree
- Freedom of FSM (almost)
- Node is a list of *actions*
- Select action using *rules*
- Action leads to *subactions*

Behavior Trees

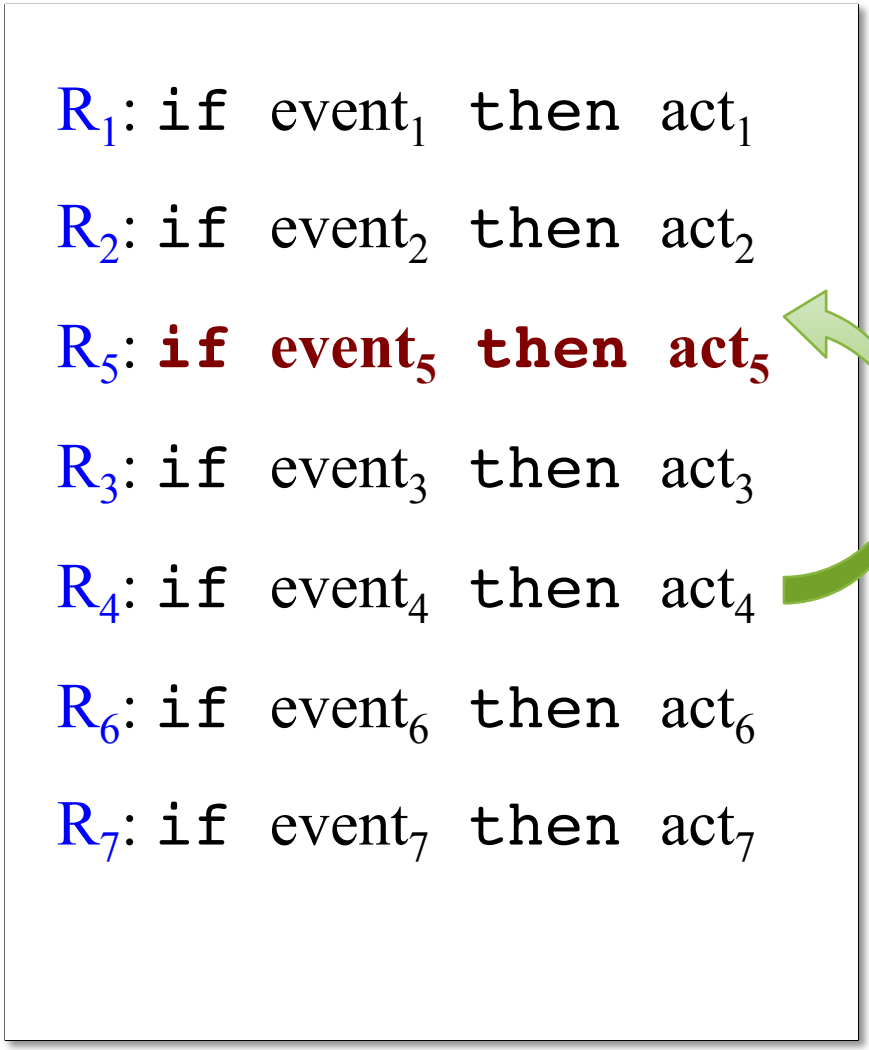


Behavior Trees



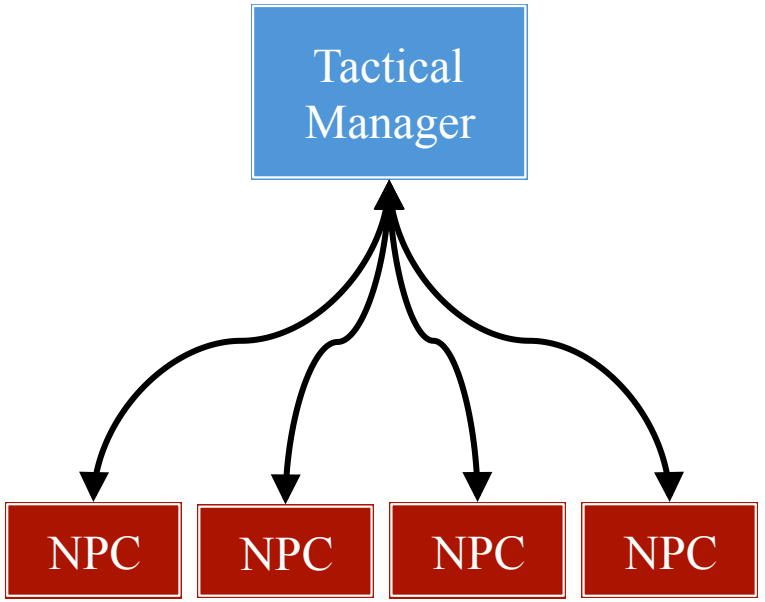
Impulses

- Correspond to certain events
  - **Global**: not tied to NPC
  - Must also have duration
- Used to **reorder** rules
  - Event makes rule important
  - Temporarily up the priority
  - Restore when event is over
- Great with behavior trees
  - Reduces size of tree
  - Used in *Halo* 3 AI.

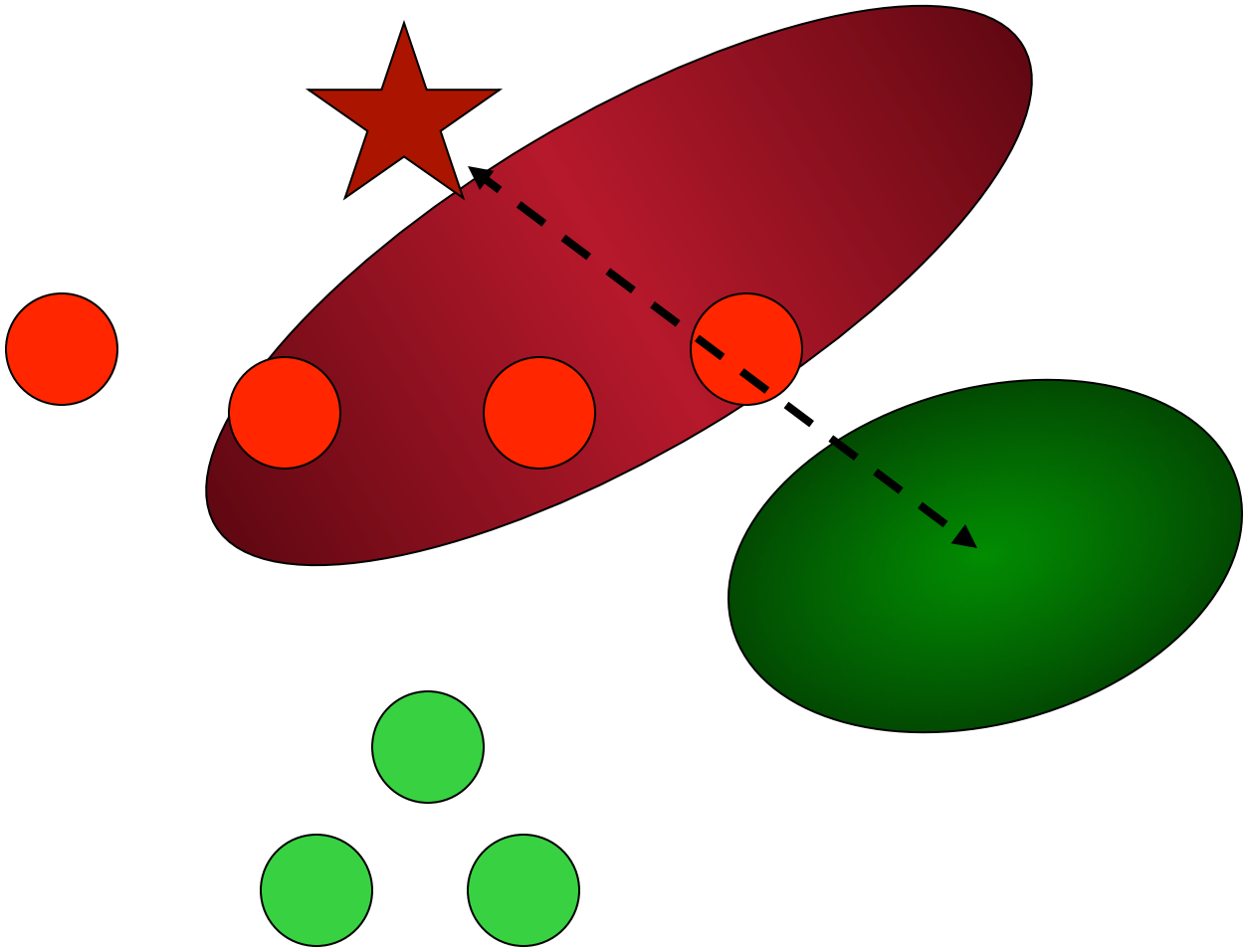


Tactical Managers

- "Invisible NPC"
  - Assigned to NPC Group
  - Performs all *thinking*
  - NPCs just follow orders
- **Applications**
  - Protecting special units
  - Flanking
  - Covering fire
  - Leapfrogging advance

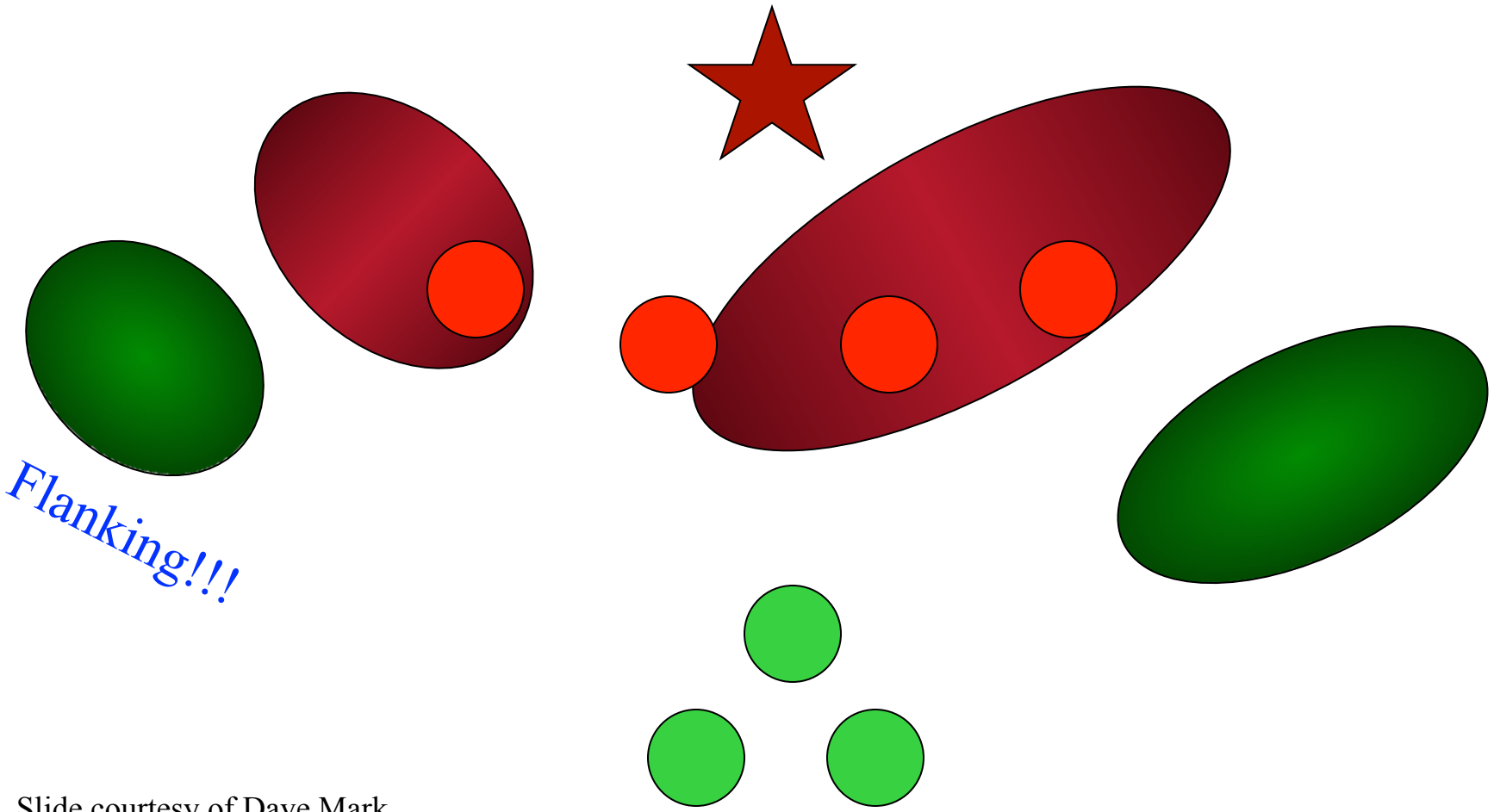


Protecting Special Units



Slide courtesy of Dave Mark

Protecting Special Units



Slide courtesy of Dave Mark

Summary

- Character AI is a **software engineering** problem
  - Sense-think-act aids code reuse and ease of design
  - Least standardized aspect of game architecture
- **Rule-based AI** is the foundation for all character AI
  - Simplified variation of sense-think-act
  - Alternative systems made to limit number of rules
- Games use **graphical models** for data-driven AI
  - Controller outside of NPC model processes AI
  - Graph stored in NPC model tailors AI to individuals