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

```java
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;
    }!
}

move(int dx, int dy) {
    x += dx;
    y += dy;
}
```

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**

```java
move(int dx, int dy) {
    x += dx;
    y += dy;
}
```
Making Actions Serializable

Actions as Data

method(a₀, ..., aₙ)

("method", a₀, ..., aₙ)

or

(*method, a₀, ..., aₙ)

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
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

```java
if (sense_1) {
    if (sense_11) { ... 
    else if (sense_12) { ... 
    } else if (sense_13) { ... 
    } else { ... 
}
} else if (sense_2) {
    if (sense_21) { ... 
    else if (sense_22) { ... 
    } else { ... 
    }
} else if (sense_3) { ... 
    if (sense_31) { ... 
    } else if (sense_32) { ... 
    } else { ... 
    }
} else { ... 
```
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?
# 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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>event₁</td>
<td>act₁</td>
</tr>
<tr>
<td>R₂</td>
<td>event₂</td>
<td>act₂</td>
</tr>
<tr>
<td>R₃</td>
<td>event₃</td>
<td>act₃</td>
</tr>
<tr>
<td>R₄</td>
<td>event₄</td>
<td>act₄</td>
</tr>
<tr>
<td>R₅</td>
<td>event₅</td>
<td>act₅</td>
</tr>
<tr>
<td>R₆</td>
<td>event₆</td>
<td>act₆</td>
</tr>
<tr>
<td>R₇</td>
<td>event₇</td>
<td>act₇</td>
</tr>
</tbody>
</table>
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

\[ R_1: \text{if } A, B, C, \text{ then} \]
\[ R_2: \text{if } A, B, D, \text{ then} \]
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?

Field storing a single delegate or a **set of delegates**
Treat Components Like Rules

- Each component is a rule
- Chooses 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 \( f: S \rightarrow S \)
  - 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**:
  - \( \text{move}(dx,dy):\)
    - \( x = x + dx \)
    - \( y = y + dy \)
  - \( \text{damage}(d):\)
    - \( \text{hp} = \text{hp} - d \)
- 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*

Diagram:
- 90-95% of time
- Updated State
- Matching Rules
- Resolve Conflicts
- Act
- Selected Rule
Finite State Machines

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

List of States and Transitions:
- Attack: E, -S, -D
- Wander: -E, -S, -D
- Chase: -E, S, -D
- Spawn D: (-E, -S)

Events:
- E = Enemy Seen
- S = Sound Heard
- D = Die
Finite State Machines

Only check rules for *outgoing* edges

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

- **Model**
  - Manages the data
  - Reacts to requests

- **Controller**
  - Updates model
  - Updates view

- **View**
  - Displays model
  - Provides interface
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
Problems with FSMs

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

No edge from Attack to Chase
Problems with FSMs

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

Requires a redundant state
Problems with FSMs

Adding a new feature can double states

Events
- E = Enemy Seen
- S = Sound Heard
- D = Die
- L = Low Health
Problems with FSMs

Adding a new feature can double states

Might as Well Go Back to Rule Based AI

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

Wander-L
- E, S, D, L

Attack-E
E, S, D, L

Attack-ES
E, S, D, L

Retreat-E
E, S, D, L

Retreat-S
E, S, D, L

Chase
E, S, D, L

Spawn D
(-E, S, D, L)

Might as Well Go Back to Rule Based AI
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
  - Start with the test at the root
  - Descend the branch according to the test
  - Repeat until a leaf is reached
Decision Tree Example

Start Here

D?

D? t f

Spawn E?

E? t f

L? S? Wander

L? t f

Retreat Attack

Retreat t f

Chase
Decision Tree Example

Start Here

D?

E?

L?

S?

Single AI Rule

Spawn

Retreat

Attack

L?

Wander

Retreat

Chase

Action

Start Here
**Finite State Machines**

- Not limited to attributes
- Allow “arbitrary” behavior

**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

Ordered Rules

Retreat  Engage  Idle

Impulses
Reorder

Wander  Guard

Ordered Rules with Actions

Rule Outcome

Flee  Hide

Act  Root

Thinking and Acting
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.

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
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

Thinking and Acting
Protecting Special Units

Flanking!!!

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