Lecture 22

Strategic AI
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)
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
Example: Tic-Tac-Toe

- Next move for player O?
  - If have a winning move, make it
  - If opponent can win, block it
  - Take the center if available
  - Corners are better than edges

- Very easy to program
  - Just check the board state
  - Tricky part is prioritization
Example: Real Time Strategy

Example from Microsoft’s *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 ? 1100))

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

Strategic AI
The Problems with Rules

- Rules only do one step
  - May not be best move
  - Could lose long term
- Next move for player O?
  - If can win, then do it
  - If X can win, then block it
  - Take the center if possible
  - Corners > edges
- Need to **look ahead**
The Problems with Rules

- Rules only do one step
  - May not be best move
  - Could lose long term
- Next move for player O?
  - If can win, then do it
  - If X can win, then block it
  - Take the center if possible
  - Corners > edges
- Need to **look ahead**
Multiple Steps: Planning

- **Plan**: actions necessary to reach a goal
  - Goal is a (pseudo) specific game state
  - Actions change game state (e.g. verbs)

- **Planning**: steps to generate a plan
  - **Initial State**: state the game is currently in
  - **Goal Test**: determines if state meets goal
  - **Operators**: action the NPC can perform
What Should We Do?

- Pickup?
- Shoot?
- Pickup?
Identify desired goal
- Ex: Kill enemy, get gold
- Design appropriate test

List all relevant actions
- Ex: Build, send troops

Look-ahead Search
- Start with initial state
- Try all actions (look-ahead)
- Stop if reached goal
- Continue if not at goal

Simplification: No Opponent

Tree Search
Planning Issues

- **Exponential** choices
  - Search action *sequences*
  - How far are we searching?
  - Cannot do this in real life!

- Game state is **complex**
  - Do we look at entire state?
  - Faster to “do” than to plan

- Must **limit** search
  - Reduce actions examined
  - Simplify game state
## Internal State Representation

### Simplified World Model
- Includes primary resources
  - **Example**: ammo, health
- Rough notion of position
  - **Example**: in/outside room
  - Both characters and items
- Game mechanic details
  - **Example**: respawn rate
  - Allows tactical decisions

### Uses of Internal State
- Notice changes
  - Health is dropping
  - Enemy must be nearby
- Remember recent events
  - Enemy has left the room
  - Chase after fleeing enemy
- Remember older events
  - Picked up health 30 sec ago
## Internal State Representation

### Simplified World Model

- Includes primary resources
  - **Example**: ammo, health

- Rough notion of position
  - **Example**: in/outside

- Both characters and items
  - **Example**: respawn rate
  - Allows tactical decisions

### Uses of Internal State

- Notice changes
  - Health is dropping

- Health nearby

- Remember recent events
  - Enemy has left the room
  - Chase after fleeing enemy

- Remember older events
  - Picked up health 30 sec ago

---

**Similar to Non-Digital Prototype**
Internal State and Memory

- Each NPC has own state
  - Represents NPC memory
  - Might not be consistent
- Useful for character AI
  - Models sensory data
  - Models communication
- Isolates planning
  - Each NPC plans separately
  - Coordinate planning with a strategic manager
Strategy versus Tactics

Slide courtesy of Dave Mark
Internal State for Quake II

- **Self**
  - Current-health
  - Last-health
  - Current-weapon
    - Ammo-left
  - Current-room
    - Last-room
  - Current-armor
    - Last-armor
  - Available-weapons
- **Enemy**
  - Current-weapon
  - Current-room
  - Last-seen-time
  - Estimated-health
  - Current-time
- **Random-number**
- **Powerup**
  - Type
  - Room
  - Available
  - Estimated-spawn-time
- **Map**
  - Rooms
  - Halls
  - Paths
- **Parameters**
  - Full-health
  - Health-powerup-amount
  - Ammo-powerup-amount
  - Respawn-rate
Internal Action Representation

**Simplified Action Model**

- Internal Actions = *operators*
  - Just mathematical functions
  - Operators alter internal state

- **Pre-conditions**
  - What is required for action
  - Often resource requirement

- **Effects**
  - How action changes state
  - Both global and for NPC

**Designing Actions**

- Extrapolate from gameplay
  - Start with an internal state
  - Pick “canonical” game state
  - Apply game action to state
  - Back to internal state

- Remove any uncertainty
  - Deterministic NPC behavior
  - “Average” random results
  - Or pick worse case scenario
Internal Action Representation

Simplified Action Model

- Internal Actions = operators
  - Just mathematical functions
  - Operators alter internal state

- Pre-conditions
  - What is required for action
  - Often resource requirement

- Effects
  - How action changes state
  - Both global and for NPC

Designing Actions

- Extrapolate from gameplay
  - Start with an internal state
  - Pick “canonical” game state
  - Apply game action to state
  - Back to internal state

- Remove any uncertainty
  - Deterministic NPC behavior
  - “Average” random results
  - Or pick worse case scenario

Like Gameplay Specification, but actions, interactions combined

Strategic AI
Example: Pick-Up Health Op

- **Preconditions:**
  - `Self.current-room = Powerup.current-room`
  - `Self.current-health < full-health`
  - `Powerup.type = health`
  - `Powerup.available = yes`

- **Effects:**
  - `Self.last-health = self.current-health`
  - `Self.current-health = current-health + health-powerup-amount`
  - `Powerup.available = no`
  - `Powerup.estimated-spawn-time = current-time + respawn-rate`
Building Internal Models

- Planning is only as accurate as model
  - Bad models $\Rightarrow$ bad plans
  - But complex models $\Rightarrow$ slow planning

- Look at your nondigital prototype!
  - Heavily simplified for playability
  - Resources determine internal state
  - Nondigital verbs are internal actions

- One of many reasons for this exercise
What Should We Do?

- **Pickup?**
  - Self.current-health = 20
  - Self.current-weapon = blaster

- **Shoot?**
  - Enemy.estimated-health = 50

- **Pickup?**
  - Powerup.type = health-pak
  - Powerup.available = yes
  - Powerup.type = Railgun
  - Powerup.available = yes
One Step: Pick-up Railgun

Self.current-health = 10  
Self.current-weapon = railgun  
Enemy.estimated-health = 50  
Powerup.type = health-pak  
Powerup.available = yes  
Powerup.type = Railgun  
Powerup.available = no
One Step: Shoot Enemy

Self.current-health = 10
Self.current-weapon = blaster

Enemy.estimated-health = 40

Powerup.type = health-pak
Powerup.available = yes
Powerup.type = Railgun
Powerup.available = yes

Pickup?  Shoot  Pickup?
One Step: Pick-up Health-Pak

Self.current-health = 90
Self.current-weapon = blaster

Enemy.estimated-health = 50

Powerup.type = health-pak
Powerup.available = no

Powerup.type = Railgun
Powerup.available = yes
State Evaluation Function

- Need to **compare** states
  - Is either state better?
  - How far away is goal?

- Might be **partial order**
  - Some states incomparable
  - If not goal, just continue

- Purpose of planning
  - Find good states
  - Avoid bad states
State Evaluation: Quake II

- **Example 1:** Prefer higher self.current-health
  - Always pick up health powerup
  - **Counter example:**
    - Self.current-health = 99%
    - Enemy.current-health = 1%

- **Example 2:** Prefer lower enemy.current-health
  - Always shoot enemy
  - **Counter example:**
    - Self.current-health = 1%
    - Enemy.current-health = 99%
State Evaluation: Quake II

**Example 3:** Prefer higher self.health – enemy.health
- Shoot enemy if I have health to spare
- Otherwise pick up a health pack
- Counter examples?

**Examples of more complex evaluations**
- If self.health > 50% prefer lower enemy.health
  - Otherwise, want higher self.health
- If self.health > low-health prefer lower enemy.health
  - Otherwise, want higher self.health
Two Step Look-Ahead

- Self.current-health = 80
- Self.current-weapon = blaster
- Enemy.estimated-health = 40
- Powerup.type = health-pak
- Powerup.available = no
- Powerup.type = Railgun
- Powerup.available = yes

Shoot

Pickup

(Scene description and actions based on the current health and weapon status of the player and the enemy, as well as the availability of powerups.)
Three Step Look-Ahead

Self.current-health = 100  
Self.current-weapon = railgun

Enemy.estimated-health = 0

Powerup.type = health-pak
Powerup.available = no

Powerup.type = Railgun
Powerup.available = no
Look-Ahead Search

One-Step Lookahead

\[
\text{op pickBest(state) \{}
\begin{align*}
&\text{foreach op satisfying precond \{} \\
&\quad \text{newstate} = \text{op(state)} \\
&\quad \text{evaluate newstate}
\}
\]
\]
\[ \text{return op with best evaluation} \]

Multistep Tree Search

\[
\text{[op] bestPath(&state,depth) \{}
\begin{align*}
&\text{if depth == 0 \{ return [] \}} \\
&\text{foreach op satisfying precond \{} \\
&\quad \text{newstate} = \text{op(state)} \\
&\quad [\text{nop}]=\text{bestPath(newstate,depth-1)} \\
&\quad \text{evaluate newstate}
\}
\]
\]
\[ \text{pick op+[nop] with best state} \\
\text{modify state to reflect op+[nop]} \\
\text{return op+[nop]} \]

Look-Ahead Search

- Are more steps better?
  - Longer, more elaborate plans
  - More time & space consuming
  - Opponent or environment can mess up plan
  - Simplicity of internal model causes problems

- In this class, limit three or four steps
  - Anything more, and AI is too complicated
  - **Purpose is to be challenging, not to win**
Opponent: New Problems

- Pickup? (Opponent: New Problems)
- Shoot? (Opponent: New Problems)
- Pickup? (Opponent: New Problems)

- Pickup? (Opponent: New Problems)
- Shoot? (Opponent: New Problems)
- Pickup? (Opponent: New Problems)

- Pickup? (Opponent: New Problems)
- Shoot? (Opponent: New Problems)
- Pickup? (Opponent: New Problems)

- Pickup? (Opponent: New Problems)
- Shoot? (Opponent: New Problems)
- Pickup? (Opponent: New Problems)

Self.current-health = 20
Self.current-weapon = blaster

Enemy.estimated-health = 50

Powerup.type = health-pak
Powerup.available = yes

Powerup.type = Railgun
Powerup.available = yes

Strategic AI
Opponent Model

- **Solution 1**: Assume the worst
  - Opponent does what would be worst for you
  - Full game tree search; exponential

- **Solution 2**: What would I do?
  - Opponent does what you would in same situation

- **Solution 3**: Internal opponent model
  - Remember what did last time
  - Or remember what they like to do
Opponent Interference

• Opponent actions may prevent yours
  • **Example**: Opponent grabs railgun first
  • Need to take into account in your plan

• **Solution**: Iteration
  • Plan once with no interference
  • Run again, assuming best plans of the opponent
  • Keep iterating until happy (or run out of time)

• Planning is very *expensive!*
Asynchronous AI

Thread 1
- Update
- Check
- Draw

Thread 2
- Request plan
- Check
- Buffer
- Answer
- Check for request
- Compute answer
- Store in buffer

Game Loop

Planning Engine
Using Asynchronous AI

- Give AI a **time budget**
  - If planning takes too long, abort it
  - Use counter in update loop to track time

- **Beware of stale plans**
  - Actual game state has probably changed
  - When find a plan, make sure it is still good
  - Evaluate (quickly) with new internal state
  - Make sure result is “close” to what thought
Planning: Optimization

- **Backwards Planning**
  - **Idea**: few operators achieve goal conditions
  - **Implementation**:
    - For each operator, reverse the effect
    - Check reversed effect satisfies pre-conditions

- **Possible to use backwards and forwards**
  - Start on each end, and check for meets
  - Does not work well with numerical resources
To Plan or Not to Plan

• Advantages
  • Less predictable behavior
  • Can handle unexpected situations
  • More accurate than rule-based AI

• Disadvantages
  • Less predictable behavior (harder to debug)
  • Planning takes a lot of processor time
  • Planning takes memory
  • Need simple but accurate internal representations
Other Possibilities

- There are many more options available
  - Neural nets
  - Decision trees
  - General machine learning
  - Take **CS 4700** if want to learn more

- Quality is a matter of heated debate
  - Better to spend time on internal state design
  - Most AI is focused on perception modeling
Summary

- Rule-based AI is simplest form of strategic AI
  - Only limited to one-step at a time
  - Can easily make decisions that lose in long term

- More complicated behavior requires **planning**
  - Simplify the game to turn-based format
  - Use classic AI search techniques

- Planning has advantages and disadvantages
  - Remember, the desire is to **challenge**, not to **win**