Chapter 5.3
Artificial Intelligence:
Agents, Architecture, and Techniques

Artificial Intelligence

- Intelligence embodied in a man-made device
- Human level AI still unobtainable

Game Artificial Intelligence: What is considered Game AI?

- Is it any NPC behavior?
  - A single "if" statement?
  - Scripted behavior?
- Pathfinding?
- Animation selection?
- Automatically generated environment?
- Best shot at a definition of game AI?

Possible Game AI Definition

Inclusive view of game AI:

> "Game AI is anything that contributes to the perceived intelligence of an entity, regardless of what's under the hood."

Goals of an AI Game Programmer

Different than academic or defense industry

1. AI must be intelligent, yet purposely flawed
2. AI must have no unintended weaknesses
3. AI must perform within the constraints
4. AI must be configurable by game designers or players
5. AI must not keep the game from shipping

Specialization of Game AI Developer

- No one-size fits all solution to game AI
  - Results in dramatic specialization
- Strategy Games
  - Battlefield analysis
  - Long term planning and strategy
- First-Person Shooter Games
  - One-on-one tactical analysis
  - Intelligent movement at footstep level
- Real-Time Strategy games the most demanding, with as many as three full-time AI game programmers

Game Agents

- May act as an
  - Opponent
  - Ally
  - Neutral character
- Continually loops through the Sense-Think-Act cycle
  - Optional learning or remembering step

Sense-Think-Act Cycle: Sensing

- Agent can have access to perfect information of the game world
  - May be expensive/difficult to tease out useful info
- Game World Information
  - Complete terrain layout
  - Location and state of every game object
  - Location and state of player
- But isn't this cheating???

Sensing: Enforcing Limitations

- Human limitations?
- Limitations such as
  - Not knowing about unexplored areas
  - Not seeing through walls
  - Not knowing location or state of player
- Can only know about things seen, heard, or told about
- Must create a sensing model

Sensing:
Human Vision Model for Agents

- Get a list of all objects or agents; for each:
  1. Is it within the viewing distance of the agent?
     - How far can the agent see?
     - What does the code look like?
  2. Is it within the viewing angle of the agent?
     - What is the agent's viewing angle?
     - What does the code look like?
  3. Is it unobscured by the environment?
     - Most expensive test, so it is purposely last
     - What does the code look like?

Sensing: Vision Model

- Isn't vision more than just detecting the existence of objects?
- What about recognizing interesting terrain features?
  - What would be interesting to an agent?

Sensing: Human Hearing Model

- Humans can hear sounds
  - Can recognize sounds
    - Knows what emits each sound
  - Can sense volume
    - Indicates distance of sound
  - Can sense pitch
    - Sounds muffled through walls have more bass
  - Can sense location
    - Where sound is coming from

Sensing: Modeling Hearing

- How do you model hearing efficiently?
  - Do you model how sounds reflect off every surface?
  - How should an agent know about sounds?

Sensing: Modeling Hearing Efficiently

- Event-based approach
  - When sound is emitted, it alerts interested agents
- Use distance and zones to determine how far sound can travel

Sensing: Communication

- Agents might talk amongst themselves!
  - Guards might alert other guards
  - Agents witness player location and spread the word
- Model sensed knowledge through communication
  - Event-driven when agents within vicinity of each other

Sensing: Reaction Times

- Agents shouldn't see, hear, communicate instantaneously
- Players notice!
- Build in artificial reaction times
  - Vision: ¼ to ½ second
  - Hearing: ¼ to ½ second
  - Communication: > 2 seconds

Sense-Think-Act Cycle: Thinking

- Sensed information gathered
- Must process sensed information
- Two primary methods
  - Process using pre-coded expert knowledge
  - Use search to find an optimal solution

Thinking: Expert Knowledge

- Many different systems
  - Finite-state machines
  - Production systems
  - Decision trees
  - Logical inference
- Encoding expert knowledge is appealing because it’s relatively easy
  - Can ask just the right questions
  - As simple as if-then statements
- Problems with expert knowledge
  - Not very scalable

Thinking: Search

- Employs search algorithm to find an optimal or near-optimal solution
- A* pathfinding common use of search

Thinking:
Machine Learning

- If imparting expert knowledge and search are both not reasonable/possible, then machine learning might work
- Examples:
  - Reinforcement learning
  - Neural networks
  - Decision tree learning
- Not often used by game developers
  - Why?

Thinking: Flip-Flopping Decisions

- Must prevent flip-flopping of decisions
- Reaction times might help keep it from happening every frame
- Must make a decision and stick with it
  - Until situation changes enough
  - Until enough time has passed

Sense-Think-Act Cycle: Acting

- Sensing and thinking steps invisible to player
- Acting is how player witnesses intelligence
- Numerous agent actions, for example:
  - Change locations
  - Pick up object
  - Play animation
  - Play sound effect
  - Converse with player
  - Fire weapon

Acting: Showing Intelligence

- Adeptness and subtlety of actions impact perceived level of intelligence
- Enormous burden on asset generation
- Agent can only express intelligence in terms of vocabulary of actions
- Current games have huge sets of animations/assets
  - Must use scalable solutions to make selections

Extra Step in Cycle: Learning and Remembering

- Optional 4th step
- Not necessary in many games
  - Agents don't live long enough
  - Game design might not desire it

Learning

- Remembering outcomes and generalizing to future situations
- Simplest approach: gather statistics
  - If 80% of time player attacks from left
  - Then expect this likely event
- Adapts to player behavior

Remembering

- Remember hard facts
  - Observed states, objects, or players
- For example
  - Where was the player last seen?
  - What weapon did the player have?
  - Where did I last see a health pack?
- Memories should fade
  - Helps keep memory requirements lower
  - Simulates poor, imprecise, selective human memory

Remembering within the World

- All memory doesn't need to be stored in the agent – can be stored in the world
- For example:
  - Agents get slaughtered in a certain area
  - Area might begin to "smell of death"
  - Agent's path planning will avoid the area
  - Simulates group memory

Making Agents Stupid

- Sometimes very easy to trounce player
  - Make agents faster, stronger, more accurate
- Sometimes necessary to dumb down agents, for example:
  - Make shooting less accurate
  - Make longer reaction times
  - Engage player only one at a time
  - Change locations to make self more vulnerable

Agent Cheating

- Players don’t like agent cheating
  - When agent given unfair advantage in speed, strength, or knowledge
- Sometimes necessary
  - For highest difficultly levels
  - For CPU computation reasons
  - For development time reasons
- Don’t let the player catch you cheating!
  - Consider letting the player know upfront

Finite-State Machine (FSM)

- Abstract model of computation
- Formally:
  - Set of states
  - A starting state
  - An input vocabulary
  - A transition function that maps inputs and the current state to a next state

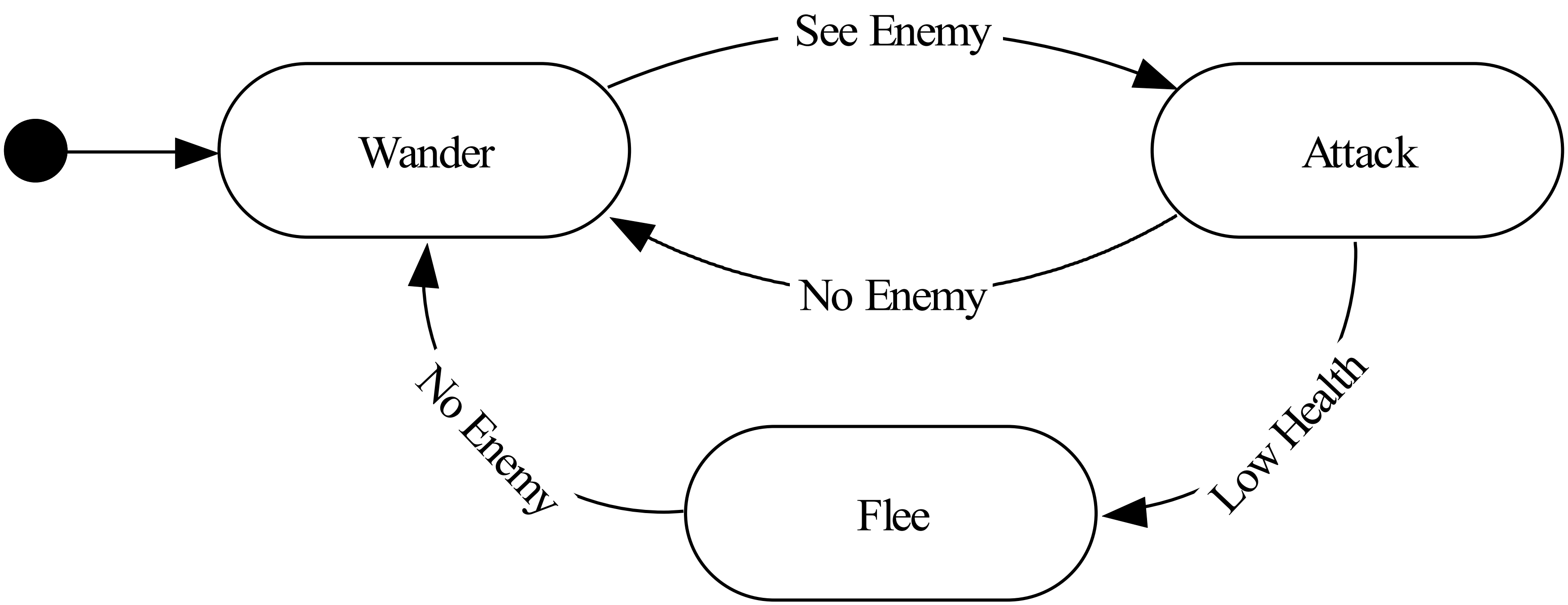
Finite-State Machine: In Game Development

Deviate from formal definition

1. States define behaviors (containing code)
   - Wander, Attack, Flee
2. Transition function divided among states
   - Keeps relation clear
3. Blur between Moore and Mealy machines
   - Moore (within state), Mealy (transitions)
4. Leverage randomness
5. Extra state information
   - For example, health

- Most common game AI software pattern
  - Natural correspondence between states and behaviors
  - Easy to diagram
  - Easy to program
  - Easy to debug
  - Completely general to any problem
- Problems
  - Explosion of states
  - Often created with ad hoc structure

Finite-State Machine: UML Diagram



Finite-State Machine: Approaches

- Three approaches
  - Hardcoded (switch statement)
  - Scripted
  - Hybrid Approach

Finite-State Machine: Hardcoded FSM

```
void RunLogic( int * state ) {
    switch( state )
    {
        case 0:  //Wander
            Wander();
            if( SeeEnemy() )     { *state = 1; }
            break;

        case 1:  //Attack
            Attack();
            if( LowOnHealth() ) { *state = 2; }
            if( NoEnemy() )      { *state = 0; }
            break;

        case 2:  //Flee
            Flee();
            if( NoEnemy() )      { *state = 0; }
            break;
    }
}
```

Finite-State Machine: Problems with switch FSM

1. Code is ad hoc
   - Language doesn't enforce structure
2. Transitions result from polling
   - Inefficient – event-driven sometimes better
3. Can't determine 1st time state is entered
4. Can't be edited or specified by game designers or players

Finite-State Machine: Scripted with alternative language

```
AgentFSM
{
    State( STATE_Wander )
        OnUpdate
            Execute( Wander )
            if( SeeEnemy )    SetState( STATE_Attack )
        OnEvent( AttackedByEnemy )
            SetState( Attack )
    State( STATE_Attack )
        OnEnter
            Execute( PrepareWeapon )
        OnUpdate
            Execute( Attack )
            if( LowOnHealth ) SetState( STATE_Flee )
            if( NoEnemy )     SetState( STATE_Wander )
        OnExit
            Execute( StoreWeapon )
    State( STATE_Flee )
        OnUpdate
            Execute( Flee )
            if( NoEnemy )     SetState( STATE_Wander )
}
```

Finite-State Machine: Scripting Advantages

1. Structure enforced
2. Events can be handed as well as polling
3. OnEnter and OnExit concept exists
4. Can be authored by game designers
   - Easier learning curve than straight C/C++

Finite-State Machine: Scripting Disadvantages

- Not trivial to implement
- Several months of development
  - Custom compiler
    - With good compile-time error feedback
  - Bytecode interpreter
    - With good debugging hooks and support
- Scripting languages often disliked by users
  - Can never approach polish and robustness of commercial compilers/debuggers

Finite-State Machine: Hybrid Approach

- Use a class and C-style macros to approximate a scripting language
- Allows FSM to be written completely in C++ leveraging existing compiler/debugger
- Capture important features/extensions
  - OnEnter, OnExit
  - Timers
  - Handle events
  - Consistent regulated structure
  - Ability to log history
  - Modular, flexible, stack-based
  - Multiple FSMs, Concurrent FSMs
- Can't be edited by designers or players

Finite-State Machine: Extensions

- Many possible extensions to basic FSM
  - OnEnter, OnExit
  - Timers
  - Global state, substates
  - Stack-Based (states or entire FSMs)
  - Multiple concurrent FSMs
  - Messaging

Common Game AI Techniques

- Whirlwind tour of common techniques

Common AI Techniques: A* Pathfinding

- Directed search algorithm used for finding an optimal path through the game world
- A* is regarded as the best
  - Guaranteed to find a path if one exists
  - Will find the optimal path
  - Very efficient and fast

Common AI Techniques: Command Hierarchy

- Strategy for dealing with decisions at different levels
  - From the general down to the foot soldier
- Modeled after military hierarchies
  - General directs high-level strategy
  - Foot soldier concentrates on combat

Common AI Techniques: Dead Reckoning

- Method for predicting object’s future position based on current position, velocity and acceleration
- Works well since movement is generally close to a straight line over short time periods
- Can also give guidance to how far object *could have moved*

Common AI Techniques: Emergent Behavior

- Behavior that wasn't explicitly programmed
- Emerges from the interaction of simpler behaviors or rules

Common AI Techniques: Flocking

- Example of emergent behavior
  - Simulates flocking birds, schooling fish
- Developed by Craig Reynolds
  - 1987 SIGGRAPH paper
- Three classic rules
  1. Separation – avoid local flockmates
  2. Alignment – steer toward average heading
  3. Cohesion – steer toward average position

Common AI Techniques: Formations

- Group movement technique
  - Mimics military formations
- Similar to flocking, but actually distinct
- Each unit guided toward formation position
  - Flocking doesn't dictate goal positions

Common AI Techniques: Influence Mapping

- Method for viewing/abstracting distribution of power within game world
- Typically 2D grid superimposed on land
- Unit influence is summed into each grid cell
  - Unit influences neighboring cells with falloff
- Facilitates decisions
  - Can identify the “front” of the battle
  - Can identify unguarded areas

Common AI Techniques: Level-of-Detail AI

- Optimization technique like graphical LOD
- Only perform AI computations if player will notice
- For example
  - Only compute detailed paths for visible agents
  - Off-screen agents don't think as often

Common AI Techniques: Manager Task Assignment

- Manager organizes cooperation between agents
  - Manager may be invisible in game
  - Avoids complicated negotiation and communication between agents
- Manager identifies important tasks and assigns them to agents

Common AI Techniques: Obstacle Avoidance

- Paths generated from pathfinding algorithm consider only static terrain, not moving obstacles
- Given a path, agent must still avoid moving obstacles
  - Requires trajectory prediction
  - Requires various steering behaviors

Common AI Techniques: Scripting

- Scripting specifies game data or logic outside of the game's source language
- Scripting influence spectrum
  - Level 0: Everything hardcoded
  - Level 1: Data in files specify stats/locations
  - Level 2: Scripted cut-scenes (non-interactive)
  - Level 3: Lightweight logic, like trigger system
  - Level 4: Heavy logic in scripts
  - Level 5: Everything coded in scripts

Common AI Techniques: Scripting Pros and Cons

- Pros
  - Scripts changed without recompiling game
  - Designers empowered
  - Players can tinker with scripts
- Cons
  - More difficult to debug
  - Nonprogrammers required to program
  - Time commitment for tools

Common AI Techniques: State Machine

- Most common game AI software pattern
- Set of states and transitions, with only one state active at a time
- Easy to program, debug, understand

Common AI Techniques: Stack-Based State Machine

- Also referred to as push-down automata
- Remembers past states
- Allows for diversions, later returning to previous behaviors

Common AI Techniques: Subsumption Architecture

- Popularized by the work of Rodney Brooks
- Separates behaviors into concurrently running finite-state machines
- Lower layers
  - Rudimentary behaviors (like obstacle avoidance)
- Higher layers
  - Goal determination and goal seeking
- Lower layers have priority
  - System stays robust

Common AI Techniques: Terrain Analysis

- Analyzes world terrain to identify strategic locations
- Identify
  - Resources
  - Choke points
  - Ambush points
  - Sniper points
  - Cover points

Common AI Techniques: Trigger System

- Highly specialized scripting system
- Uses if/then rules
  - If condition, then response
- Simple for designers/players to understand and create
- More robust than general scripting
- Tool development simpler than general scripting

Promising AI Techniques

- Show potential for future
- Generally not used for games
  - May not be well known
  - May be hard to understand
  - May have limited use
  - May require too much development time
  - May require too many resources

Promising AI Techniques: Bayesian Networks

- Performs humanlike reasoning when faced with uncertainty
- Potential for modeling what an AI should know about the player
  - Alternative to cheating
- RTS Example
  - AI can infer existence or nonexistence of player build units

Promising AI Techniques: Blackboard Architecture

- Complex problem is posted on a shared communication space
  - Agents propose solutions
  - Solutions scored and selected
  - Continues until problem is solved
- Alternatively, use concept to facilitate communication and cooperation

Promising AI Techniques: Decision Tree Learning

- Constructs a decision tree based on observed measurements from game world
- Best known game use: Black & White
  - Creature would learn and form "opinions"
  - Learned what to eat in the world based on feedback from the player and world

Promising AI Techniques: Filtered Randomness

- Filters randomness so that it appears random to players over short term
- Removes undesirable events
  - Like coin coming up heads 8 times in a row
- Statistical randomness is largely preserved without gross peculiarities
- Example:
  - In an FPS, opponents should randomly spawn from different locations (and never spawn from the same location more than 2 times in a row).

Promising AI Techniques: Fuzzy Logic

- Extension of classical logic
- In classical crisp set theory, an object either does or doesn’t belong to a set
- In fuzzy set theory, an object can have continuous varying degrees of membership in fuzzy sets

Promising AI Techniques: Genetic Algorithms

- Technique for search and optimization that uses evolutionary principles
- Good at finding a solution in complex or poorly understood search spaces
- Typically done offline before game ships
- Example:
  - Game may have many settings for the AI, but interaction between settings makes it hard to find an optimal combination

Promising AI Techniques: N-Gram Statistical Prediction

- Technique to predict next value in a sequence
- In the sequence 18181810181, it would predict 8 as being the next value
- Example
  - In street fighting game, player just did Low Kick followed by Low Punch
    - Predict their next move and expect it

Promising AI Techniques: Neural Networks

- Complex non-linear functions that relate one or more inputs to an output
- Must be trained with numerous examples
  - Training is computationally expensive making them unsuited for in-game learning
  - Training can take place before game ships
    - Once fixed, extremely cheap to compute

Promising AI Techniques: Perceptrons

- Single layer neural network
- Simpler and easier to work with than multi-layer neural network
- Perceptrons get “stimulated” enough to either fire or not fire
  - Simple yes/no output

Promising AI Techniques: Perceptrons (2)

- Game example: Black & White
  - Creature used perceptron for hunger
  - Three inputs: low energy, tasty food, and unhappiness
  - If creature ate and received positive or negative reinforcement, then perceptron weights were modified
    - Results in learning

Promising AI Techniques: Planning

- Planning is a search to find a series of actions that change the current world state into a desired world state
- Increasingly desirable as game worlds become more rich and complex
- Requires
  - Good planning algorithm
  - Good world representation
  - Appropriate set of actions

Promising AI Techniques: Player Modeling

- Build a profile of the player's behavior
  - Continuously refine during gameplay
  - Accumulate statistics and events
- Player model then used to adapt the AI
  - Make the game easier
  - Make the game harder

Promising AI Techniques: Production Systems

- Formal rule-based system
  - Database of rules
  - Database of facts
  - Inference engine to decide which rules trigger – resolves conflicts between rules
- Example
  - *Soar* used experiment with Quake 2 bots
  - Upwards of 800 rules for competent opponent

Promising AI Techniques: Reinforcement Learning

- Machine learning technique
  - Discovers solutions through trial and error
  - Must reward and punish at appropriate times
  - Can solve difficult or complex problems like physical control problems
- Useful when AI’s effects are uncertain or delayed

Promising AI Techniques: Reputation System

- Models player's reputation within the game world
- Agents learn new facts by watching player or from gossip from other agents
- Based on what an agent knows
  - Might be friendly toward player
  - Might be hostile toward player
- Affords new gameplay opportunities
  - "Play nice OR make sure there are no witnesses"

Promising AI Techniques: Smart Terrain

- Put intelligence into inanimate objects
- Agent asks object how to use it
- Agents can use objects for which they weren't originally programmed for
  - Allows for expansion packs or user created objects, like in The Sims
- Enlightened by Affordance Theory
  - Objects by their very design afford a very specific type of interaction

Promising AI Techniques: Speech Recognition

- Players can speak into microphone to control some aspect of gameplay
- Limited recognition means only simple commands possible
- Problems with different accents, different genders, different ages (child vs adult)

Promising AI Techniques: Text-to-Speech

- Turns ordinary text into synthesized speech
- Cheaper than hiring voice actors
- Quality of speech is still a problem
  - Not particularly natural sounding
  - Intonation problems
  - Algorithms not good at "voice acting"
- Large disc capacities make recording human voices not that big a problem
  - No need to resort to worse sounding solution

Promising AI Techniques: Weakness Modification Learning

- General strategy to keep the AI from losing to the player in the same way every time
- Two main steps
  1. Record a key gameplay state that precedes a failure
  2. Recognize that state in the future and change something about the AI behavior
     - AI might not win more often or act more intelligently, but won't lose in the same way every time
     - Keeps "history from repeating itself"