CS 378: Computer Game Technology

Introduction to Game AI
Spring 2012

Today

- AI
  - Overview
  - State Machines

What is AI?

- AI is the control of every non-human entity in a game
  - The other cars in a car game
  - The opponents and monsters in a shooter
  - Your units, your enemy's units and your enemy in a RTS game
- But, typically does not refer to passive things that just react to the player and never initiate action
  - That's physics or game logic
  - For example, the blocks in Tetris are not AI, nor is the ball in the game you are doing, nor is a flag blowing in the wind
  - It's a somewhat arbitrary distinction

AI in the Game Loop

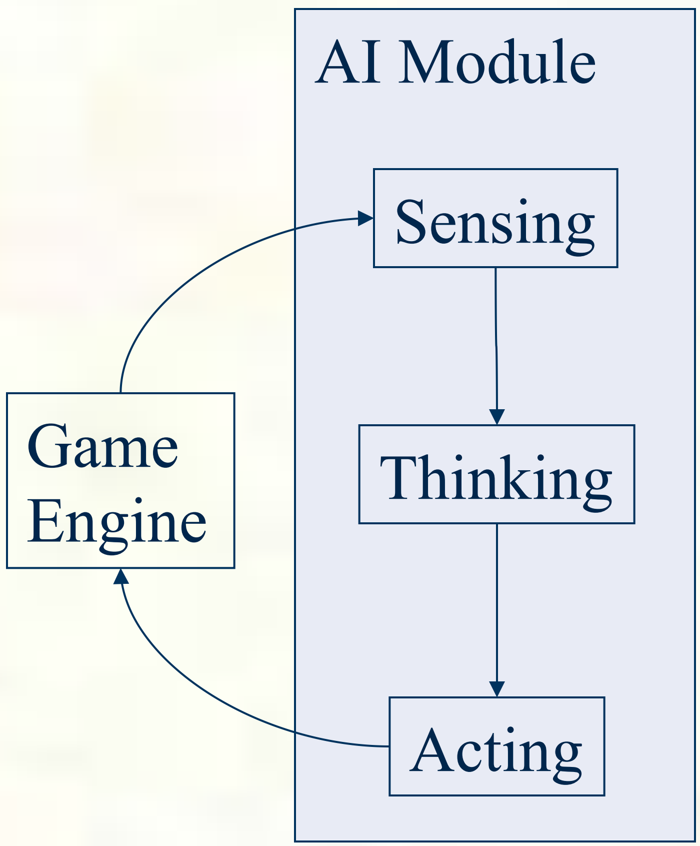
- AI is updated as part of the game loop, after user input, and before rendering
- There are issues here:
  - Which AI goes first?
  - Does the AI run on every frame?
  - Is the AI synchronized?

AI and Animation

- AI determines what to do and the animation does it
  - AI drives animation, deciding what action the animation system should be animating
  - Scenario 1: The AI issues orders like “move from A to B”, and it’s up to the animation system to do the rest
  - Scenario 2: The AI controls everything down to the animation clip to play
- Which scenario is best depends on the nature of the AI system and the nature of the animation system
  - Is the animation system based on move trees (motion capture), or physics, or something else
  - Does the AI look after collision avoidance? Does it do detailed planning?

AI Update Step

- The sensing phase determines the state of the world
  - May be very simple - state changes all come by message
  - Or complex - figure out what is visible, where your team is, etc
- The thinking phase decides what to do given the world
  - The core of AI
- The acting phase tells the animation what to do
  - Generally not interesting



AI by Polling

- The AI gets called at a fixed rate
- Senses: It looks to see what has changed in the world. For instance:
  - Queries what it can see
  - Checks to see if its animation has finished running
- And then acts on it
- Why is this generally inefficient?

Event Driven AI

- Event driven AI does everything in response to events in the world
  - Events sent by message (basically, a function gets called when a message arrives, just like a user interface)
- Example messages:
  - A certain amount of time has passed, so update yourself
  - You have heard a sound
  - Someone has entered your field of view
- Note that messages can completely replace sensing, but typically do not. Why not?
  - Real system are a mix - something changes, so you do some sensing

AI Techniques in Games

- Basic problem: Given the state of the world, what should I do?
- A wide range of solutions in games:
  - Finite state machines, Decision trees, Rule based systems, Neural networks, Fuzzy logic
- A wider range of solutions in the academic world:
  - Complex planning systems, logic programming, genetic algorithms, Bayes-nets
  - Typically, too slow for games

Goals of Game AI

- Several goals:
  - Goal driven - the AI decides what it should do, and then figures out how to do it
  - Reactive - the AI responds immediately to changes in the world
  - Knowledge intensive - the AI knows a lot about the world and how it behaves, and embodies knowledge in its own behavior
  - Characteristic - Embodies a believable, consistent character
  - Fast and easy development
  - Low CPU and memory usage
- These conflict in almost every way

Two Measures of Complexity

- Complexity of Execution
  - How fast does it run as more knowledge is added?
  - How much memory is required as more knowledge is added?
  - Determines the run-time cost of the AI
- Complexity of Specification
  - How hard is it to write the code?
  - As more “knowledge” is added, how much more code needs to be added?
  - Determines the development cost, and risk

Expressiveness

- What behaviors can easily be defined, or defined at all?
- Propositional logic:
  - Statements about specific objects in the world – no variables
  - Jim is in room7, Jim has the rocket launcher, the rocket launcher does splash damage
  - Go to room8 if you are in room7 through door14
- Predicate Logic:
  - Allows general statement – using variables
  - All rooms have doors
  - All splash damage weapons can be used around corners
  - All rocket launchers do splash damage
  - Go to a room connected to the current room

General References

- As recommended by John Laird, academic game AI leader and source of many of these slides
- AI
  - Russell and Norvig: Artificial Intelligence: A Modern Approach, Prentice Hall, 1995
  - Nilsson, Artificial Intelligence: A New Synthesis, Morgan Kaufmann, 1998
- AI and Computer Games
  - LaMothe: Tricks of the Windows Game Programming Gurus, SAMS, 1999, Chapter 12, pp. 713-796
  - www.gameai.com
  - www.gamedev.net

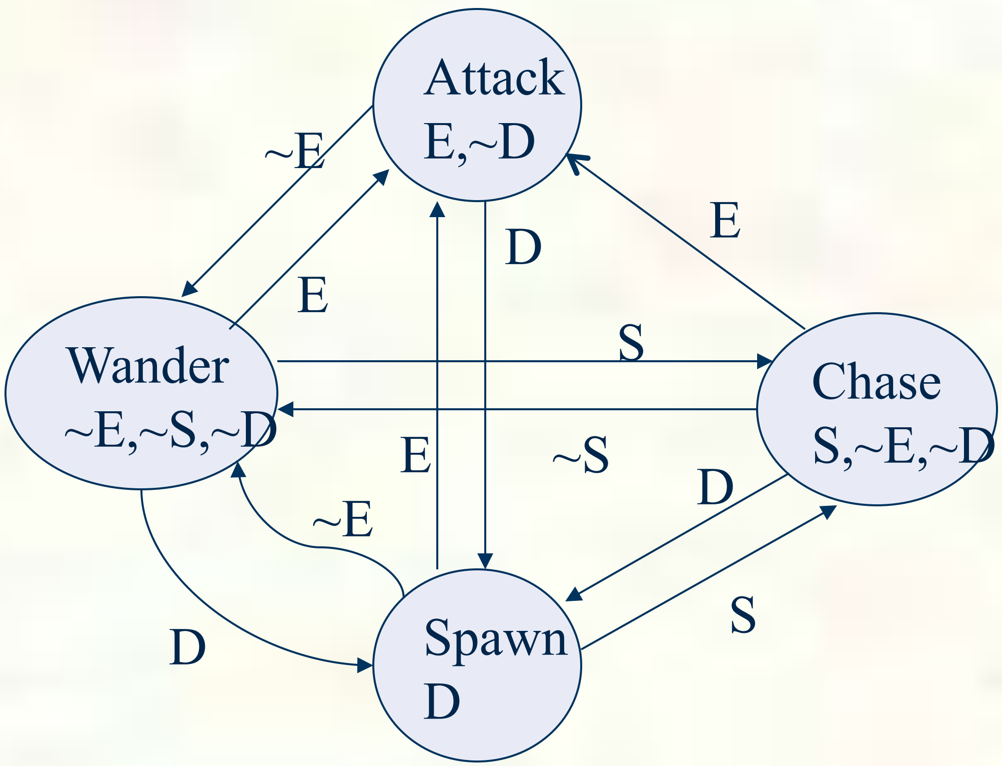
Finite State Machines (FSMs)

- A set of *states* that the agent can be in
- Connected by *transitions* that are triggered by a change in the world
- Normally represented as a directed graph, with the edges labeled with the transition event
- Ubiquitous in computer game AI
- You might have seen them, a long time ago, in formal language theory (or compilers)
  - What type of languages can be represented by finite state machines?
  - How might this impact a character's AI?
  - How does it impact the size of the machine?

Quake Bot Example

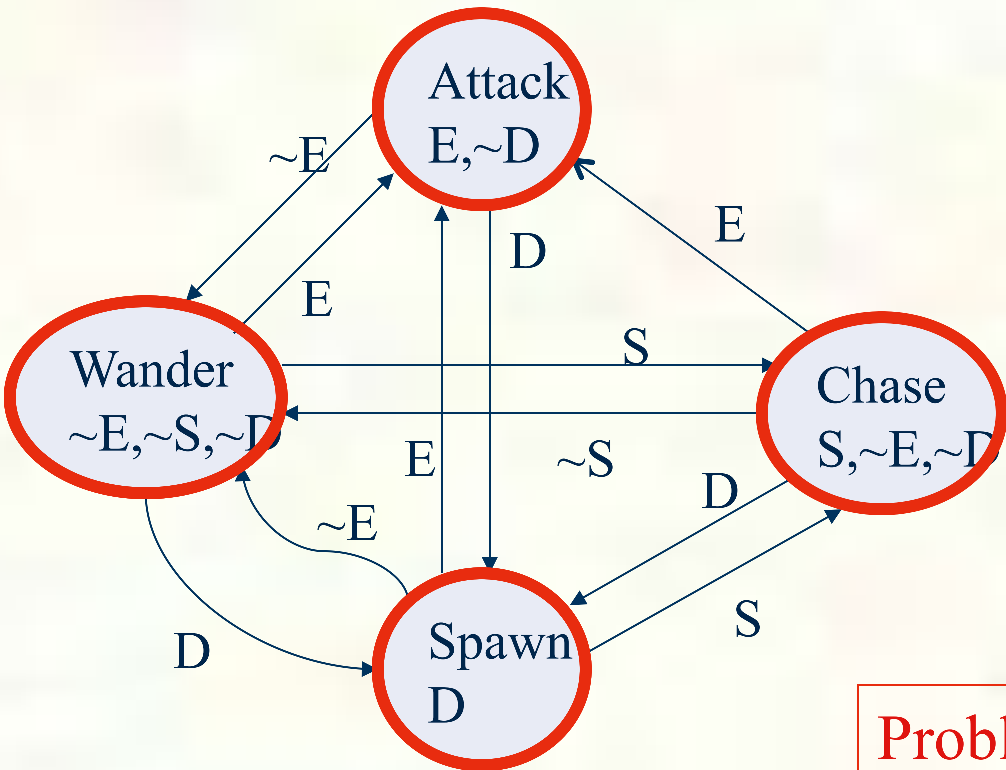
- Types of behavior to capture:
  - Wander randomly if don't see or hear an enemy
  - When see enemy, attack
  - When hear an enemy, chase enemy
  - When die, respawn
  - When health is low and see an enemy, retreat
- Extensions:
  - When see power-ups during wandering, collect them
- Borrowed from John Laird and Mike van Lent's GDC tutorial

Example FSM



- States:
  - E: enemy in sight
  - S: sound audible
  - D: dead
- Events:
  - E: see an enemy
  - S: hear a sound
  - D: die
- Action performed:
  - On each transition
  - On each update in some states (e.g. attack)

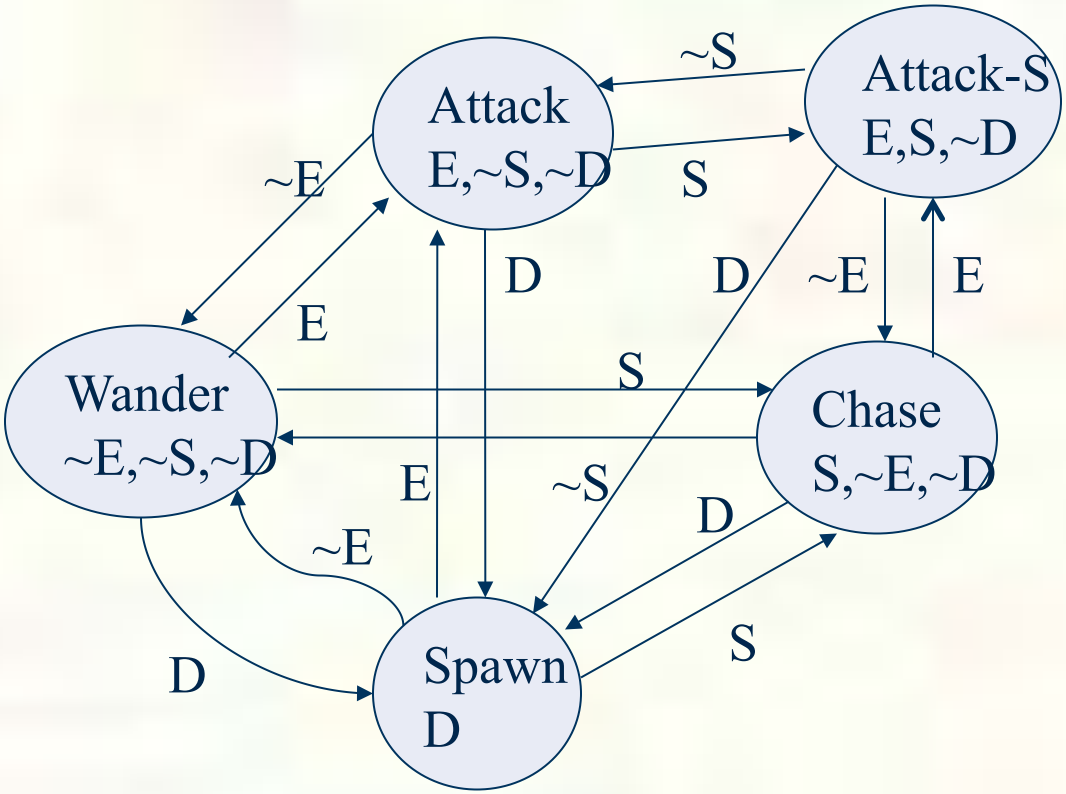
Example FSM Problem



- States:
  - E: enemy in sight
  - S: sound audible
  - D: dead
- Events:
  - E: see an enemy
  - S: hear a sound
  - D: die

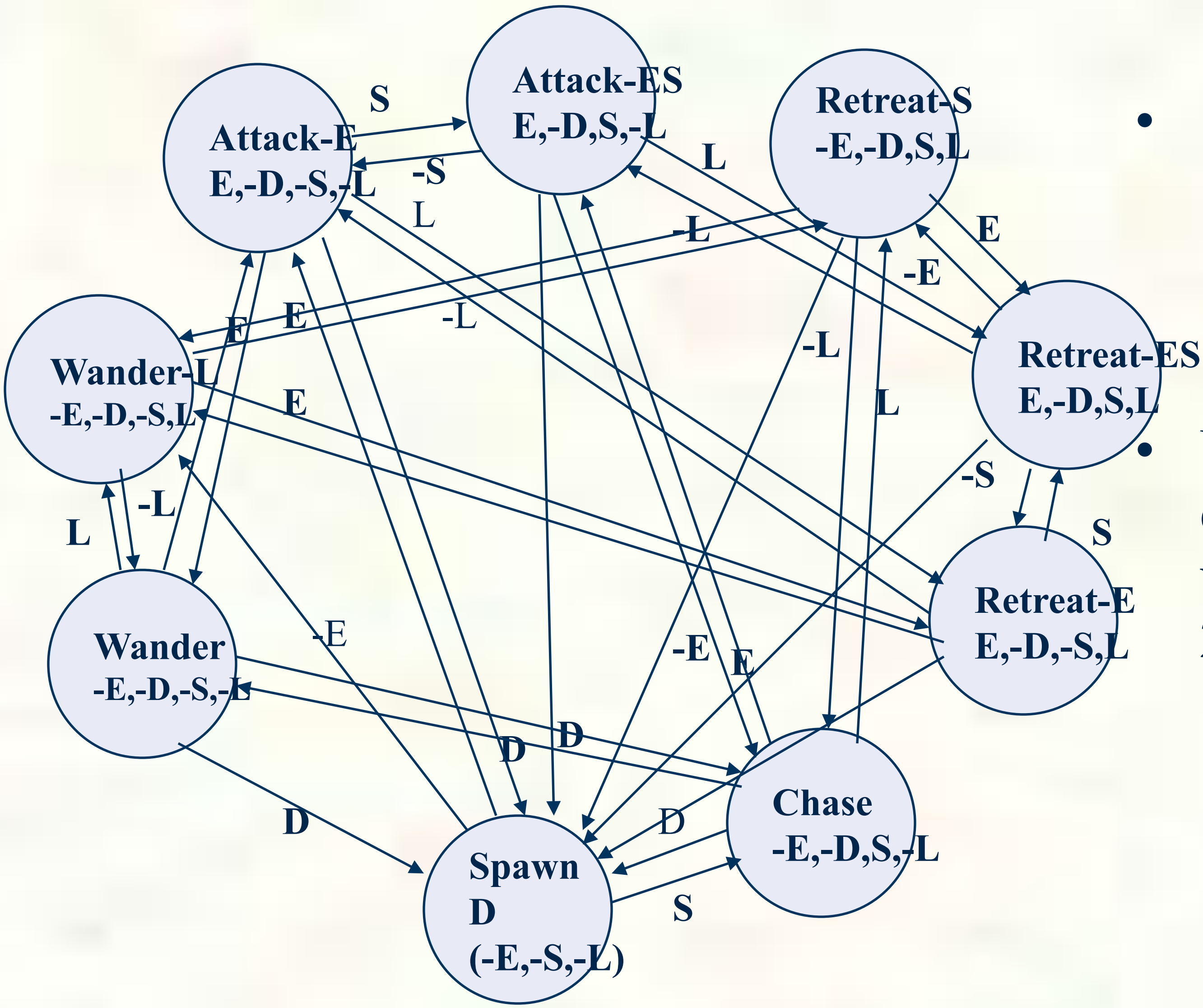
Problem: Can't go directly from attack to chase. Why not?

Better Example FSM



- States:
  - E: enemy in sight
  - S: sound audible
  - D: dead
- Events:
  - E: see an enemy
  - S: hear a sound
  - D: die
- Extra state to recall whether or not heard a sound while attacking

Example FSM with Retreat

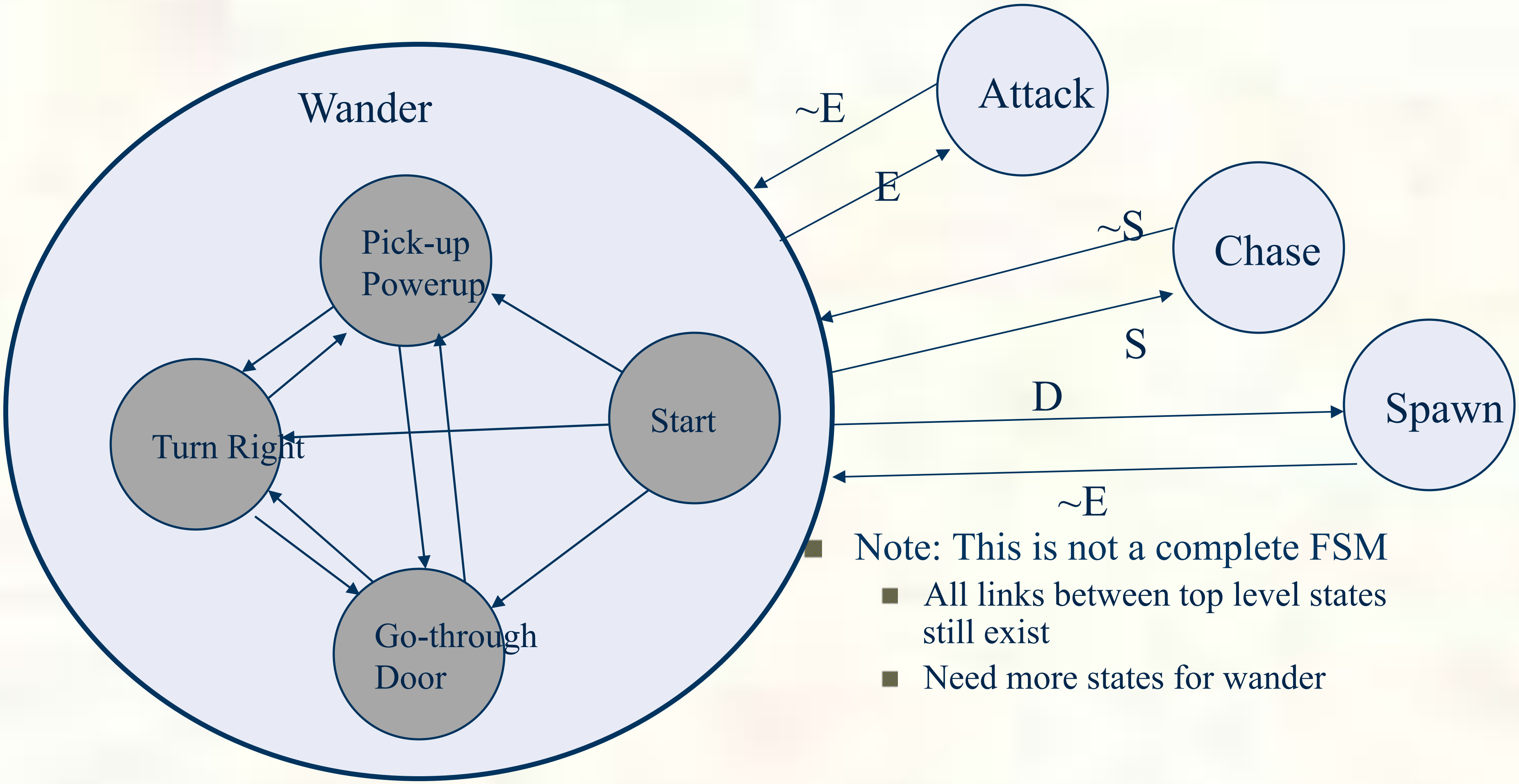


- States:
  - E: enemy in sight
  - S: sound audible
  - D: dead
  - L: Low health
- Worst case: Each extra state variable can add 2*n* extra states
  - *n* = number of existing states

Hierarchical FSMs

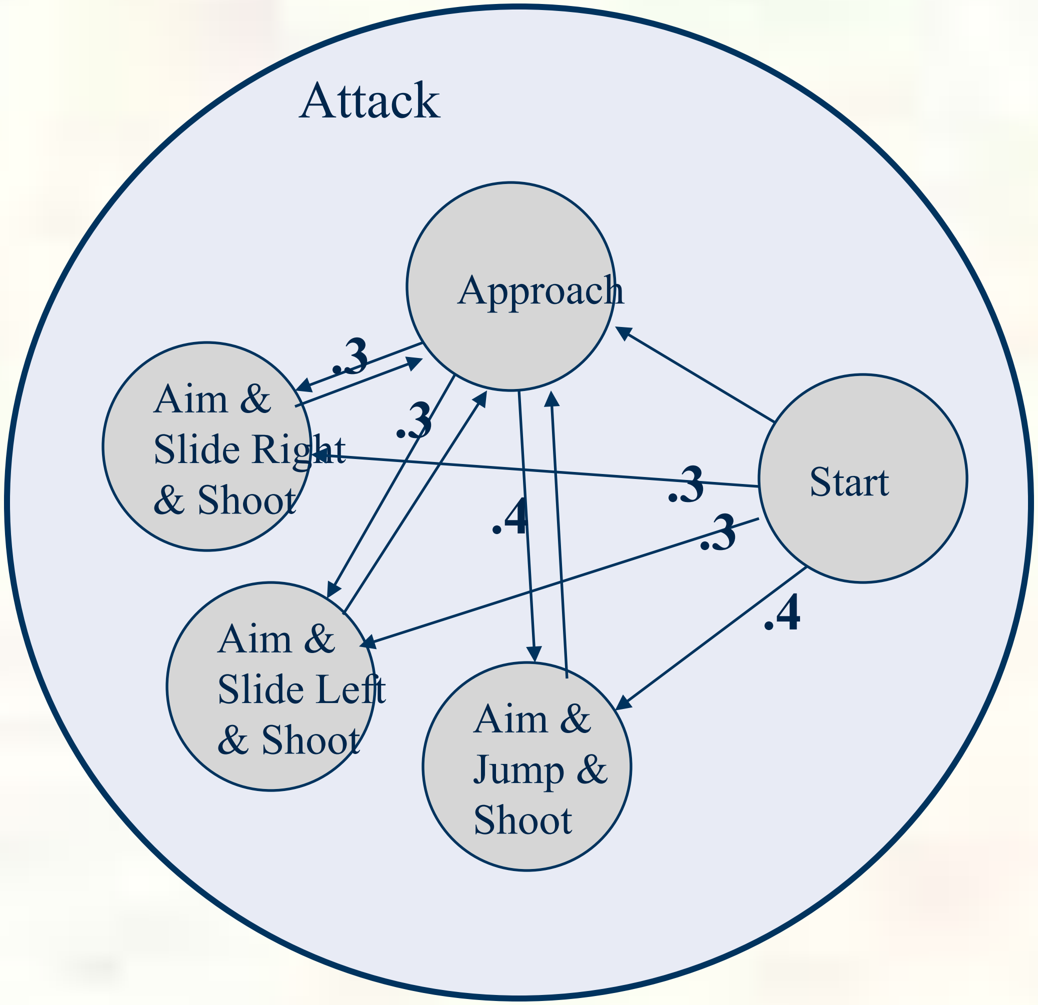
- What if there is no simple action for a state?
- Expand a state into its own FSM, which explains what to do if in that state
- Some events move you around the same level in the hierarchy, some move you up a level
- When entering a state, have to choose a state for it's child in the hierarchy
  - Set a default, and always go to that
  - Or, random choice
  - Depends on the nature of the behavior

Hierarchical FSM Example



- Note: This is not a complete FSM
  - All links between top level states still exist
  - Need more states for wander

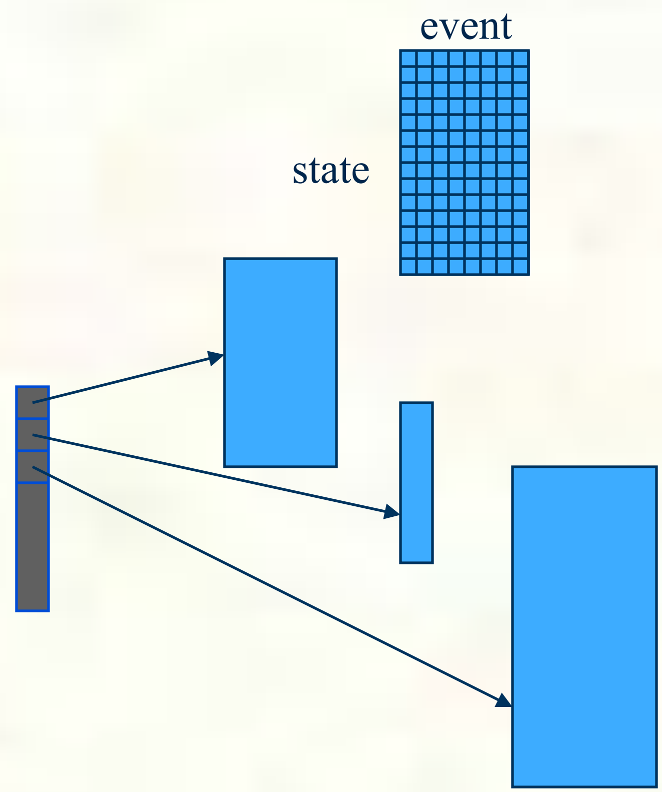
Non-Deterministic Hierarchical FSM (Markov Model)



- Adds variety to actions
- Have multiple transitions for the same event
- Label each with a probability that it will be taken
- Randomly choose a transition at run-time
- Markov Model: New state only depends on the previous state

Efficient Implementation

- Compile into an array of state-name, event
- state-name$_{i+1}$:= array[state-name$_i$, event]
- Switch on state-name to call execution logic
- Hierarchical
  - Create array for every FSM
  - Have stack of states
    - Classify events according to stack
    - Update state which is sensitive to current event
- Markov: Have array of possible transitions for every (state-name,event) pair, and choose one at random



FSM Advantages

- Very fast – one array access
- Expressive enough for simple behaviors or characters that are intended to be "dumb"
- Can be compiled into compact data structure
  - Dynamic memory: current state
  - Static memory: state diagram – array implementation
- Can create tools so non-programmer can build behavior
- Non-deterministic FSM can make behavior unpredictable

FSM Disadvantages

- Number of states can grow very fast
  - Exponentially with number of events: $s=2^e$
- Number of arcs can grow even faster: $a=s^2$
- Propositional representation
  - Difficult to put in “pick up the better powerup”, “attack the closest enemy”
  - Expensive to count: Wait until the third time I see enemy, then attack
    - Need extra events: First time seen, second time seen, and extra states to take care of counting

References

- Web references:
  - www.gamasutra.com/features/19970601/build_brains_into_games.htm
  - csr.uvic.ca/~mmania/machines/intro.htm
  - www.erlang/se/documentation/doc-4.7.3/doc/design_principles/fsm.html
  - www.microconsultants.com/tips/fsm/fsmartcl.htm
- Game Programming Gems Sections 3.0 & 3.1
  - It's very very detailed, but also some cute programming