Introduction to Game AI

CS 395 Game Design

Spring 2002

Expectations on term projects

- Designing and implementing a full game from scratch can be too hard
- Exceptions: Text-based interactive fiction, arcade games, …
- What to do?

Solution One: Leverage

- We're supplying toolkits and resources for some projects
  - Neverworld3
  - Self-explanatory simulator compiler
- There are lots of resources around if you look
  - Game engines
  - Graphics frameworks
  - Games with available source code (e.g., FreeCiv)
- Building on top of such software is *strongly* recommended.

Solution Two: Extension an existing game

- Extensions to existing games can be acceptable (aka “mods”)
  - Must involve significant design work, including analysis of tradeoffs and portfolio of experiments generated in the course of design
  - Must be substantially larger than a typical homework problem

Term Project To Do List

- Identify your teammates early
- Identify your project area early
- Request for Proposals will be made very soon

Overview

- Why AI is important for games
  - And why it needs improving
- Roles for AIs in games
  - The opponent
  - Characters
  - The World
- In-depth look: Tactical decision making in first-person shooters
  - Thanks to John Laird, Mike VanLent, for their GDC 2001 tutorial material

Why AI is important for games

- Essential to the modeled world
  - NPC's of all types: opponents, helpers, extras, …
  - Unrealistic characters ⇒ reduced immersion
  - Stupid, lame behaviors ⇒ reduced fun
  - Superhuman behaviors ⇒ reduced fun
- Until recently, given short shrift by developers
  - Graphics ate almost all the resources
  - Can't start developing the AI until the modeled world was ready to run
    - AI development always late in development cycle
- Situation rapidly changing
  - AI now viewed as helpful in selling the product
  - Still one of the key constraints on game design

Interview with Soren Johnson, www.gamespy.com

GameSpin: What about the [civ3] AI? One of the complaints that players have always had about the AI is that it cheats. Does it still cheat?

Johnson: *The AI has been totally reworked. We started from scratch. We stretched out the difficulty levels. Chieftain is easier than it was in Civ II and Deity is now harder. Does the AI cheat? Yes, but sometimes in favor of the player! Below Prince level it cheats for the player, and above Prince level it cheats against the player. At Prince level there is no cheating.*

Right now, no one at Firaxis can beat the AI at Deity level, though we certainly expect players to find ways to do so within a few months of the game being released.

When we rewrote the AI we threw out the old AI completely. We also looked at some popular tactics like ICS -- infinite city sprawl -- that some players used to beat the AI in Civ II at Deity level. These won't work in Civ III.

AI in game marketing

- “It may be hard to believe that the future of 21st century art is represented by a giant bipedal tiger who farts, break dances and flings livestock around when he's bored.”
  - Review of Black and White in Salon.com, 4/10/2000
- “The most versatile 3D Batman yet, with over 500 animated movements, special fighting moves and a multi-functional cape with its own A.I.”
  - from www.xbox.com

Game flaws engendered by bad AI

- Mindless hordes of opponents
  - “Life is much, much easier thanks to random numbers”
- Rampant cheating by computer opponents
- Focus on violence as principle mode of interaction

Roles for AIs in games

- The opponent
- Characters
- The world

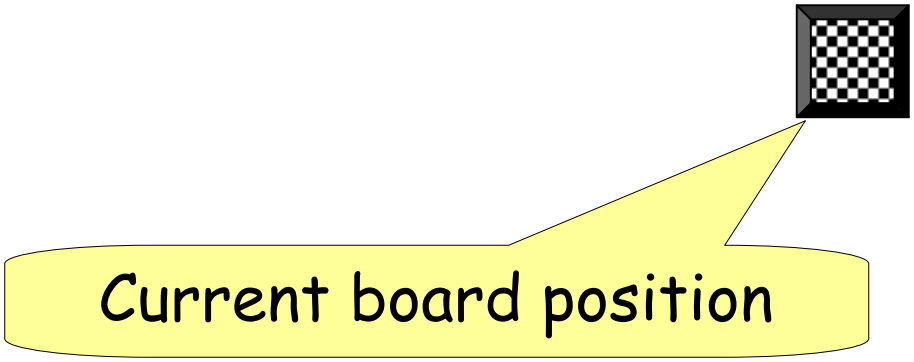
The opponent

- Emphasis: Outsmarting you
- Example: Classic board games
- Questions:
  - What problems do they have to solve?
  - How do they work?

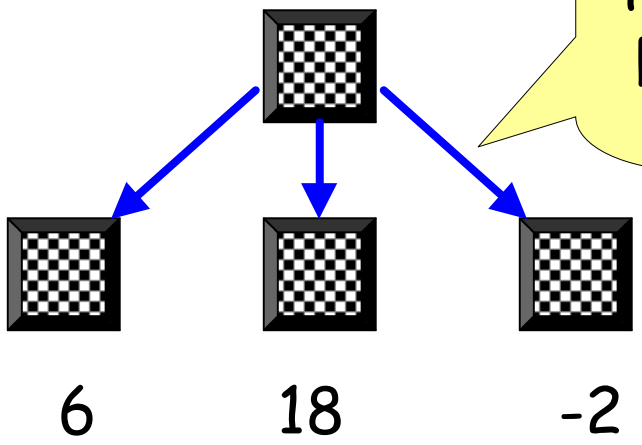
Opponents in classic board games

- *Move generator* figures out what moves are possible
- *Static evaluator* figures out how good each move is, based on changes in board position
- *Search algorithms* look ahead, in a simple form of mental simulation, to figure out the best alternative based counter-moves and counter-moves

Example



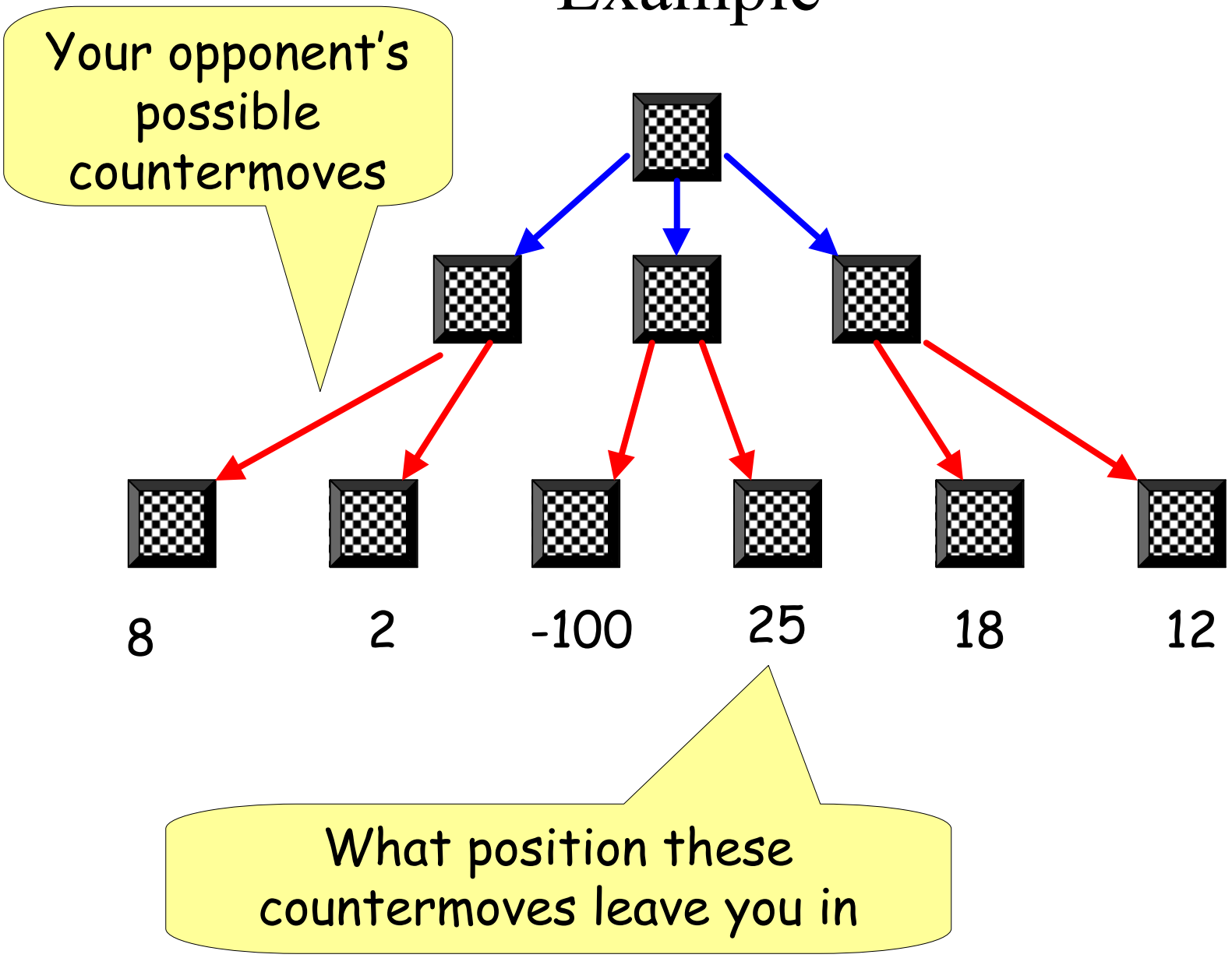
Example



Your possible moves, as produced by move generator

How good each move looks in terms of its immediate results, as computed by the static evaluator

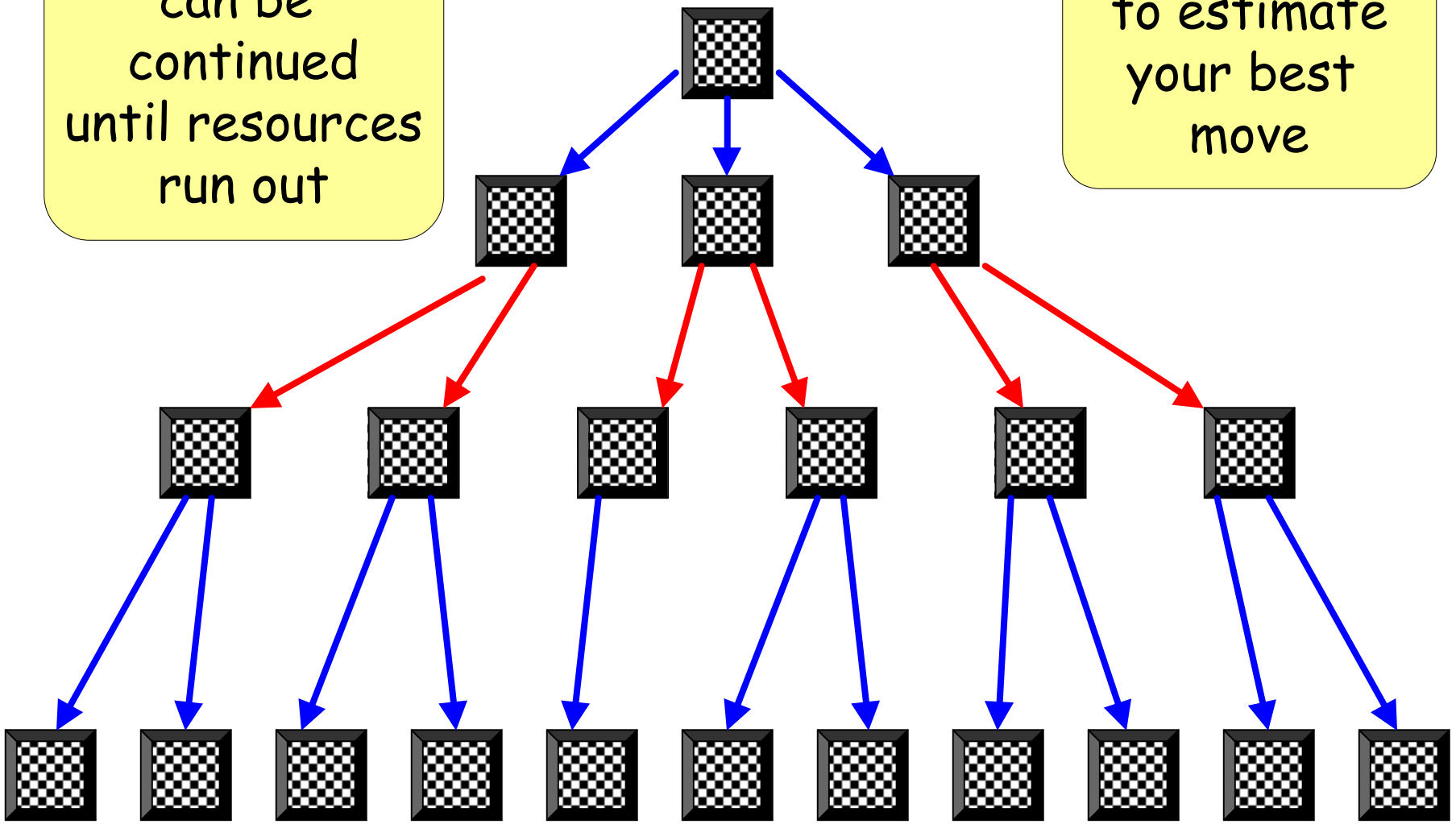
Example


Your opponent's possible countermoves
8
2
-100
25
18
12
What position these countermoves leave you in

Example

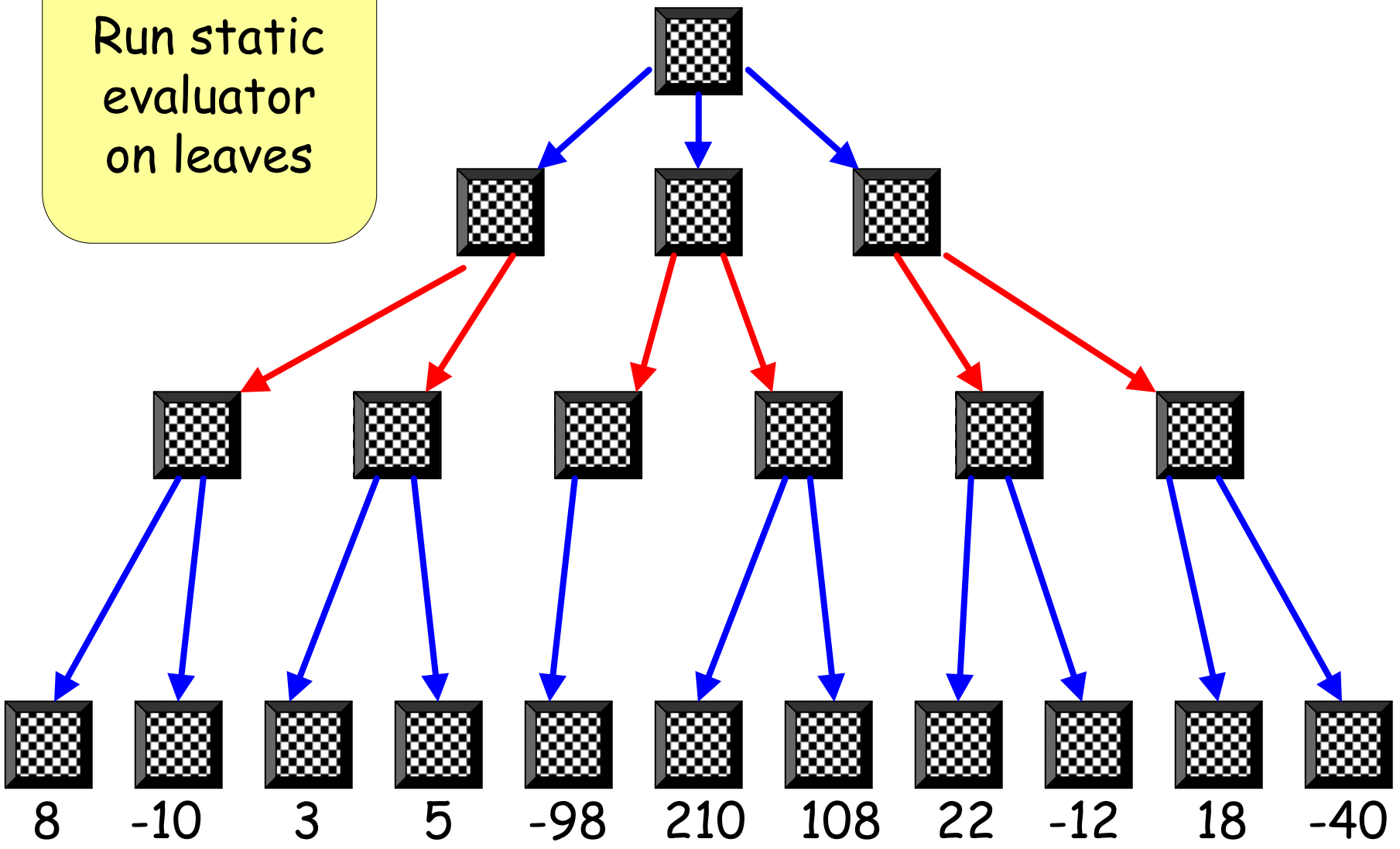
This process can be continued until resources run out

Use minimax to estimate your best move



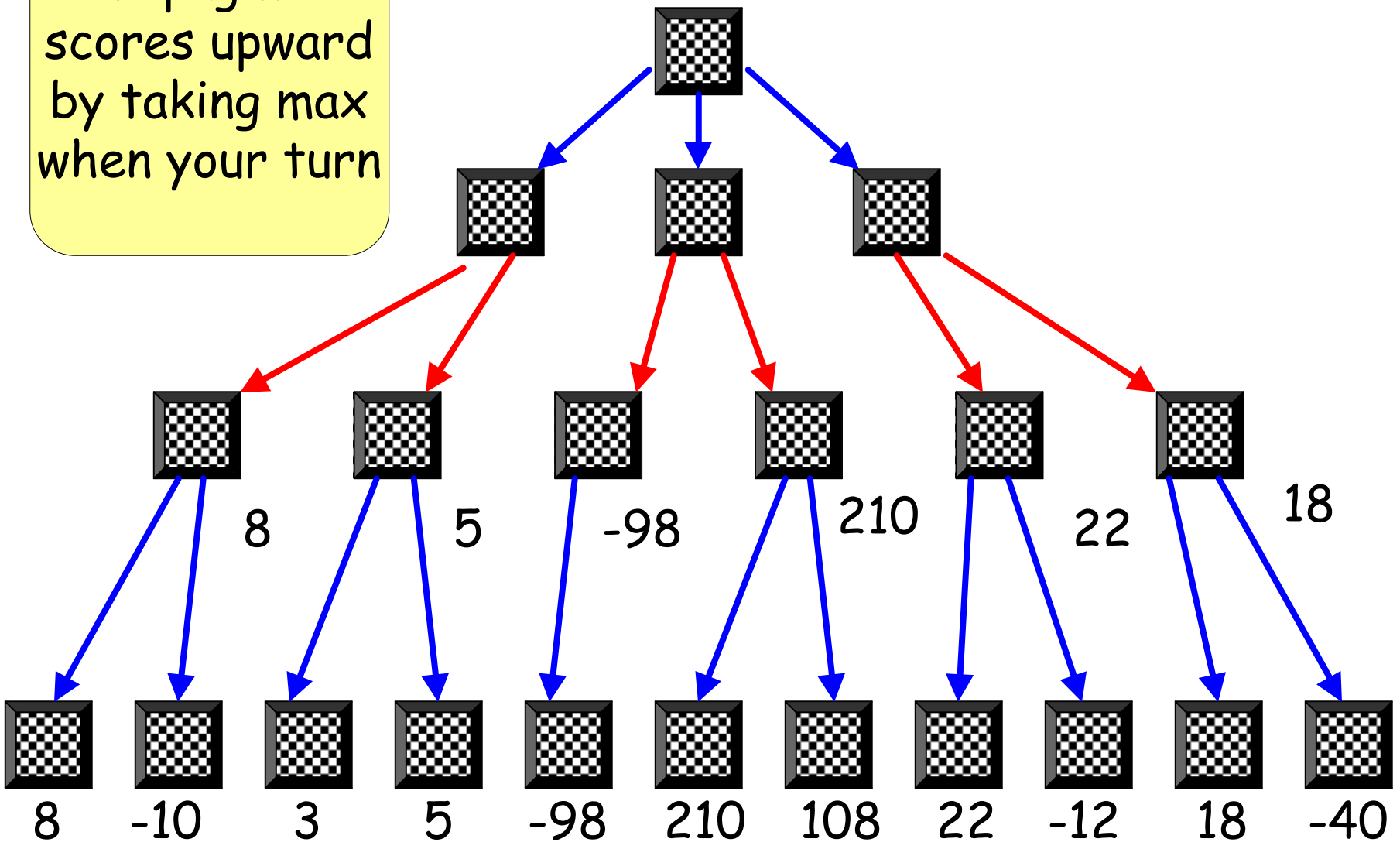
Minimax Example

Run static evaluator on leaves



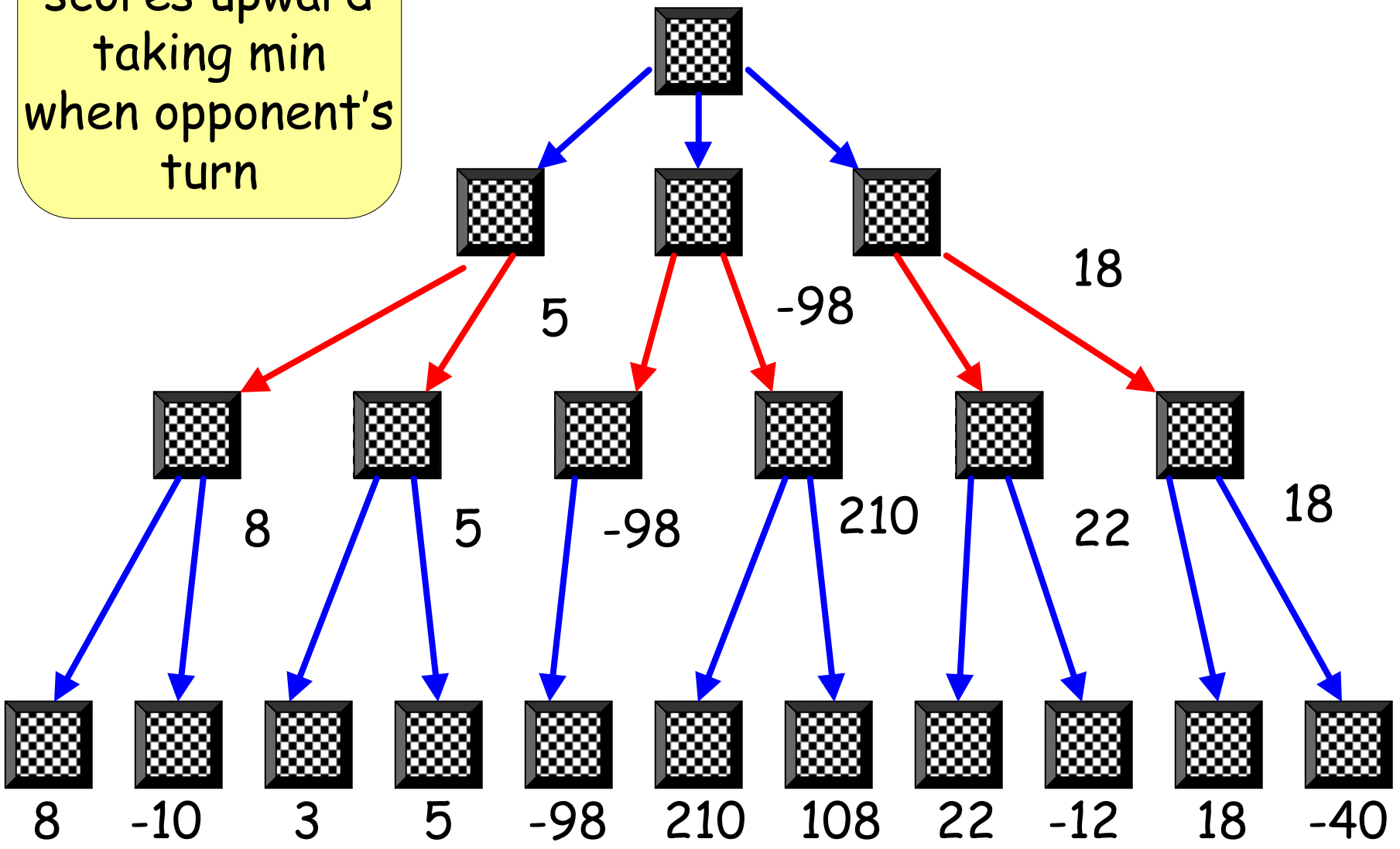
Minimax Example

Propagate scores upward by taking max when your turn



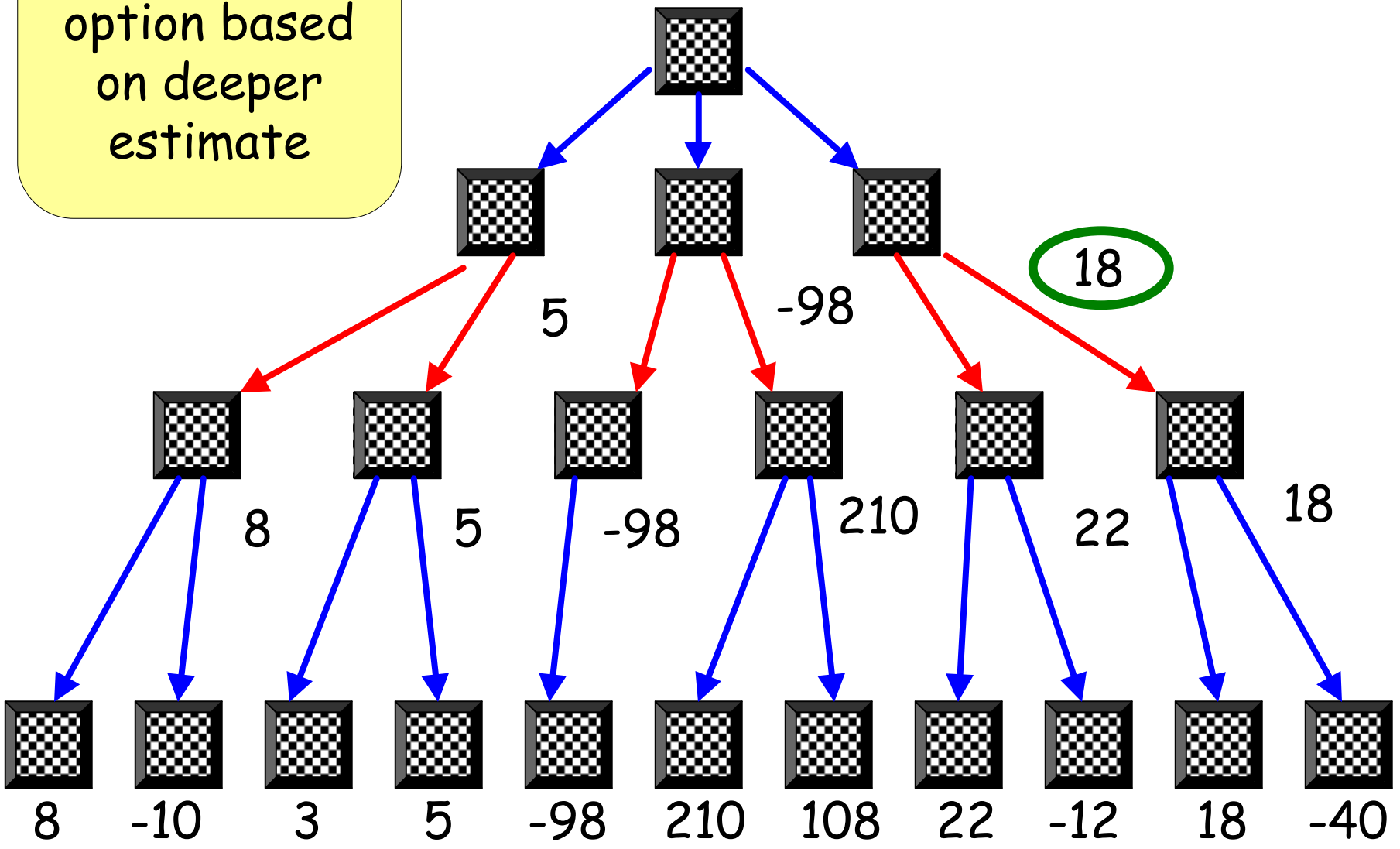
Minimax Example

Propagate scores upward taking min when opponent's turn



Minimax Example

Select best option based on deeper estimate



Search issues

- Lots of techniques for more efficient search
  - Alpha-beta pruning, “book” openings, stability measures, conspiracy numbers
- Basic problem is still exponential
  - Search and brute force only go so far
  - Q: Why did Deep Blue win?
- Knowledge/search tradeoff
  - Simon & Chase experiments: Chess experts use spatial memory for board positions
  - Standard patterns as encoding lessons from experience/deeper search?

Opponents in turn-based strategy games

- Examples: Many strategy war games, Civilization-style games
- What problems do they have to solve?
- How are they similar to, and different from, board games?

Characters

- Examples: First-person shooters, adventure/action games
- What problems do they have to solve?

The World

- Common trick: Share AI computations between computer characters
  - Pathfinding and navigation
  - Awareness of player actions and alignment
  - Strategic thinking
- Examples: More than you might think

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.

Execution Flow of an AI Engine

GAME

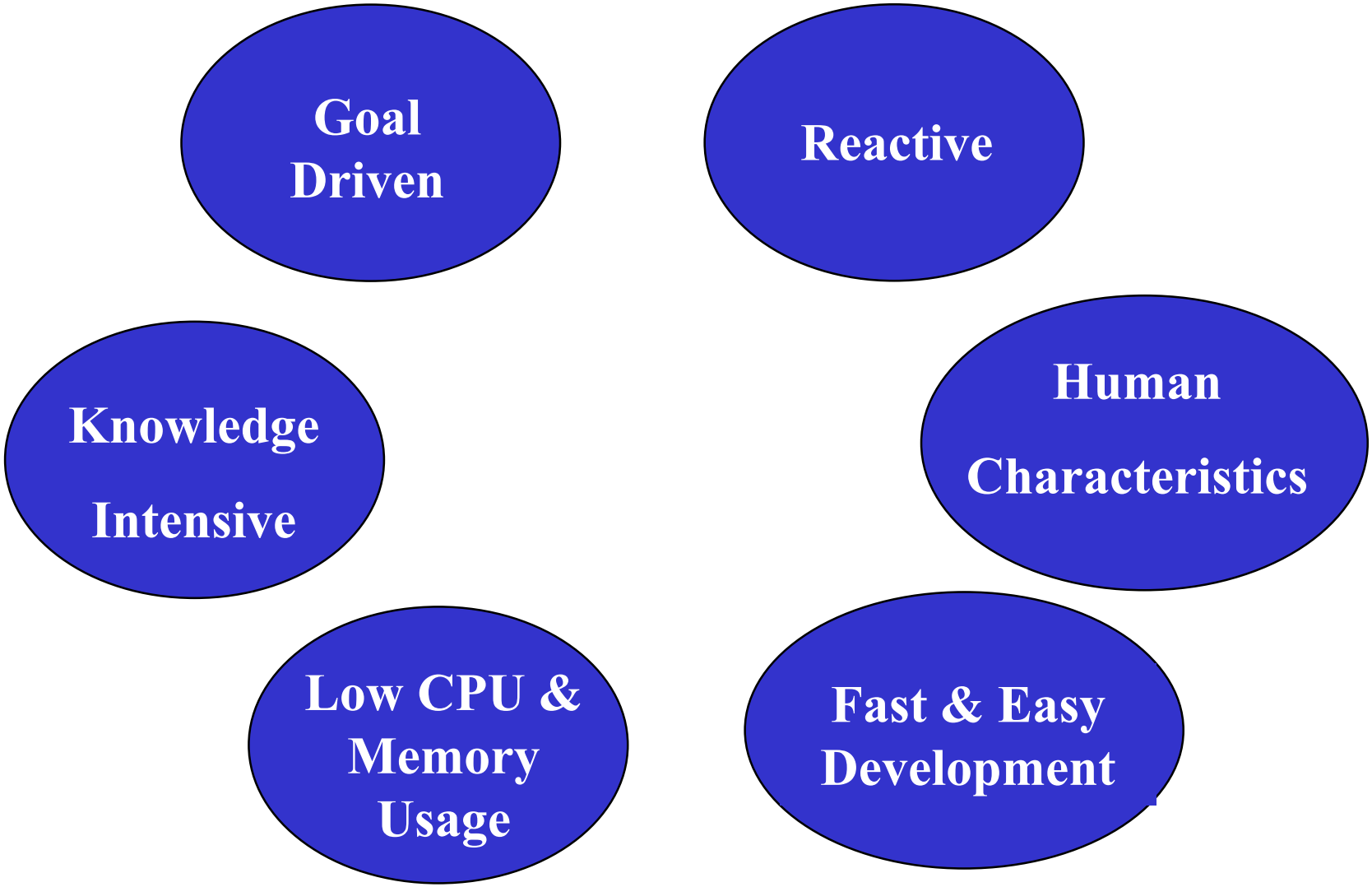
Sense

Think

Act

- Finite-state machines
- Decision trees
- Neural nets
- Fuzzy logic
- Rule-based systems
- Planning systems

Conflicting Goals for AI in Games



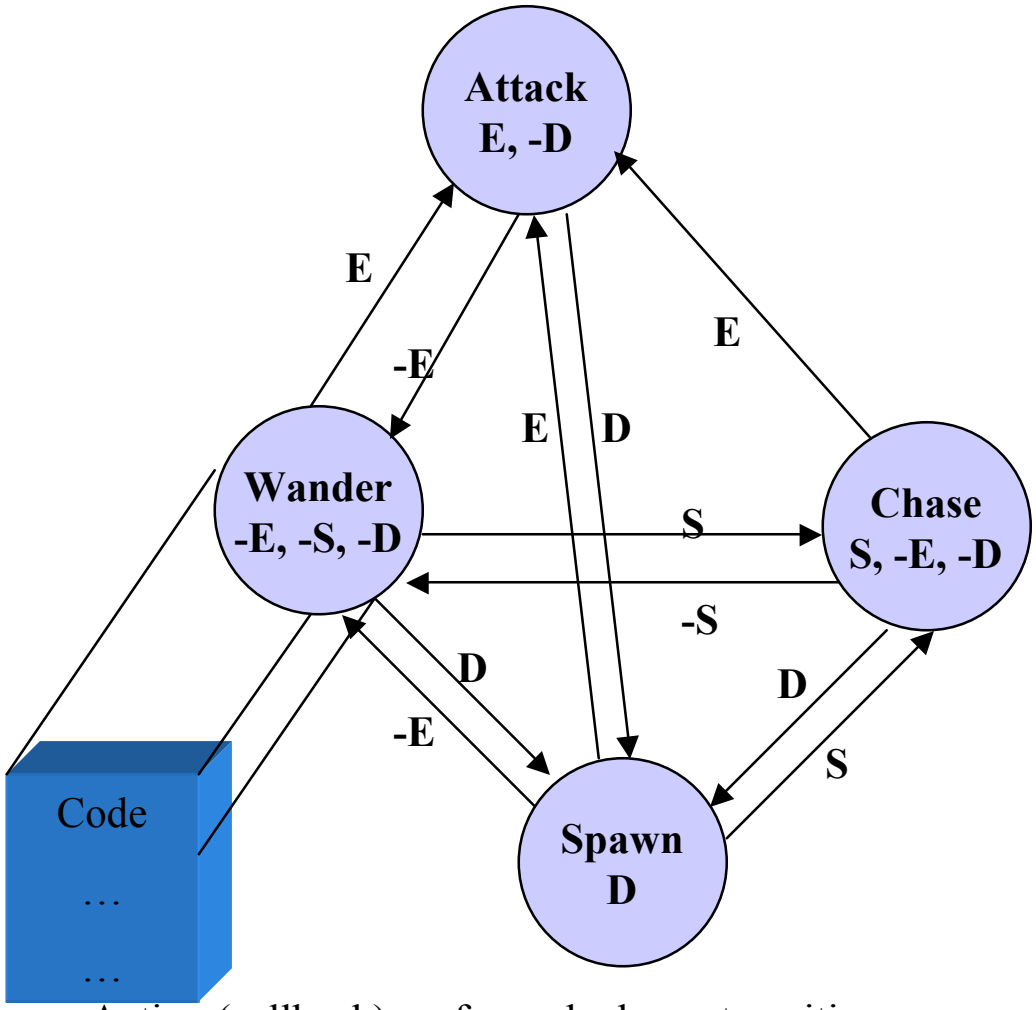
Complexity

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

Expressiveness of Specification

- What can easily be written?
- Propositional:
  - 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.

Example FSM



Events:

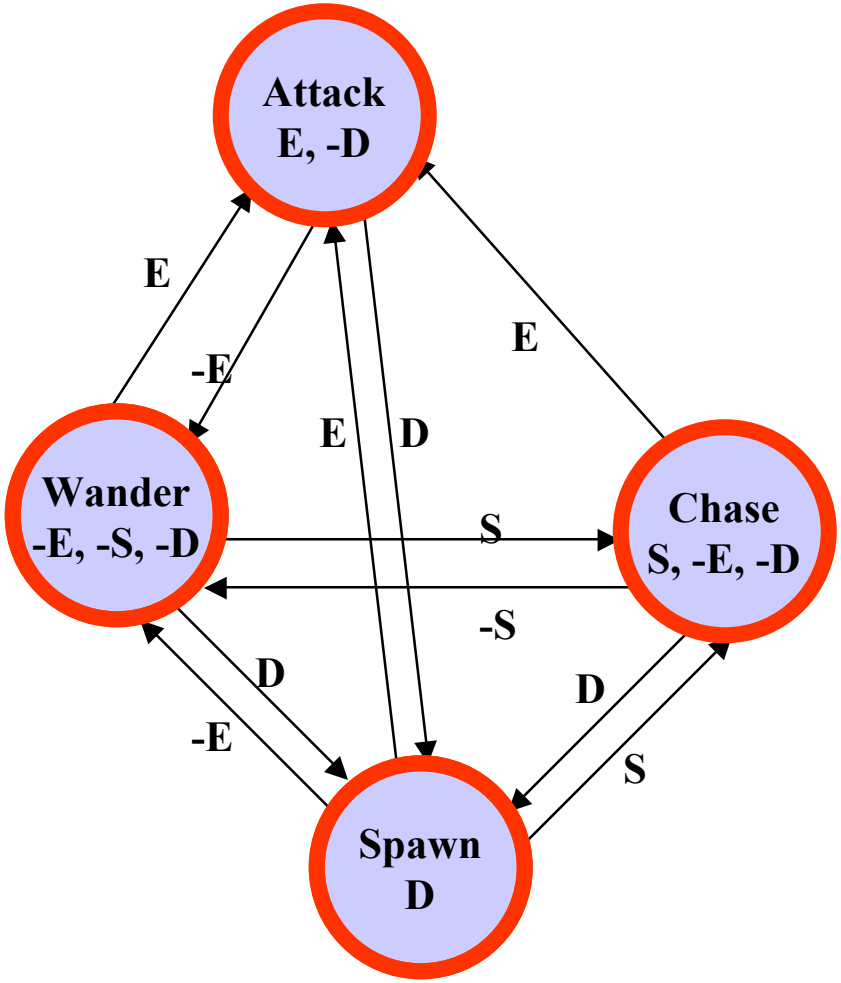
E=Enemy Seen

S=Sound Heard

D=Die

Action (callback) performed when a transition occurs

Example FSM



Events:

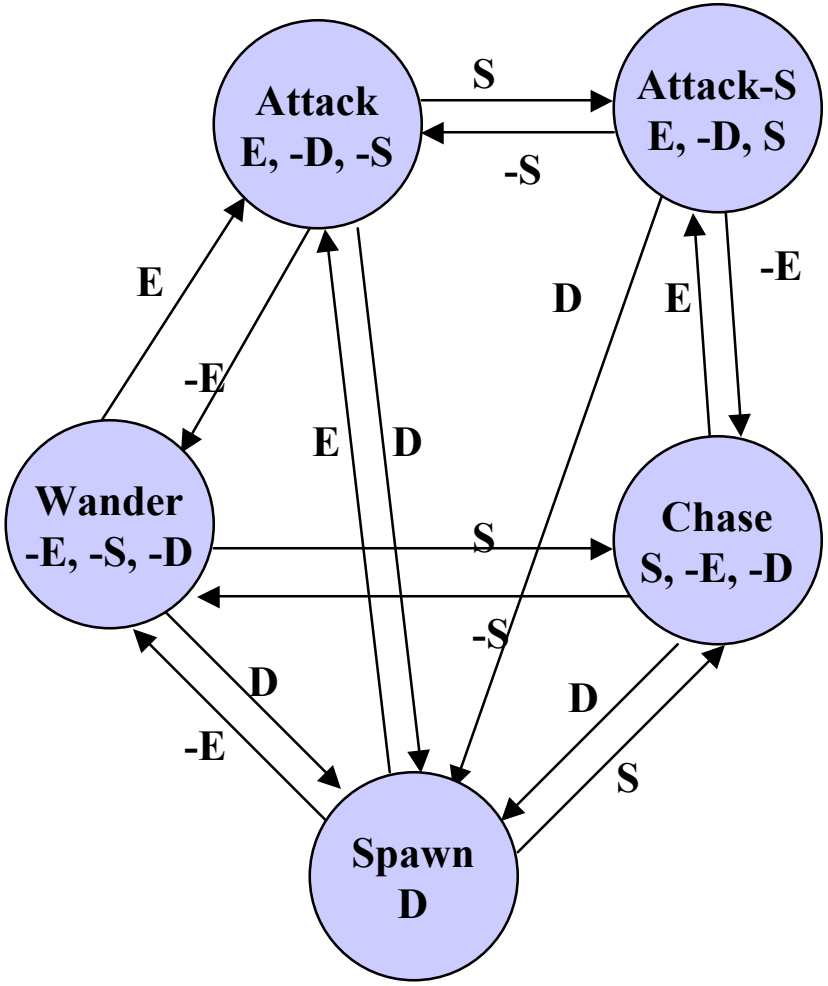
E=Enemy Seen

S=Sound Heard

D=Die

Problem: No transition from attack to chase

Example FSM - Better



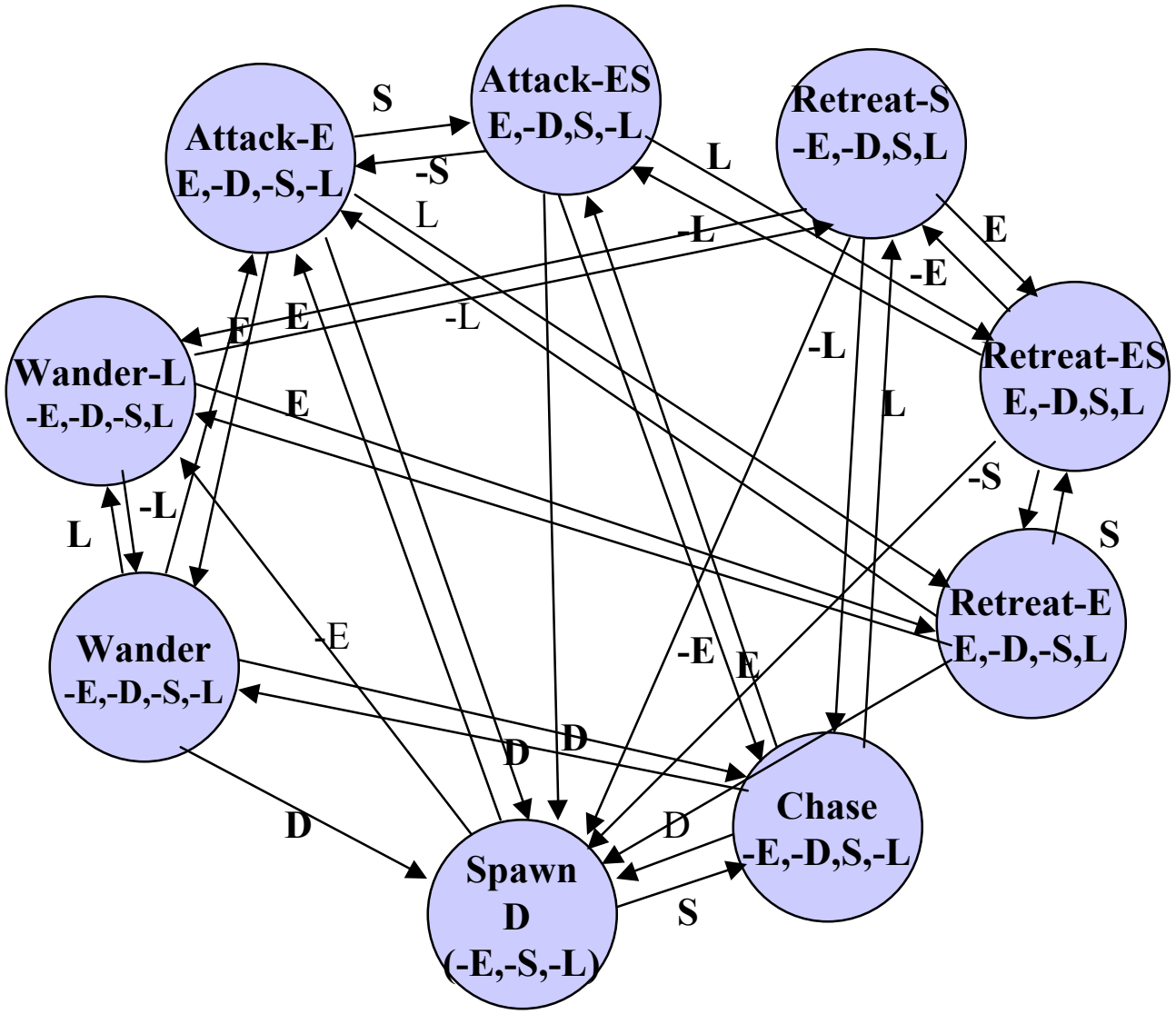
Events:

E=Enemy Seen

S=Sound Heard

D=Die

Example FSM with Retreat



Events:

E=Enemy Seen

S=Sound Heard

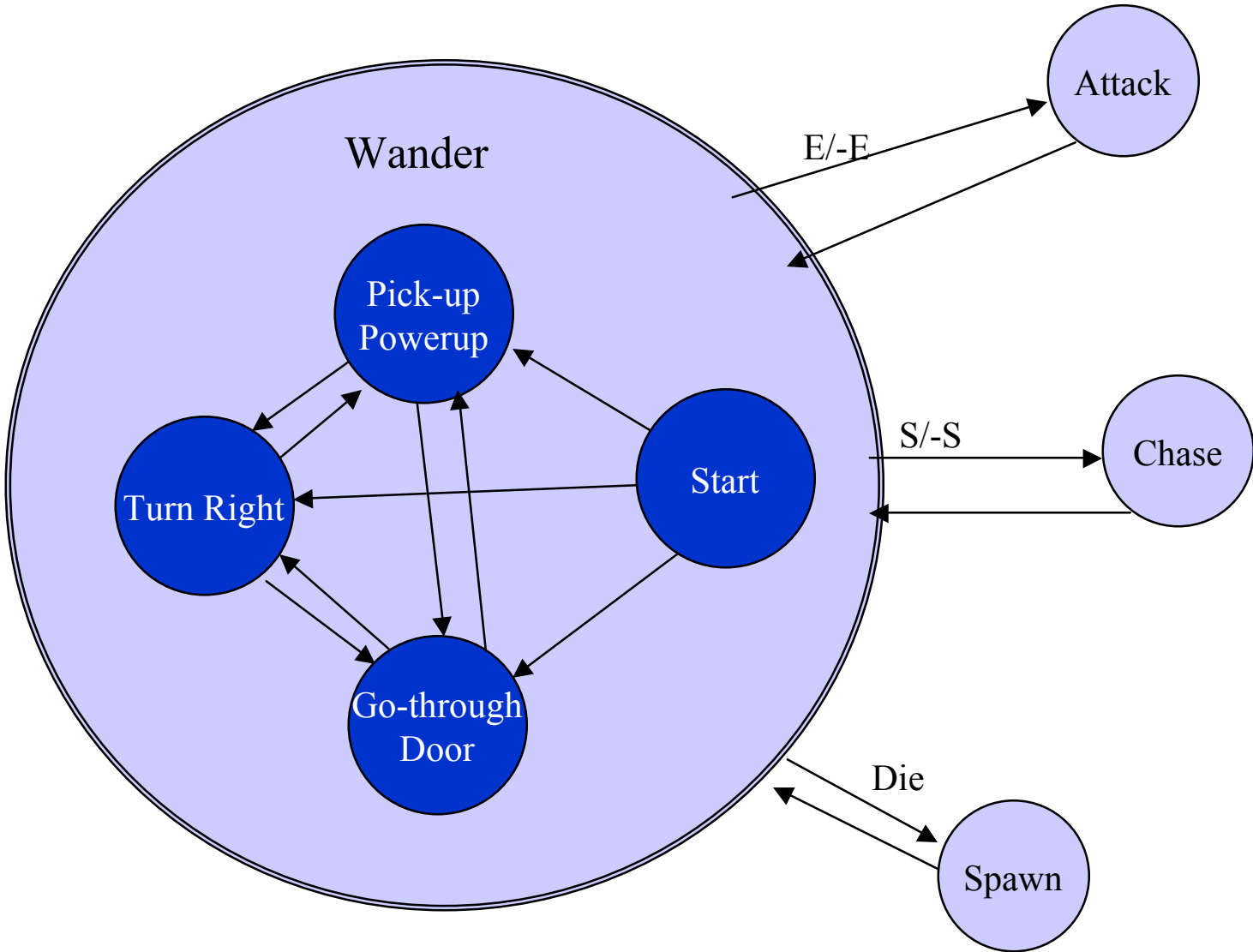
D=Die

L=Low Health

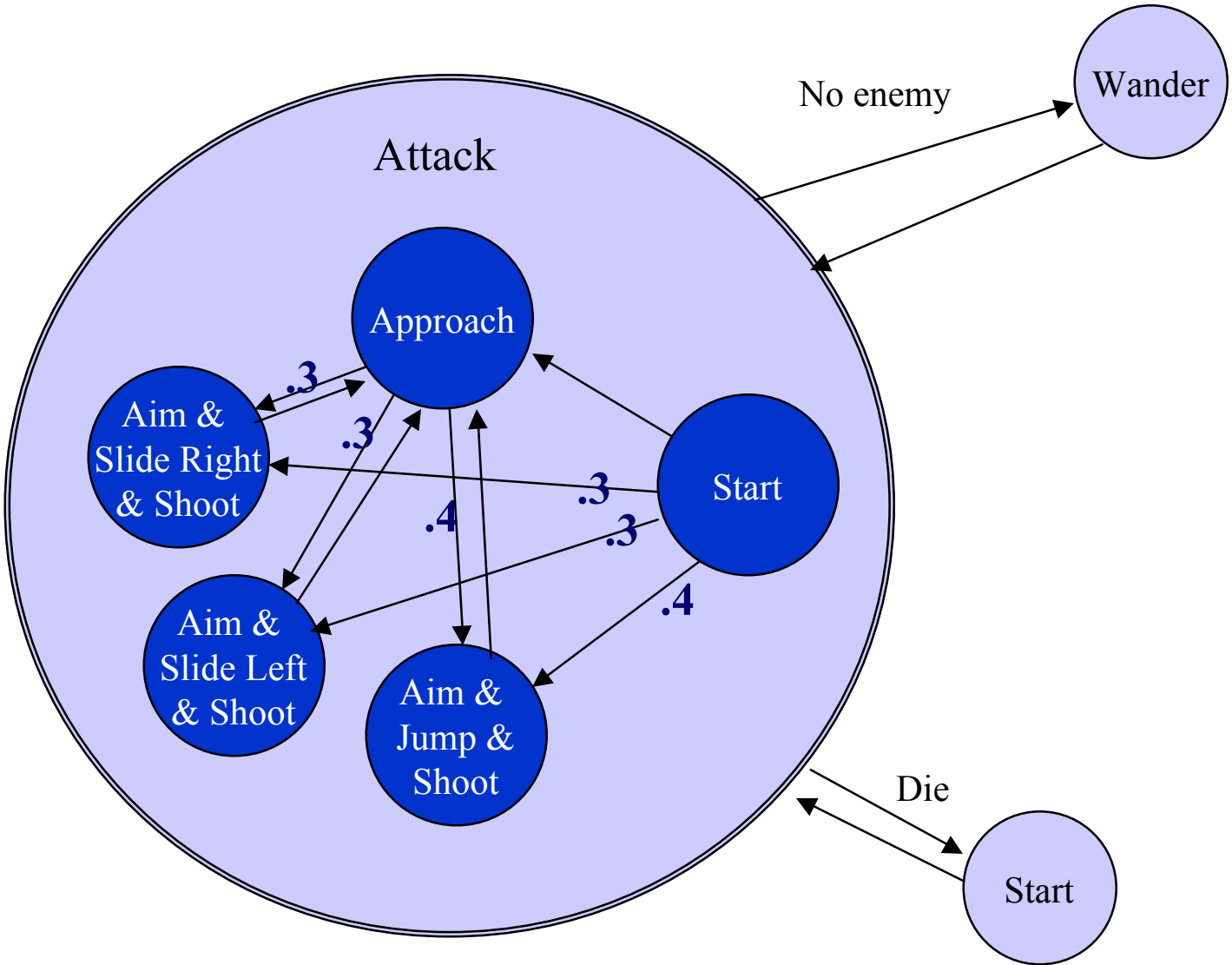
Each feature with N values can require N times as many states

Hierarchical FSM

- Expand a state into its own FSM

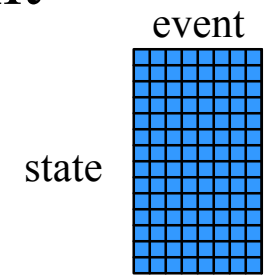


Non-Deterministic Hierarchical FSM (Markov Model)

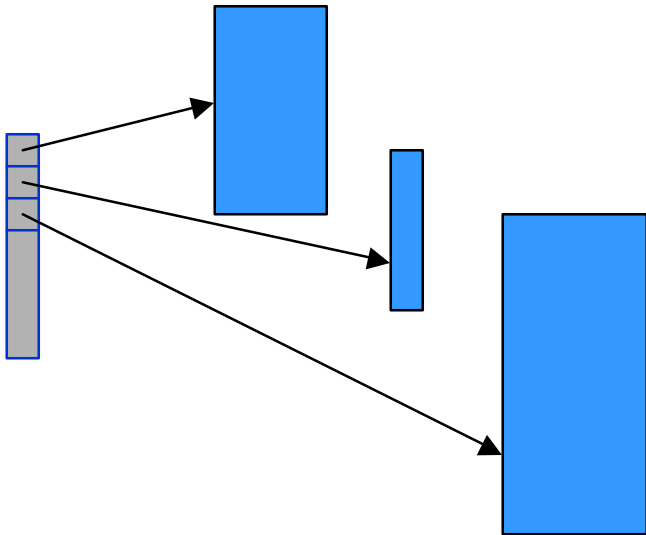


Implementation

- Compile into an array of state-name, event
- state-name := array[state-name, event]
- Uses state-name to call execution logic
- Add buffers to queue up events in case get simultaneous events



- Hierarchical
  - Create array for every FSM
  - Have stack of states
    - Classify events according to stack
    - Update state which is sensitive to current event



FSM Evaluation

- Advantages:
  - Very fast – one array access
  - 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
- 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

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
- Deloura, Game Programming Gems, Charles River Media, 2000, Section 3.0 & 3.1, pp. 221-248.