Game AI in Video- and Strategy- Games (but not e.g. serious Chess)

NPCs (Non Player Characters) *with some AI* are useful for

- challenging opponents; helpful allies; timid victims; mere bystanders
- unseen controllers of opposing armies etc.

- NPCs should be believable, appearing goal-driven, and maybe adaptive
- NPCs, as agents, should appear to *perceive* and *react* to player actions
- They go through a *sense – think – act* cycle (and possibly *learn/adapt* too)

The AI component of a game should be safe - never jeopardise delivery of the game
The AI component of a game is seldom allowed even 20% of computation time

Therefore simple, well understood and reliable techniques are most commonly used

- FSTNs for scripted behaviours (perhaps probabilistic transitions for adaptation)
- Algorithms like A* for path-finding

Desirable features of game AI

A. NPC behaviour should appear intelligent while being deliberately flawed
   - The purpose of NPC intelligence is often *not* to defeat the player, but rather to provide a challenge and ultimately lose in an entertaining way

B. NPCs should not generally appear dumb. (Boss/Drone may be distinguished.)

C. AI should run fast enough to keep within the 20% limit

D. AI should be tunable
   - developers must be able to tweak it for playability
   - sometimes, should be player-customisable – eg for making game extensions

E. AI should not crash the game, should not endanger success of the game: it should be testable to ensure that even millions of purchasers won't cause problems

Perfect and Imperfect AI

A game's AI is not required to be perfect:

- perfection is not even wanted: player should be entertained, not crushed
- imperfect heuristics can be faster than perfect algorithms

While a game's AI has access to perfect information, it should not exploit that

- The game software (physics, animation, …) knows the full truth
- Yet robotics-style perception and motion control should be avoided
  - robot sensors and actuators are not reliable, Game NPC's may be reliable
- *But players should (usually) not feel the AI is cheating*
- Human-like limitations should be placed on what AI can sense and do

Sense-(Learn)-Think-Act

Sensing (without cheating) – seeing, hearing, being told

- **visibility** can involve computationally expensive tests, therefore
  - list those relatively few objects that NPC agent is interested in, and for each
    - is it close enough?
    - if so, is it in view angle? (use limit on dot product)
    - if so, is it (or, more expensively, any part of its bounding box) unobscured?
  - sometimes also interested in non-objects: eg hiding places in FPS games
- **hearing** – loud noises (gunshots), soft sounds (tiptoes)
  - each sound occurs in an area, travels a distance
    - avoid expensive audio simulations, eg of sound reflection (to NPCs)
    - notify NPC agents within sound's travel distance
- **communication** between NPCs (by speech, gesture, writing)

Reaction times are never instantaneous, sense modality matters here

Sense-(Learn)-Think-Act

"Thinking" is traditionally the heart of AI

- If-Then rules ("productions" in Expert Systems)
  - a simple yet powerful technique especially when only a few rules suffice
  - but rule interactions lead to brittleness, resolution rules needed, hard to scale up
- Search
  - Very useful for route finding, also for chaining actions to make plans (*Shakey robot*)
  - Can get very expensive in complex search spaces (even for route finding)
- Learning (online/offline; of various kinds eg decision trees, neural networks)
  - Dangerous, not a generally simple technique, not heavily used in game industry

Decisions should not be frequently revisited – it leads to indecisive flip-flop behavior

Sense-(Learn)-Think-Act

It is only through a NPC's *behaviour* that the player is made to believe that any *sensing* and *thinking* have taken place

The NPC's choice of action should appear to be rational given some reasonable goal that can be attributed to it: attack, hide, run, communicate to player, tell others, etc

Action repertoire (especially in 3D games) may involve thousands of animations: Data-driven design allows removing animation choice from code, promoting scalability.

NPC's thinking may indicate there's no suitable action saving it from doom

- be entertaining: don't dumbly wait to be killed, let player believe NPC comprehends cower in fear; whimper "OH NO!"; cry "DON'T SHOOT!"; appear to try to flee
- In Strategy games, resigning can be better than forcing player to win at length

Sense-(Learn)-Think-Act

Very short lived NPCs have no need to learn as individuals
More long lasting (30 seconds up) may benefit from individual learning

- Remember outcomes of previous actions, use memory to influence future decisions
  - eg player resisted lightning, don't use it as much
  - but let memories fade away, especially unimportant ones
- Spot patterns in player behaviour, use memory to influence future decisions:
  - eg player keeps attacking from left
    - expect him on left for attack/defence; if running away go right

"Smart Terrain" places info & memory in world model, not in individual NPCs.

- If NPCs keep getting killed in a particular area, future short-lifetime NPCs might "smell death" and avoid that area in their route planning and attack plans.

Make NPCs not *too* smart. Don't let them be caught cheating.

The job of a Video Game AI is to lose but in an entertaining way.

- Developers must adjust or tune speed, strength, accuracy (snooker eg), resources (eg ammunition, thinking time, search depth), reaction time, omniscience

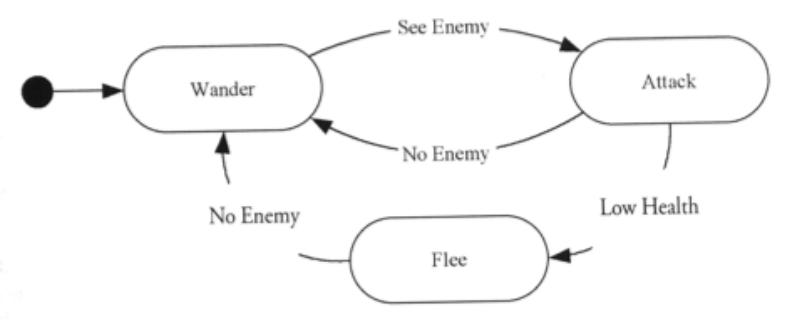
Players tend to resent opponents whose superiority seems to derive from cheating (such as great speed, lightning reactions, high accuracy, omniscience, telepathy)

Yet sometimes it is necessary to cheat like this to present a difficult challenge

- If so, being upfront about it will defuse the resentment and make the game playable
  - Boss Characters may have exceptional abilities
- Concealing it can both destroy the desire to continue playing, and destroy the reputation of the game

Finite State Machines

Example of FSM (Conceptual) driving the behaviour of one NPC



This picture and others to follow are taken from
Introduction to Game Development (2nd ed), by Steve Rabin, 2010

Actions may be performed on transitions (Mealy machine), or when in a state (Moore machine), *or both*

Finite State Machines: direct coding

FSMs can be coded in mainstream programming language, C++ eg

LISTING 5.3.3 A hybrid approach of coding an FSM directly in C++ using a supporting FSM class and C-Style macros.

```
bool AgentFSM::States( StateMachineEvent event,
                       MSG_Object * msg,
                       int state, int substate )
{
    DeclareState( STATE_Wander )
        OnUpdate
            Wander();
            if( SeeEnemy() )
                ChangeState( STATE_Attack );
        OnMsg( MSG_Attacked )
            ChangeState( Attack )

    DeclareState( STATE_Attack )
        OnEnter
            PrepareWeapon();
        OnUpdate
            Attack();
            if( LowOnHealth() )
                ChangeState( STATE_Flee );
            if( NoEnemy() )
                ChangeState( STATE_Wander );
        OnExit
            StoreWeapon();

    DeclareState( STATE_Flee )
        OnUpdate
            Flee();
            if( NoEnemy )
                ChangeState( STATE_Wander );
}
```

Finite State Machines: special-purpose languages, tools

FSMs can be coded in bespoke language, often created especially for a game project

Such a bespoke language may offer event-driven programming facilities

There are middleware providers of compilers and debuggers for such bespoke languages

```
AgentFSM
{DeclareState ( STATE_Wander )
    OnUpdate
       Execute( Wander )
       if( SeeEnemy )
            ChangeState( STATE_Attack )
    OnEvent( AttackedByEnemy )
       ChangeState( Attack )
 DeclareState ( State_Attack )
    OnEnter( PrepareWeapon )
    . . .
    OnExit( StoreWeapon )
 . . .
}
```

This code adapted from Rabin's book

Finite State Machines: Some elaborations

A. An agent may have multiple FSMs concurrently executing, for example
- a "Brain" FSM dealing with decisions about where to go and what to do
- a "movement" FSM dealing with issues like avoiding pits and bystanders

B. FSMs may be arranged in hierarchies, with common sub-behaviours represented by shared specific FSMs, simplifying the FSMs for high-level behaviours)
- ***Subsumption architecture*** is another way of hierarchically arranging multiple FSMs that run concurrently
  - lower level(s) continuously running, doing rudimentary things
    - avoiding obstacles; sensing danger; noticing getting hungry
    - sometimes generating signals/events that are picked up by higher levels
  - higher level(s) are more like conscious decision making

C. A history of FSM states may usefully be maintained, allowing an agent to resume a behaviour that was interrupted (an attack interrupts farming, e.g.)