

AI Tutorial 1: Finite State Machines



New Concepts

- ▶ The Finite State Machine
- ▶ Programming a State Machine
 - ▶ Hard-Coded Switch Statements
 - ▶ Hard-Coded State Pattern
 - ▶ Interpreted State Pattern
- ▶ State Oscillation
- ▶ AI - Behaviours and Types
- ▶ Hierarchical FSM
- ▶ Fuzzy State Machines

The Finite State Machine

What is a Finite State Machine?

- ▶ Wikipedia tells us that a FSM is:

“A behavioural model used to design computer programs. It is composed of a finite number of states associated to transitions. A transition is a set of actions that starts from one state and ends in another (or the same) state. A transition is started by a trigger, and a trigger can be an event or a condition.”

- ▶ To us, it's the building block of all game AI

FSM in Game AI

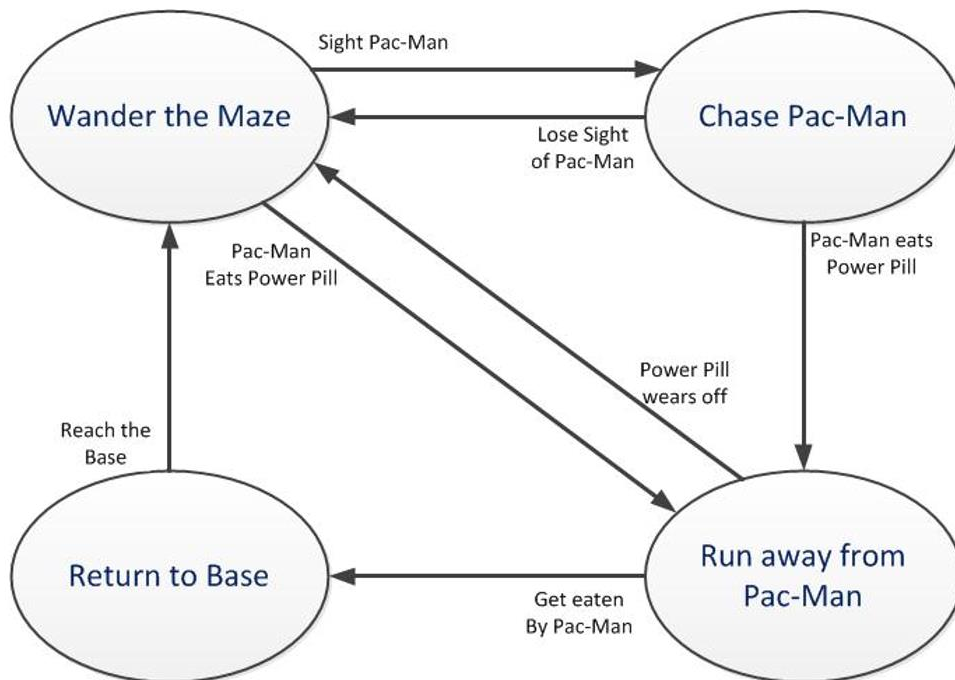
- ▶ Consider the purpose of AI in videogames
- ▶ To provide challenge/immersion for the player
- ▶ If the AI isn't doing that, it's not doing anything useful
- ▶ FSM's are a quick way of changing AI behaviours to keep the encounter, ideally, challenging - at the very least, to make the player feel engaged in some way

Examples of FSMs in Game AI

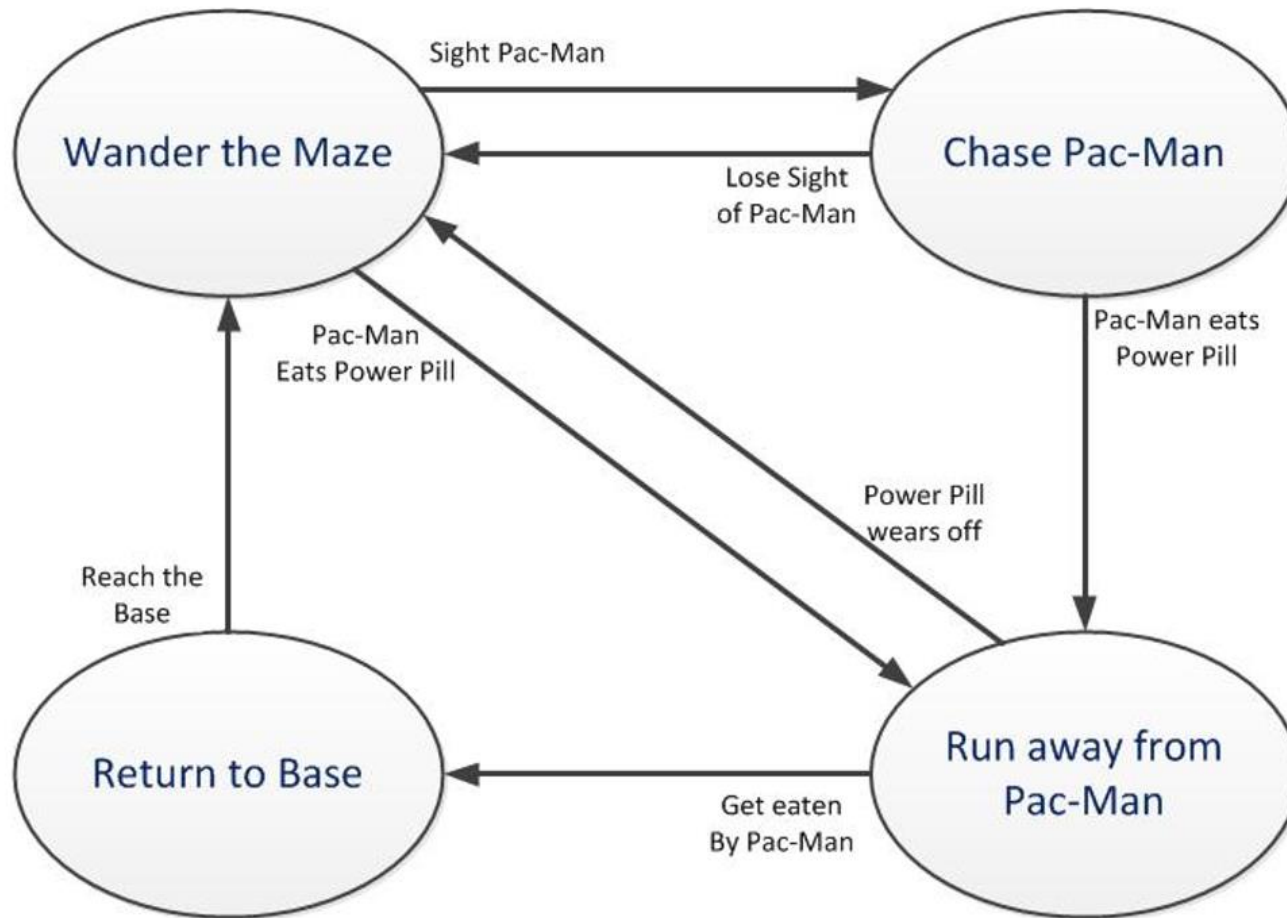
- ▶ One of the most elegant FSMs in gaming comes from ghosts in Pac-man.
- ▶ The ghosts have four behaviours:
 - ▶ Wander the maze
 - ▶ Chase Pac-man
 - ▶ Run away from Pac-man
 - ▶ Return to the central base

Examples of FSMs in Game AI

- ▶ Ghosts change (transition between) behaviours based on triggers in the game environment.
- ▶ We can summarise these transition conditions and their associated states using a STATE DIAGRAM
- ▶ State Diagram's take this form (very examinable)



Examples of FSMs in Game AI



Key Features of a Finite State Machine?

- ▶ The elements of FSMs in the quote are key to -why- it's the building block of game AI
- ▶ State of the agent, connected to
- ▶ Transitions - the process by which the agent changes state, caused by
- ▶ Triggers - events or conditions in the environment which require the agent's behaviour to change

Key Features of a Finite State Machine?

- ▶ The Finite State Machine must always be completely closed
- ▶ This means that there will never be an eventuality which is not covered by the FSM
- ▶ These eventualities can be complex, design-related problems
- ▶ Or as simple as having a state exit when a variable is >1 , or <1 , but not handling $=1$

Programming a State Machine

State Machine Types

- ▶ Many different ways to approach FSM encoding
- ▶ Vary from Expert-System logic structure to database searching
- ▶ We cover three you're likely to come across or employ yourselves
 - ▶ Hard-Coded Switch Statements
 - ▶ Hard-Coded State Patterns
 - ▶ Interpreted State Patterns

Hard-Coded Switch Statements

- ▶ Since a state machine is notionally an expert system, it can be encoded the same way - a series of conditional if-then statements.
- ▶ E.g.

```
If CurrentState == STATE_1
{
    State1Behaviour();
    If CheckTrigger1to2()
    {
        SetState (STATE_2);
    }
}
```

Hard-Coded Switch Statements

- ▶ This embeds all of the transitional logic in the checks
- ▶ Notice in the example that the behaviour code is contained within a function (e.g. `State1Behaviour`)
- ▶ Once that function updates, the algorithm decides if a transition has been triggered through some other function (e.g. `CheckTrigger1to2` to see if the transition which would move from `STATE==1` to `STATE==2` has occurred)

Hard-Coded Switch Statements

- ▶ This has the benefit of being very quick to prototype
- ▶ Very easy to understand - it's intuitive
- ▶ Very difficult to maintain
- ▶ Difficult to debug
- ▶ Difficult to encode even relatively simple FSMs
- ▶ Not extensible

Hard-Coded State Pattern

- ▶ Slightly more extensible approach to Hard-Coded Switch Statements
- ▶ Leverages benefits of object-orientation and inheritance
- ▶ Create a parent 'State' class, which all other states inherit from.
- ▶ Assign a State member variable to an agent
- ▶ Encapsulate all of that state's logic within the specific state's code, including it's outbound states
- ▶ When a transition occurs, replace 'States'

Hard-Coded State Pattern

- ▶ Might implement an FSM class which manages transitions between states, to centralise the logic slightly
- ▶ If taking this approach, will probably need some form of message-passing implemented where the FSM class instigates a transition when instructed to by a trigger detected within a state behaviour
- ▶ Still not particularly extensible as every time a new state is added, states which interconnect with it need re-encoding/recompiling

Interpreted State Pattern

- ▶ This is the data-driven variant of a hard-coded state pattern.
- ▶ Information regarding state connects and transition triggers is stored outside of the code base, and read in from a data file at start up
- ▶ As this approach is data driven, the make-up of the FSM can be tweaked and tuned without recompiling

Interpreted State Pattern

- ▶ Basically, this is the nascent form of the gameplay script, in principle
- ▶ Can be handed off to high-level designers without any knowledge of the codebase
- ▶ They only need to come back to the programmers when they want a new state encoded - and even then, not necessarily, depending on what states actually entail
 - ▶ If experience level in an RPG is represented by a state, and only contains changing stats, this could also be data driven, as an example

State Oscillation

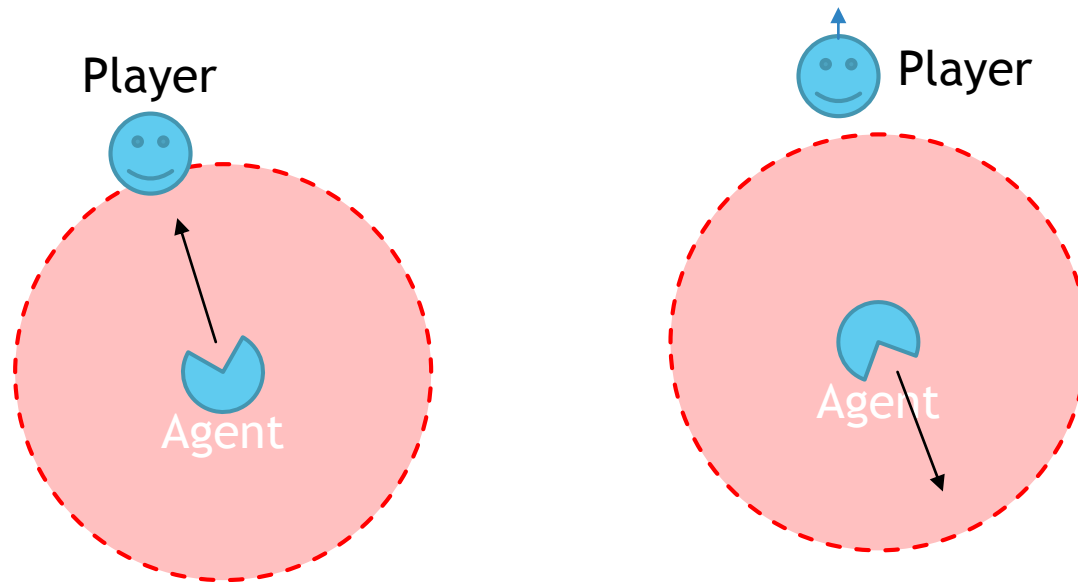
The Problem

- ▶ Envision the following scenario.
- ▶ A hostile AI agent is faced with a player.
- ▶ The agent moves slower than the player.
- ▶ When the player is inside aggro range, the mob moves towards her
- ▶ When the player moves out of aggro range, the mob moves back towards its patrol zone

Player



The Problem

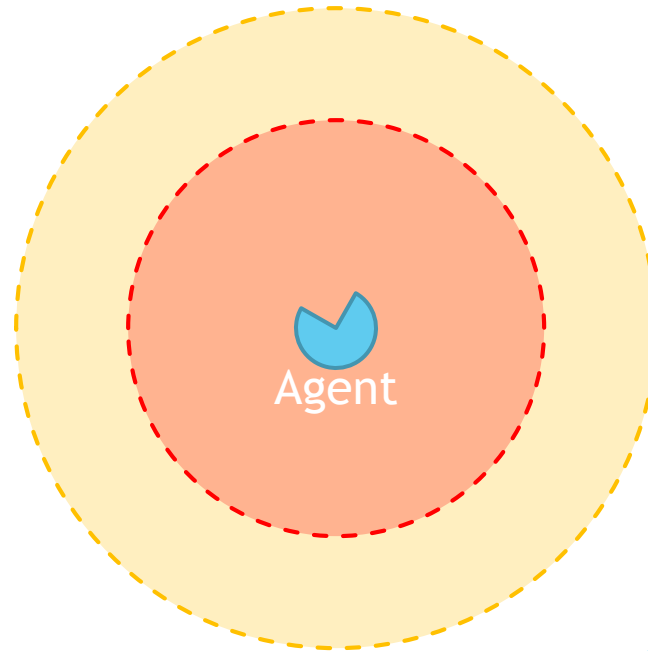


The Problem

- ▶ Easy to see how this maps to other, autonomous scenarios
 - ▶ Altitude adjustment on a plane
 - ▶ A car following a route on the road
- ▶ So how do we solve it?

The Solution

- ▶ Hysteresis
- ▶ We make the boundaries that trigger conflicting states different
- ▶ The player has to run a bigger distance away before the agent will give up
- ▶ Consequently, they have to run even further back to convince the agent to re-engage

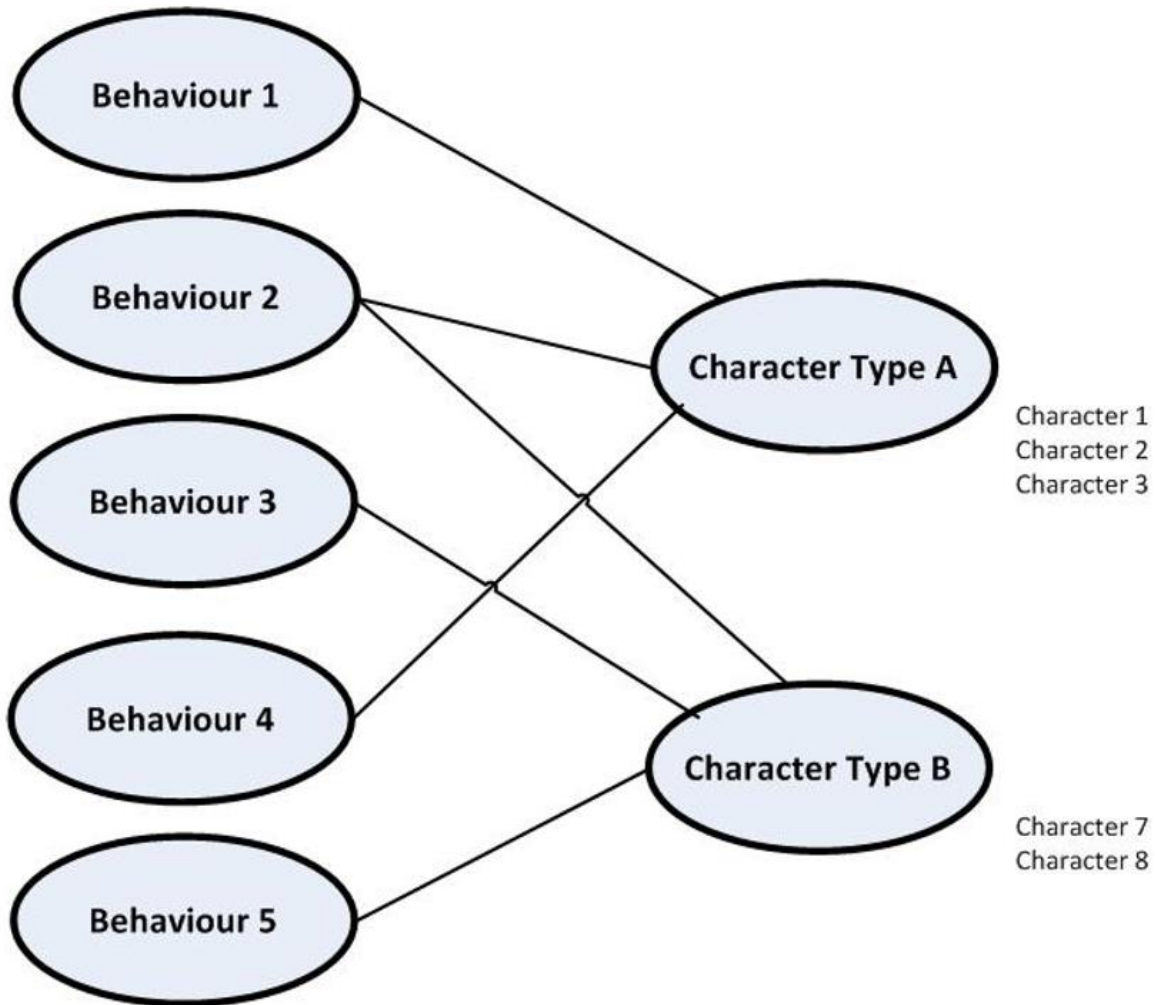


Behaviours and Types

The Difference is Crucial

- ▶ AI Behaviours are individual behaviours which an agent might exhibit
- ▶ AI Types are categories of agents defined by a specific collection of behaviours
- ▶ By structuring our system this way, the AI is efficiently encoded - no unnecessary duplication of behaviours
- ▶ A data driven approach to defining the state machine permits designers to create new AI types simply by defining them as collections of behaviours

Illustrating the Difference

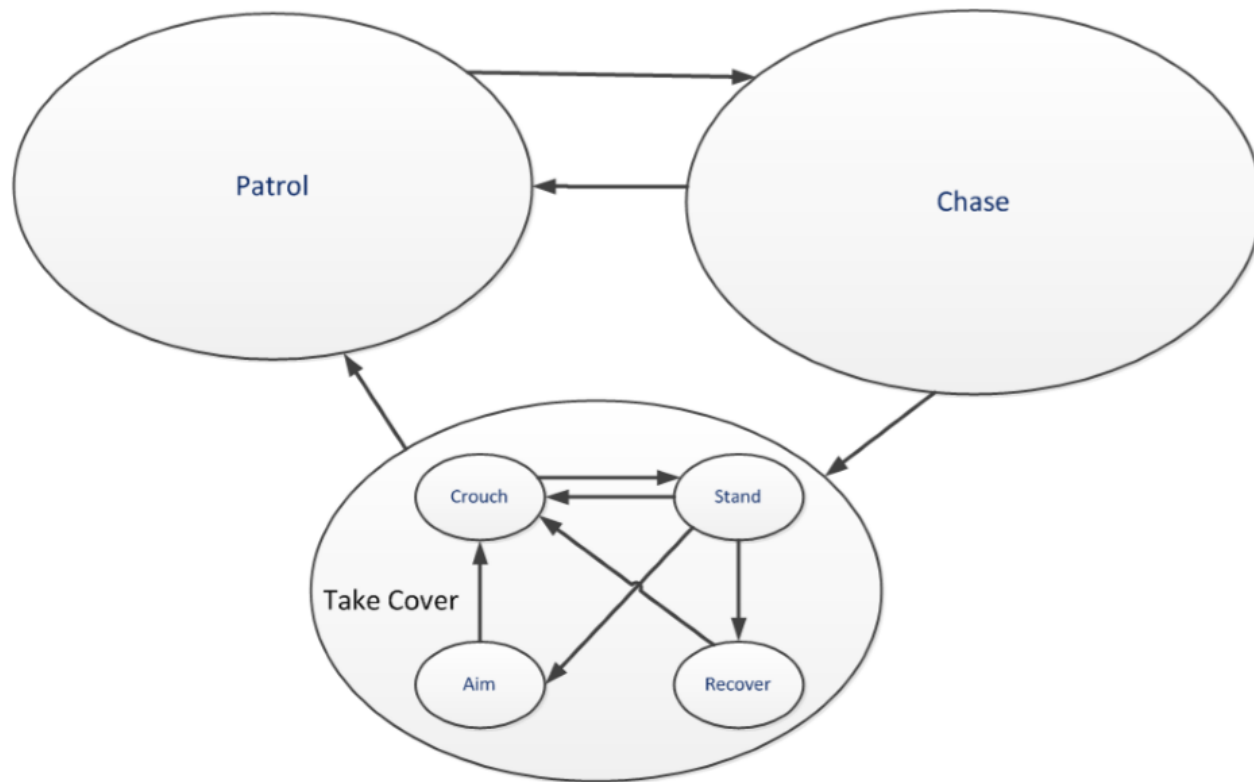


Hierarchical FSMs

Well, if we're defining agents by collections of behaviours...

- ▶ We can do the same for states, yes
- ▶ In fact, this is standard practise
- ▶ If we can define an overarching state by a collection of sub-states with their own context-specific transitions, we do so
- ▶ Consider the example of a guard in a first-person shooter game

Example of Hierarchical FSM



Hierarchical FSM

- ▶ Makes code very extensible
- ▶ Can change one small aspect of agent behaviour without worrying about how it flows to other states beyond the lower-level grouping
- ▶ Permits very sophisticated-seeming AI, at lower computational cost than implementing the same thing with a single level of FSM

Fuzzy State Machines

Fuzzy Logic?

- ▶ Alternative to binary logic
- ▶ More detail in a future lecture
- ▶ For purposes of Fuzzy State Machines, just need to know that fuzzy logic replaces binary logic - instead of something being true or false, it can be partially true.

Fuzzy State Machine

- ▶ If we can say that a state can be partially true, then we can say the behaviours associated with it might be partially followed
- ▶ Consider the example of a police officer who can shoot a player from behind the wheel of his police car or on foot
- ▶ If the police officer takes cover behind the open door of his car, firing through the window void, he can be utilising elements of both the driving and running hierarchical states
- ▶ Encoding this directly would make the FSM significantly more complicated - this way, we have the opportunity to leverage work already done

Summary

- ▶ Introduced Finite State Machines
- ▶ Discussed some fashions in which they can be encoded
- ▶ Considered some flaws with their naïve implementation (added hysteresis)
- ▶ Considered hierarchical FSMs and Fuzzy State Machines (FuSMs)

Implementation

- ▶ There's a very simple hard-coded switch statement FSM provided as sample code
- ▶ Why not extend it into a hard-coded or interpreted state pattern?