

# INFT3960 – Game Production

**Week 09**

**Module 9.1**

**Artificial Intelligence**

# Course Overview

Lec	Start Week	Modules	Topics	Assignments
1	3 Aug	Mod 1.1, 1.2	Course Overview, Design Process	
2	10 Aug	Mod 2.1, 2.2, 2.3, 2.4	Unity3D Introduction, Introduction C#, Variables and Components, Hello World	
3	17 Aug	Mod 3.1, 3.2, 3.3	Booleans, Loops, Lists and Arrays	Assign 1 21 Aug, 11:00 pm
4	24 Aug	Mod 4.1, 4.2	Functions and Parameters, Debugging	
5	31 Aug	Mod 5.1, 5.2	Classes, Object Oriented	
6	7 Sep	Mod 6.1, 6.2, 6.3	Agile Processes, Risks and Prototypes, Testing	
7	14 Sep	Mod 7.1, 7.2	Puzzles, Guiding the Player	Assign 2 18 Sep, 11:00 pm
8	21 Sep	Mod 8.1	Game Physics	
9	12 Sep	Mod 9.1	AI for Games	
10	19 Oct	Mod 10.1, 10.2	Game Interface, Storytelling in Games	
11	26 Oct	Mod 11.1, 11.2	Graphics Pipeline, Animation in Games	Assign 3 1 Nov, 11:00pm
12	2 Nov	Mod 12.1, 12.2	Networked Games, Course Review	

## Course Details

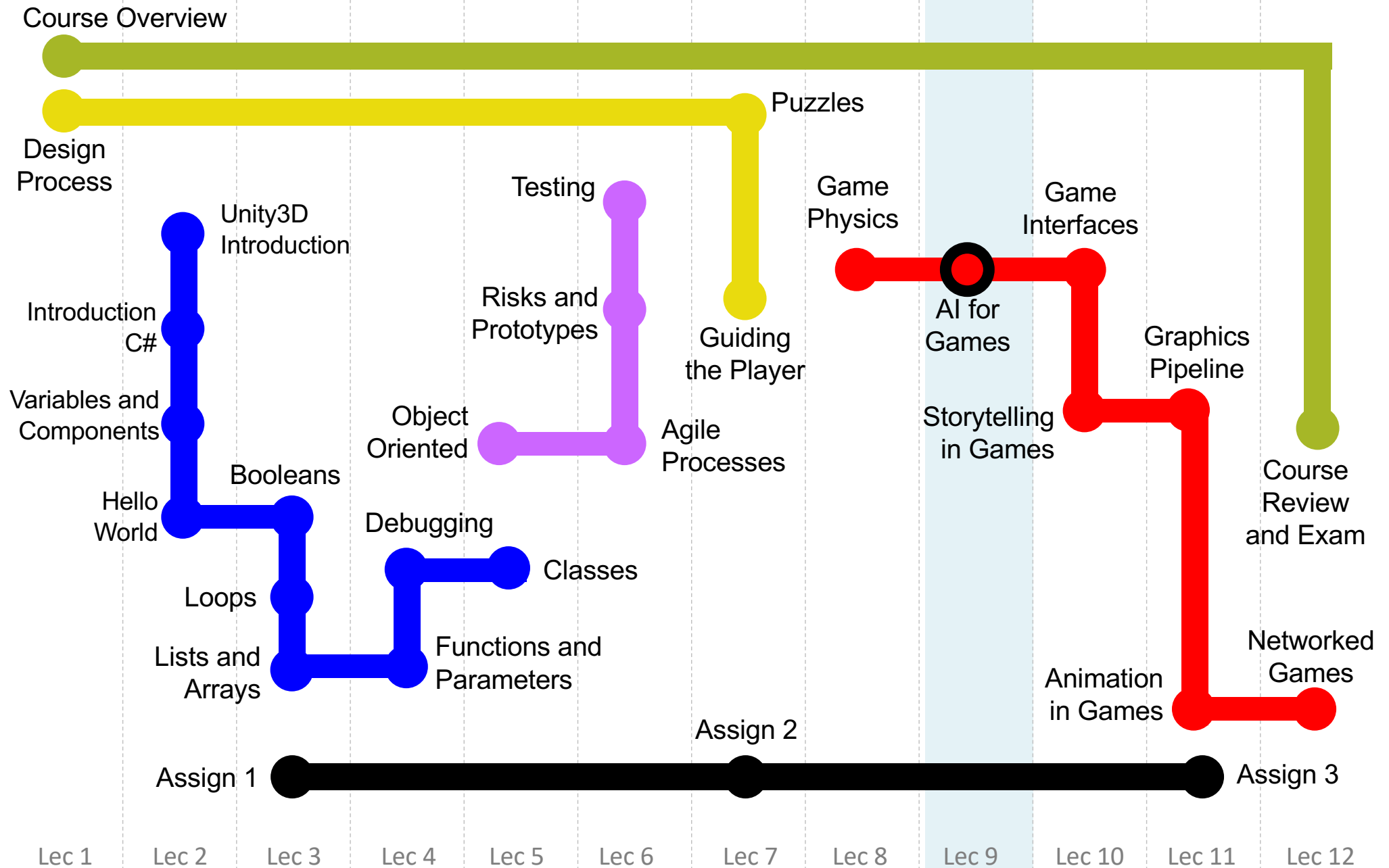
## Game Design

## Unity 3D and C#

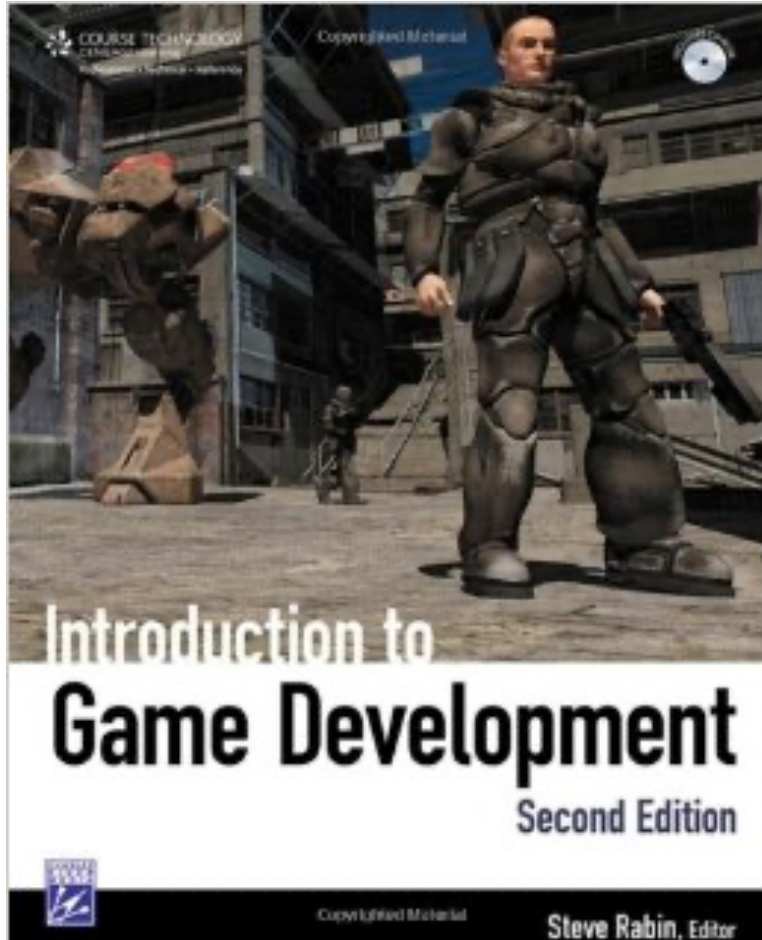
## Development Process

## Core Game Concepts

## Assignments




# Source



Introduction to  
Game  
Development  
  
by Steve Rabin

# AI for Games– Topics

- 
- AI for Games
  - Game Agents
  - Common AI Techniques
  - Advanced AI

# Artificial Intelligence

AI describes the intelligence embodied in any manufactured device.

If we design a character or opponent in a video game that acts on its own, it is generally accredited with possessing AI

# AI for Games

Very different to some academic definitions of AI

In games the goal of an AI programmer is to create both entertaining and challenging opponents while shipping the product on time

The AI must be intelligent, yet purposely flawed

# AI for Games

The AI must have no unintended weaknesses

The AI must perform within the CPU and memory constraints of the game

The AI must be configurable by game designers/players

The AI must not stop the game from shipping



# Game AI Requirements

The AI must be intelligent, yet purposely flawed.

Opponents must present a challenge.

Opponents must keep the game entertaining and fun.

Opponents must lose to the player in a challenging and fun manner.

# Game AI Requirements

The AI must have no unintended weaknesses.

There must be no “golden paths” to defeating the AI every time in the same way.

The AI must not fail miserably or look dumb.

# Game AI Requirements

The AI must perform within the CPU and memory constraints of the game.

Most games are real time and must have their AIs react in real time.

Game AI seldom receives more than 10 to 20 percent of the frame time.

# Game AI Requirements

The AI must be configurable by game designers/players.

Designers must be able to adjust the difficulty level, tune the AI, and occasionally script specific interactions.

If the game is extensible, players can tweak or customize the AI.

# Game AI Requirements

The AI must not stop the game from shipping.

The AI techniques employed must not put the game at risk.

Experimental techniques must be proved early in the development cycle during preproduction.

If the AI is given latitude to evolve or change, it must be testable to guarantee that it doesn't deteriorate when released to millions of consumers.

# Game Agents

eg. Individual units in action-adventure, FPS or RTS games (or 2D Platformers)

3 key stages:

1. Sense – game provides unit with information
2. Think – Unit decides on next action
3. Act – Unit acts, and communicates act to player (if possible)
4. Learning (optional)

▪

# Game Agents - Sensing

## Vision

- Is the object within the viewing distance of the agent?  
(Pythagoras)
- Is the object within the viewing angle of the agent?  
(Dot Product)
- Is the object obscured by the environment?  
(Environmental ray cast)

# Game Agents - Sensing

## Hearing

- Event Driven Notifications, eg gunfire, footsteps

## Communication

- Agents may communicate with each other

## Reaction Time

- Simple timers



# Game Agents - Thinking

## Two Valuable Techniques

- Expert Knowledge
- Search Algorithm

## Expert Knowledge

Often 'coded' by a designer in a scripting language

If-Then rules

e.g. "If you see an enemy that is weaker than you, attack the enemy; otherwise, run away and get backup support"

# Game Agents - Thinking

## Two Valuable Techniques

- Expert Knowledge
- Search Algorithm

## Search

Used for example to choose where to move,  
e.g. Move to a 'cover node' in a navigation network for a FPS

## Flip-Flopping

Once a decision has been made it stays made for a while to prevent race states

# Game Agents - Acting

Only in the acting step is the player able to witness the agent's intelligence

Common  
Actions

- change locations
- play an animation
- play a sound effect
- pick up an item
- converse with the player
- fire a weapon

May be driven by data (designers/animators)

# Game Agents - Learning

## Optional extra step

For many games with agents as antagonists (e.g. Halo) the agents don't really live long enough for learning to be important – so this becomes optional

# Game Agents - Learning

It can make more interesting gameplay, e.g. If player always attacks from left, agents can compensate

So can keep statistical information about past events and act

Some information can be stored in terrain rather than Agent itself (related to smart terrain)

# Game Agents – Artificial Stupidity

Easy to make agents that can dominate the player

Game agents can have full perfect access to all game information, perfect accuracy, and instant reaction time

The point of game AI is to lose! (but in an interesting way)

If Agents always wins – the player will think the game is cheating

# Game Agents – Artificial Stupidity

- Make agents dumb
- Lower their accuracy
- Give them longer reaction times
- Change positions to make themselves vulnerable
- Engage the player one at a time (not as a group)
- Telegraph their attacks

# Common AI Techniques

## Finite State Machines

- Pathfinding

- Decision Trees

- Command Hierarchy

- Dead Reckoning

- Emergent Behaviours

- Influence Mapping

## Level-of-Detail AI

- Manager Task

- Obstacle avoidance

- Scripting

- Terrain Analysis

- Trigger systems

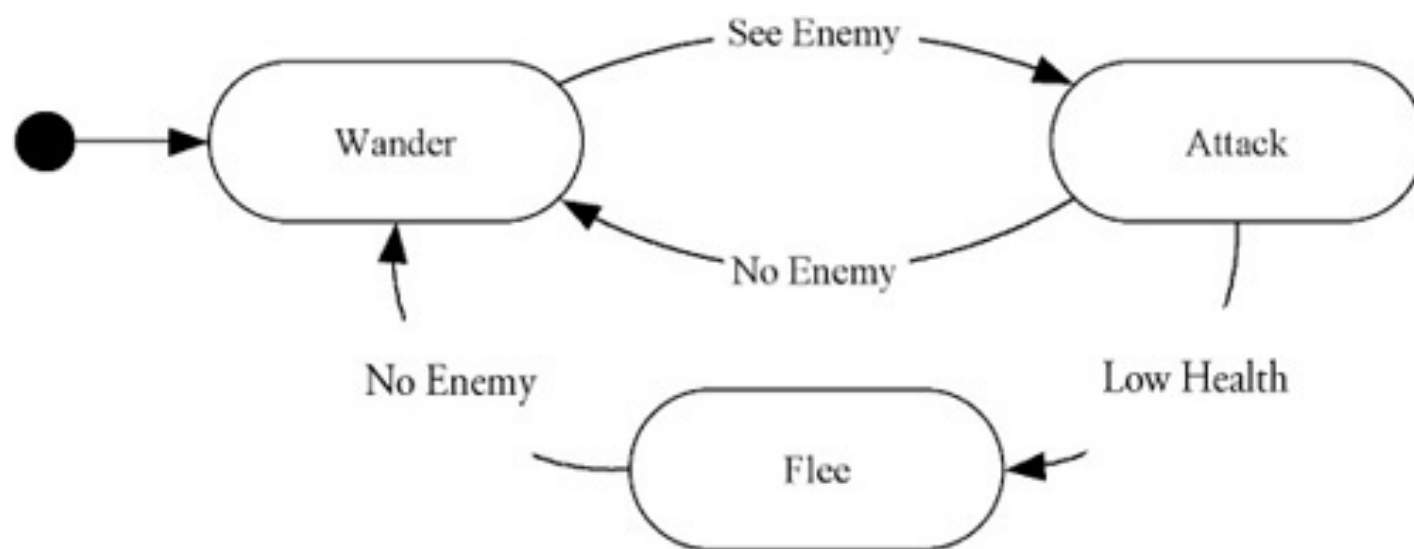
- Advanced AI



# Finite State Machines

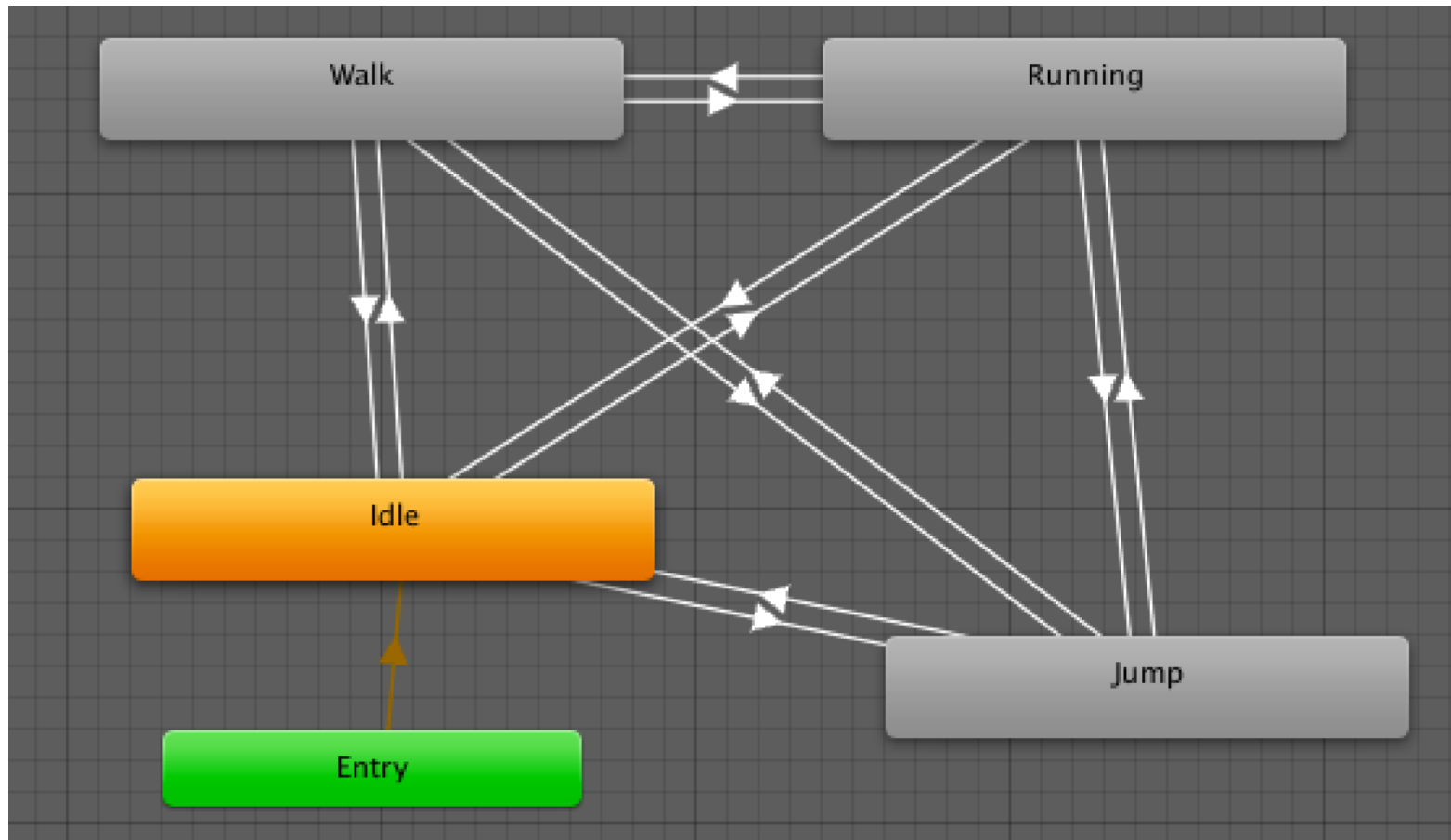
The workhorse of game agent AI

- simple to program
- easy to comprehend
- easy to debug
- can be hard to maintain as project gains complexity



# Finite State Machines

Note: Unity also uses this concept to manage animations



# Finite State Machines

Formally, a finite-state machine is an abstract model of computation that consists of a set of states, a starting state, an input vocabulary, and a transition function that maps inputs and current states to a next state.

Computation begins with the starting state and transitions to new states as inputs are received.

The FSM can perform work within a given state, known as a Moore machine, or on the transitions between states, known as a Mealy machine.

# Game - Finite State Machines

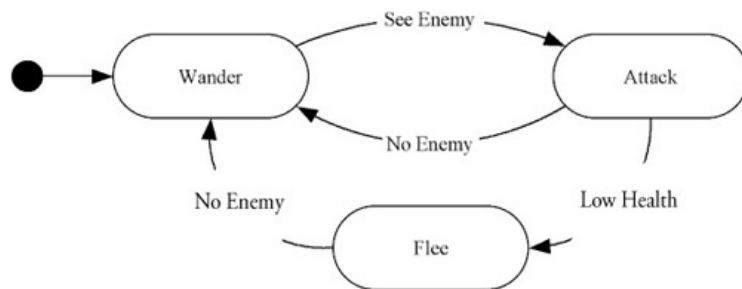
Deviate from the strict definitions

- States may contain behavioural code, or call on information outside of the state machine
- Work may be done both during the state and on transition
- May leverage probability

# Game - Finite State Machines

## Simple Code

```
public class Enemy : MonoBehaviour {  
  
    private enum State  
    {  
        Wander, Attack, Flee  
    }  
  
    private State state;  
  
    public void UpdateState()  
    {  
        switch (state)  
        {  
  
            case State.Wander:  
                Wander();  
                if (SeeEnemy()) { state = State.Attack; }  
                break;  
  
            case State.Attack:  
                Attack();  
                if (LowOnHealth()) { state = State.Flee; }  
                if (NoEnemy()) { state = State.Wander; }  
                break;  
  
            case State.Flee:  
                Flee();  
                if (NoEnemy()) { state = State.Wander; }  
                break;  
  
        }  
    }  
}
```



# Extending - Finite State Machines

Add a stack to track past states – AI can go back to what it was doing before seeing the player

Have a stack of FSMs (hierarchical FSM)

FSM substates

# Common AI Techniques

- Finite State Machines

- Pathfinding

- Decision Trees

- Command Hierarchy

- Dead Reckoning

- Emergent Behaviours

- Influence Mapping

- Level-of-Detail AI

- Manager Task

- Obstacle avoidance

- Scripting

- Terrain Analysis

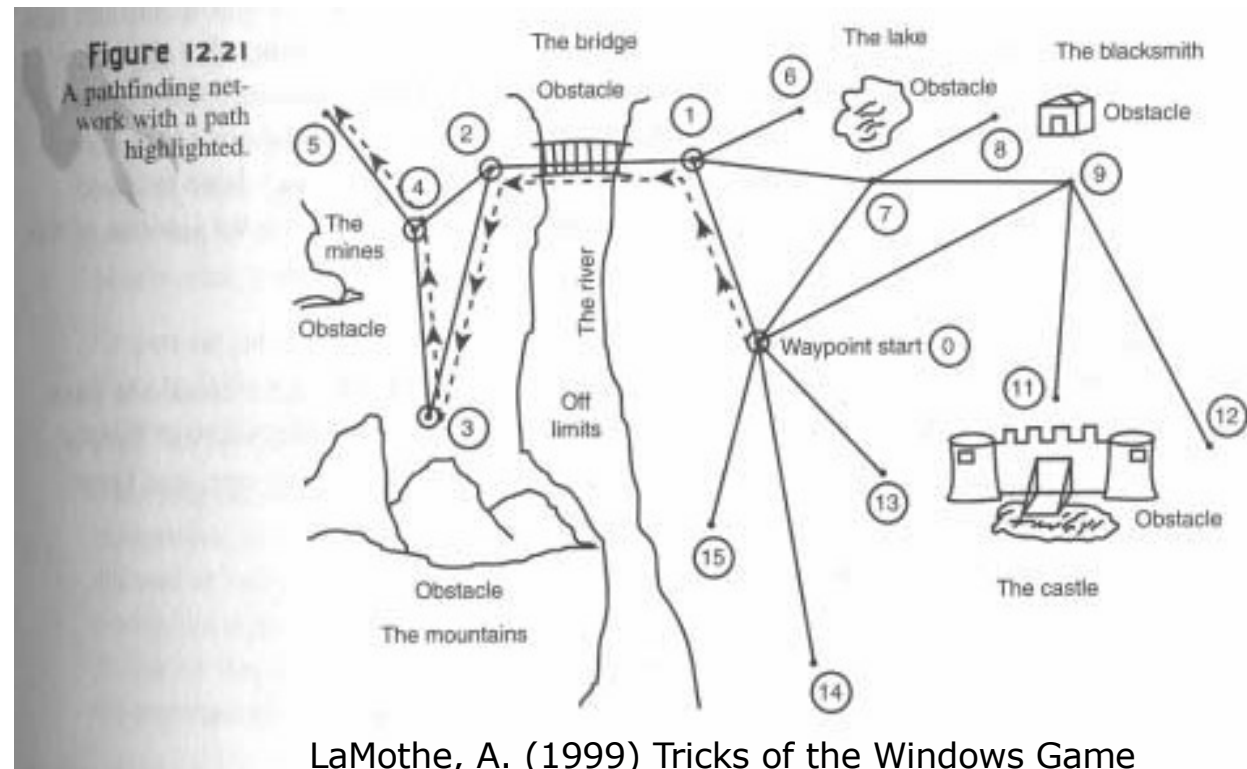
- Trigger systems

- Advanced AI

# Pathfinding

Pathfinding is a similar problem to finding the way through a maze.

Depending on the game world the path may be through large open terrain or involve more enclosed areas.



LaMothe, A. (1999) Tricks of the Windows Game Programming Gurus. New York, Sams Publishing



# Pathfinding - Search Space

## Representing Search Space

To perform pathfinding the pathfinding system needs to understand the level (gameworld)

We need to break up the level in a way the AI can understand

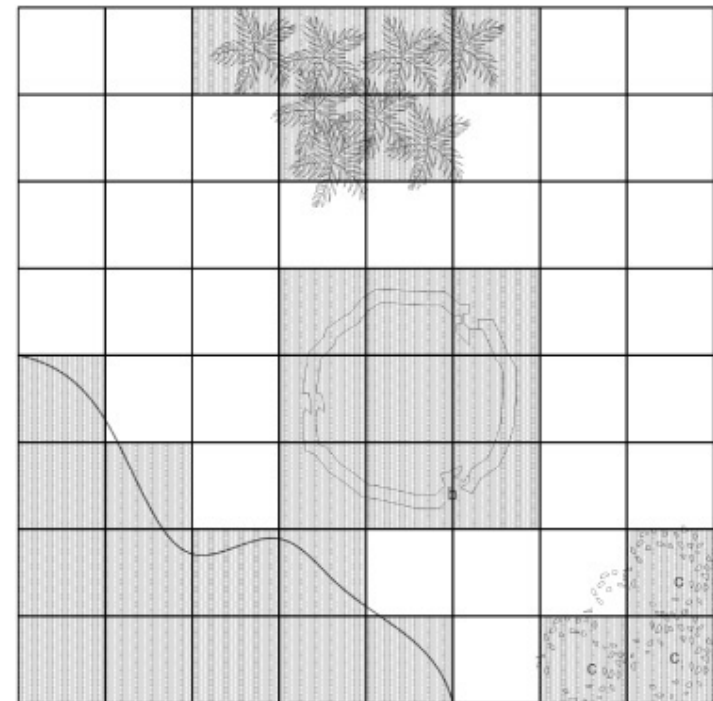
Three most common ways:

- Grids
- Waypoint graphs
- Navigation meshes

# Pathfinding - Search Space

## Grids

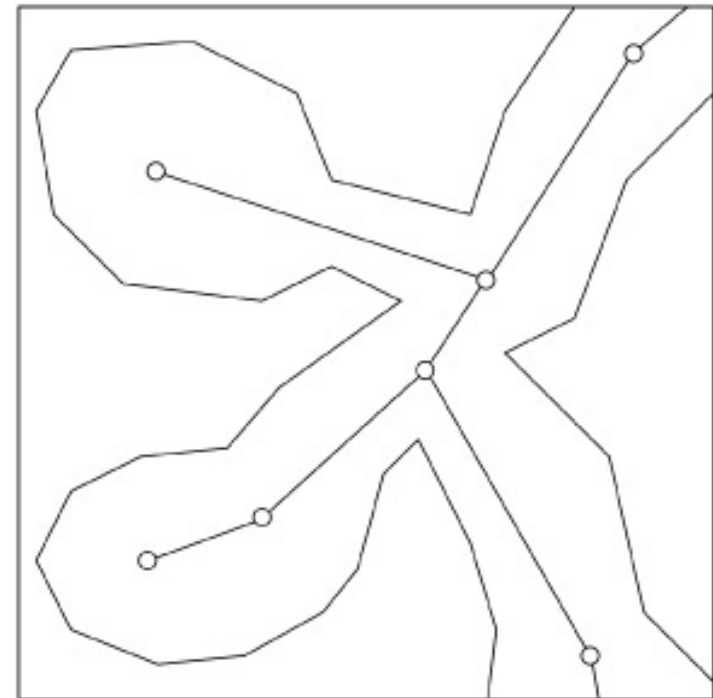
- 2 Dimensional grids are intuitive
- Commonly used for RTS and other strategy games
- Very fast and efficient
- Can only represent 2D
- Levels must align to cells



# Pathfinding - Search Space

## Waypoint Graphs

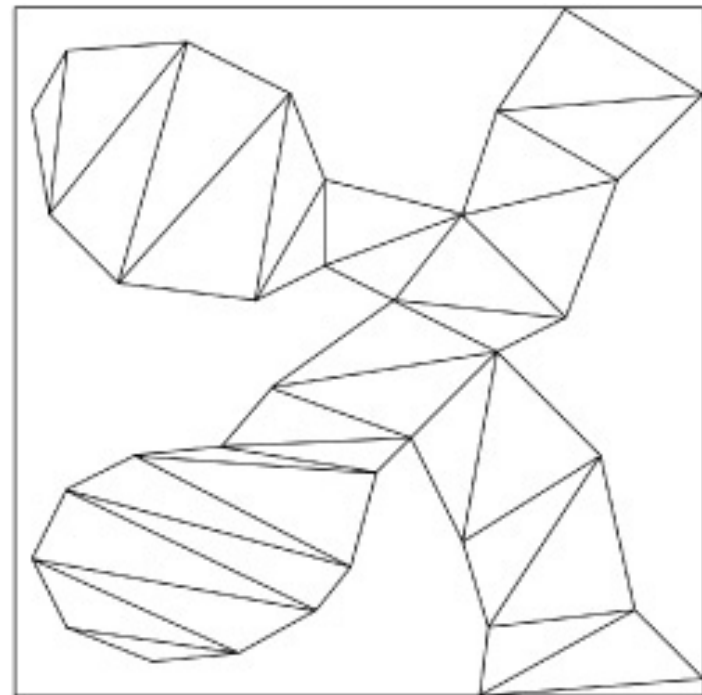
- Specifies the lines in the level that are safe for traversing
- The waypoint nodes are connected through links.
- A link connects exactly two nodes together
- More flexible than grids
- Nodes can contain extra information e.g. cover nodes



# Pathfinding - Search Space

## Navigation Meshes

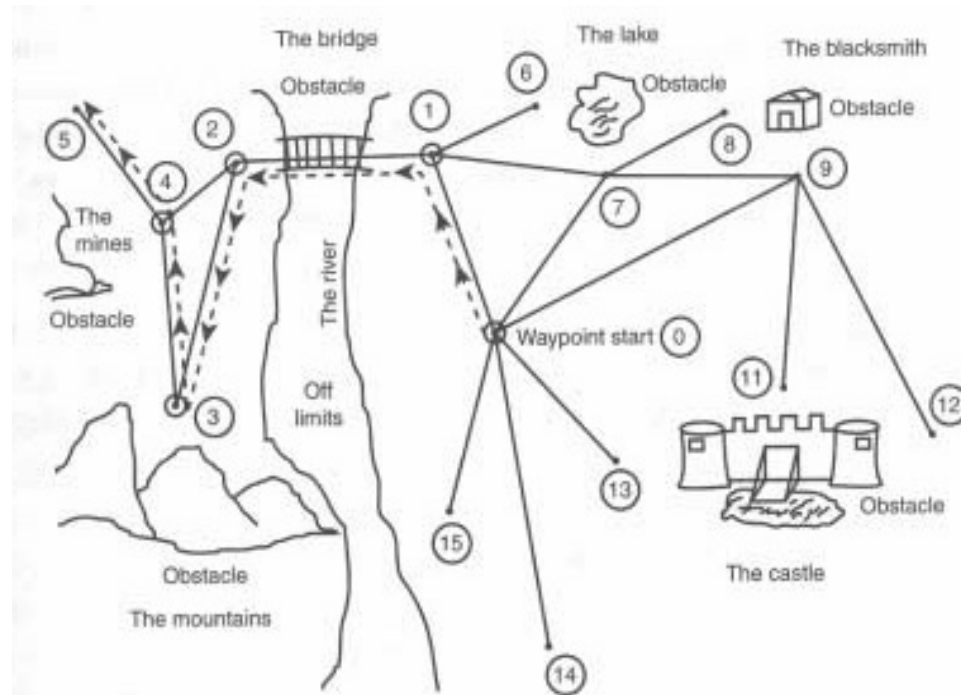
- Fuller representation of traversable areas
- Allows characters to take shortcuts
- Can use mesh to specify where AI can and can't go
- Unity 3D uses Navigation meshes



# Pathfinding – $A^*$

A\* is one of many algorithms for solving this problem. It is one of many graph searching algorithms (works with waypoint graphs)

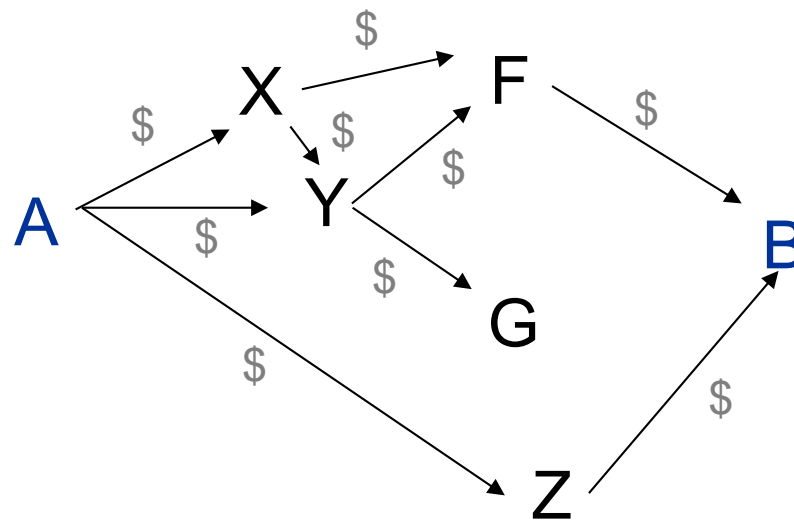
It uses a heuristic (approximate) estimate based on the distance between points and the "cost" of travelling between them.



# Pathfinding – A\*

From a starting location (A) it incrementally searches the routes through various intermediary points to find the route that is most likely to lead to the desired destination (B)

Also calculates the cost of getting there – this may be related to distance, difficulty, etc



(A, X, F, B)

(A, Y, F, B)

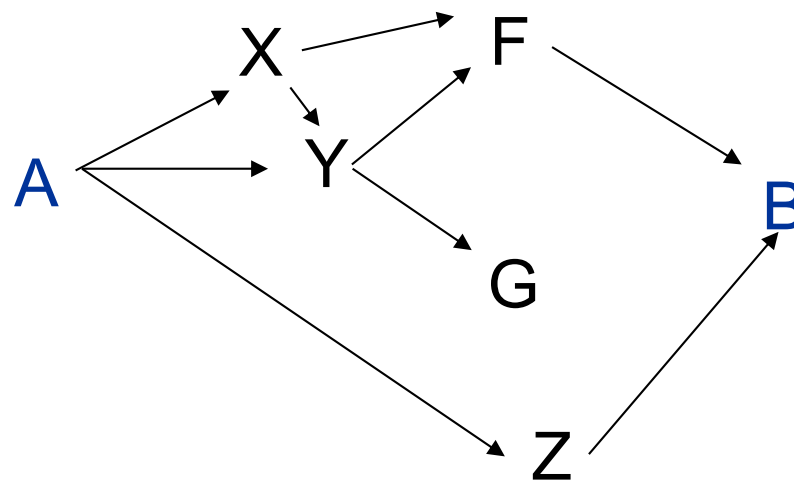
(A, X, Y, F, B)

(A, Z, B)

# Pathfinding – A\*

The algorithm begins with the starting location (A) and then adds all the locations accessible from this node to an open list

These intermediary locations are also called waypoints. The locations on this list (X,Y,F, Z) are then assigned a (distance-cost) measure.



(A, X, F, B)

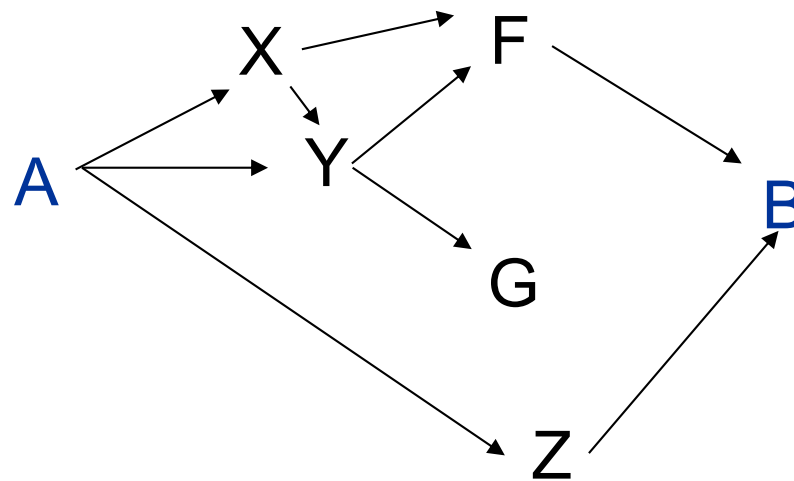
(A, Y, F, B)

(A, X, Y, F, B)

(A, Z, B)

# Pathfinding – A\*

This measure allows the list to be sorted in order of how likely they are of providing the optimal (cheapest/best) route to the destination (B)



\$

(A, X, F, B)  
(A, X, Y, F, B)  
(A, Y, F, B)  
(A, Z, B)



# Common AI Techniques

- Finite State Machines
  - Pathfinding
  - Decision Trees
  - Command Hierarchy
  - Dead Reckoning
  - Emergent Behaviours
  - Influence Mapping
- Level-of-Detail AI
  - Manager Task
  - Obstacle avoidance
  - Scripting
  - Terrain Analysis
  - Trigger systems
  - Advanced AI

# Decision Trees

Decision trees are branching structures that are often used to make high-level strategic decisions.

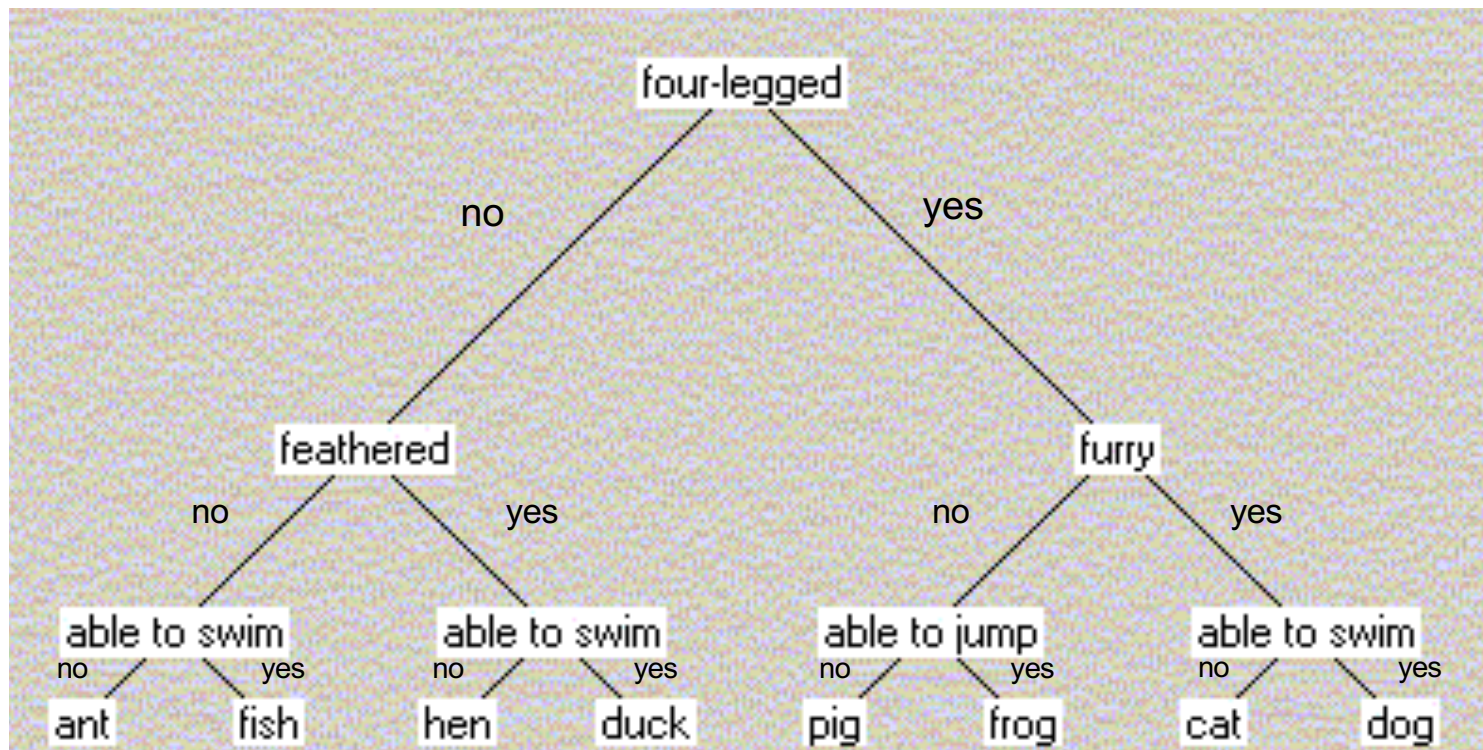
For example, if an NPC in a strategy game should prepare an attack or concentrate on resource gathering.

A different number of choices can occur at each branch.

Where there are only two it is called a binary decision tree.

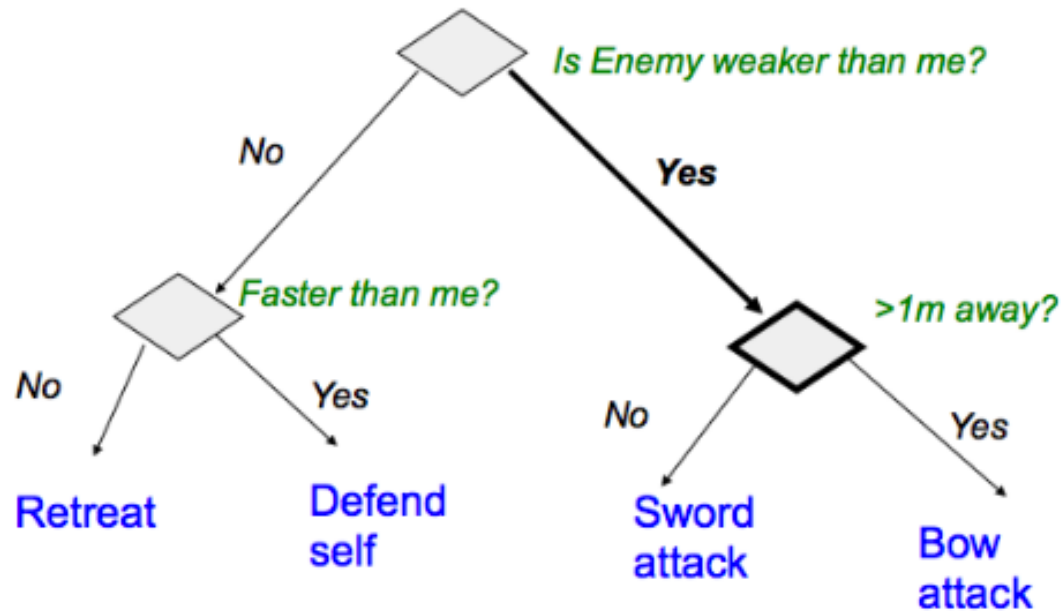
# Decision Trees

eg. A binary decision tree that can be followed to work out the identify of eight different animals.



# Decision Trees

e.g. NPC - Attack or not attack – which weapon ?



# Decision Trees

The branches of the tree are test conditions, which lead to different sub-trees.

A final leaf node contains a resulting decision.

Decision trees, like finite state machines are conceptual design tools and the tree once finalised can be coded by simple selection statements.

# Common AI Techniques

Finite State Machines

Pathfinding

Decision Trees

Command Hierarchy

Dead Reckoning

Emergent Behaviours

Influence Mapping

Level-of-Detail AI

Manager Task

Obstacle avoidance

Scripting

Terrain Analysis

Trigger systems

Advanced AI

# Command Hierarchy

AI has different 'levels' of command (and different levels of complexity)

e.g. General and Troop AI in a Real Time Strategy (RTS) game.

(Armies have a long history of hierarchical organisation so are an obvious use – but many large organisations eg governments can also function in this way)

High level agents (eg generals) come up with strategies to control groups of lower level agents (who may often have much simpler AI objectives). There may be any number of intermediate levels of AI.

# Dead Reckoning

Guesses where the player will be based on their current heading and speed (assuming no sudden changes)

Often used in FPS to estimate where an enemy or player will be – a short time in the future.

Often very important across networks (so in online games) – where some adjustments for lag will need to happen – and so may also require some extra smoothing algorithms

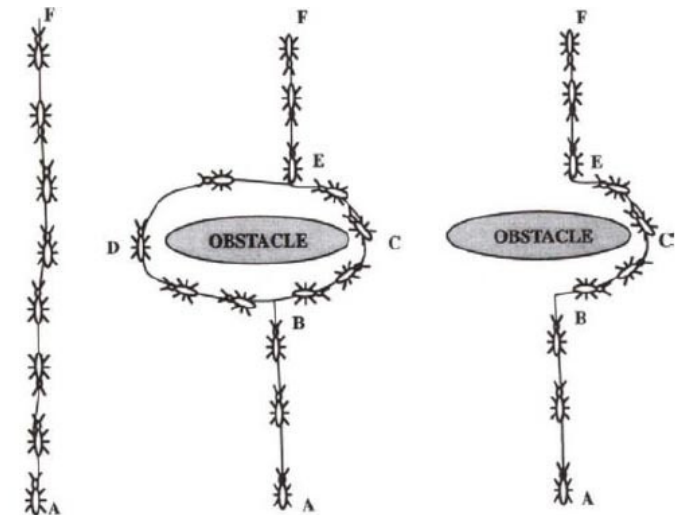
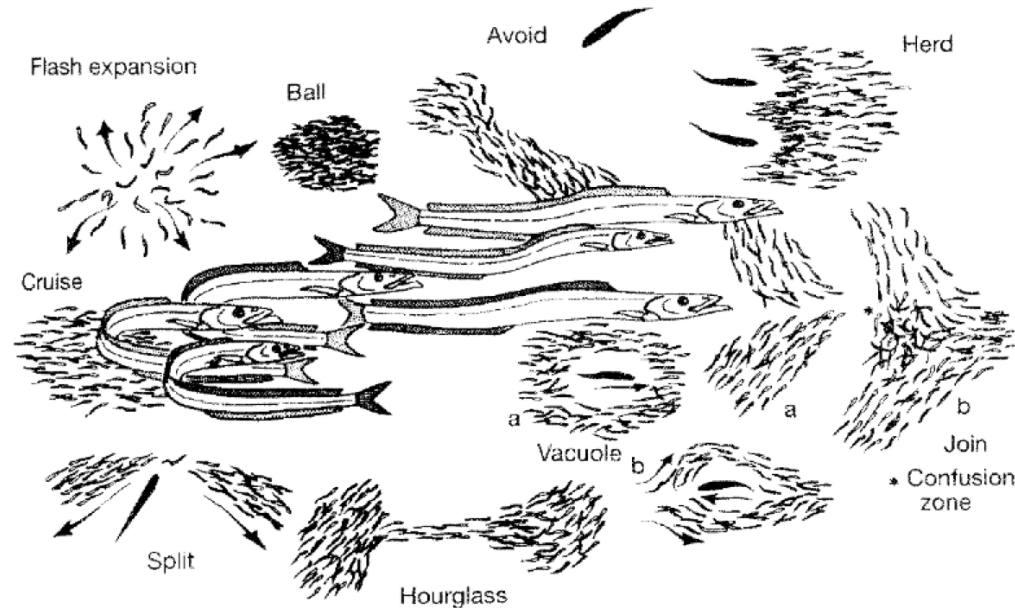
.



# Emergent Behaviours

Such as flocks, ants, societies

Involves the collective motion of a large number of self-propelled entities and is a collective animal behaviour exhibited by many living beings such as birds, fish, bacteria, insects and people.

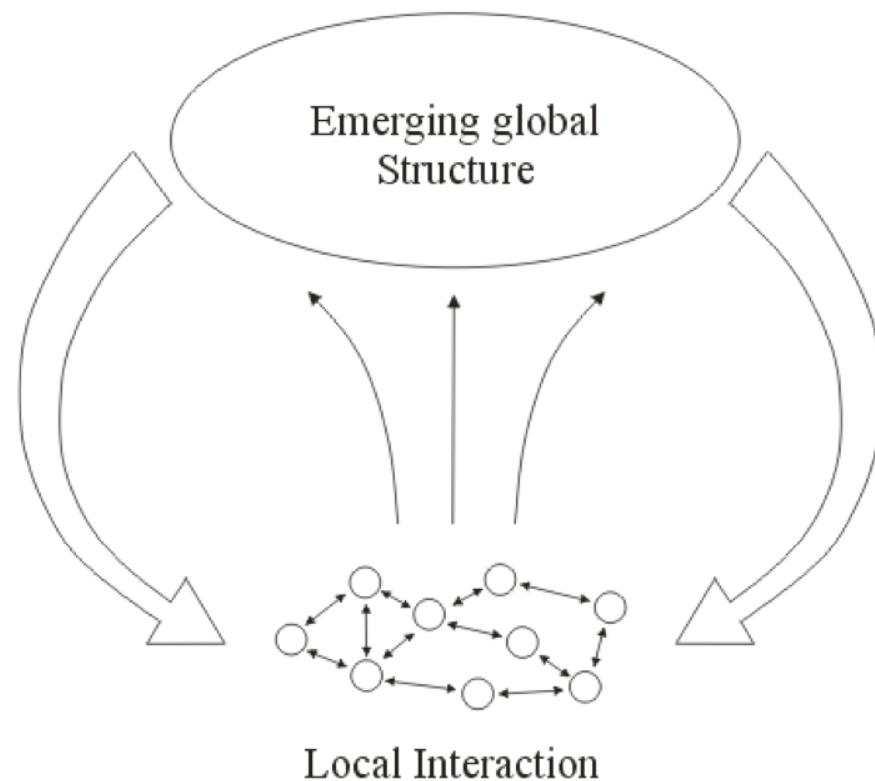


# Emergent Behaviours

Considered an emergent behaviour as it arises from simple rules that are followed by individuals and does not involve any central coordination.

Often studied as *complex systems*

We covered this a bit in Mod 5.2 Object oriented – (also see Chapter 27 of textbook – “boids”)



# Influence Mapping

Originally used in RTS games but are now used in FPS etc

Viewing the distribution of power within a game world.

Can consider 3 types of information for decision-making

## Situation Summary.

Who's in control of what area?

Where are the borders between the territories?

## Historical Statistics

Was this area being assaulted?

How well did my previous attack go?

## Future Predictions

Where did the enemy go?

Where will his influence extend to in the future.

# Level-of-Detail AI

## Level of Detail (LOD)

Typically used for graphics – simpler models (less polygons) are used when an object is far away – but same can apply for AI

Uses simple AI in situations where it won't be noticed by the player, e.g. in crowded battle scenes, guards are a long way away, etc

# Manager Task Assignment

A separate AI to assign tasks to unit AI based on the situation

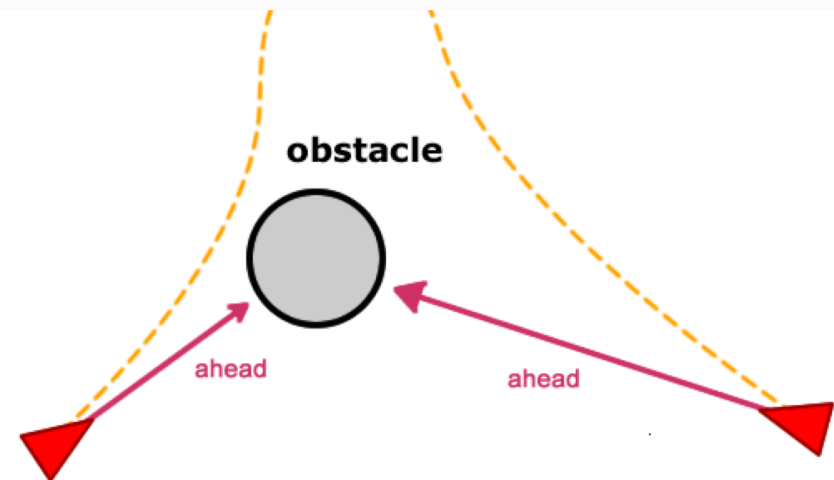
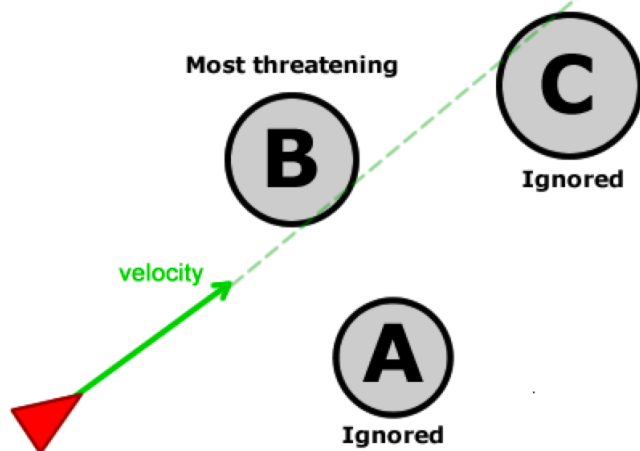
The task manager AI constantly looks at the AI's internal state to determine whether new tasks should be assigned.

Tasks have priorities, and are placed on a Priority Queue when assigned. (These are removed when completed). new tasks are added as needed – eg If enemies are sensed, it adds high-priority DestroyThreat tasks to the task queue.

When the agent is done with its current task, it can be assigned another one off the task queue.

# Obstacle Avoidance

Avoid dynamic obstacles in blocking unit paths – only close threatening obstacles need to be considered – although depends on how far ahead the AI is “seeing” ahead.



# Scripting

Often a specialist language or software may be written to allow a non-programmer to work on AI design.

This is especially true of commercial games written in C++ (which can be very technical – even more so than C#)

It allows fast simple coding of short scripts – actions that occur when something happens. (eg. an event such as a collision, reaching a certain point, completing a challenge or sub goal...}

# Terrain Analysis

AI considers some abstract information about the terrain for decision making

Especially where terrain is automatically generated (so can't pre-plan things)

Identify strategic terrain, e.g. chokepoints (bridge), height advantage in battle,...

Related to pathfinding, obstacle avoidance, influence mapping.



# Trigger Systems

Trigger events and script based on location triggers

Often just implemented as if/then/switch code based on a player triggering an event – eg reaching a location and some state

These are managed in an event cue – that usually get processed after normal updates occur (This avoids complicating the normal update cycle)

# Agents / Robots / NPCs

A lot of the AI in Games has much in common with AI used for robots (eg planning, obstacle avoidance, pathfinding, task management).

For games the situation can generally be simplified – as you know more about the game world and its constraints than the real world (also it's not so bad if a software agent walks off a cliff)

This is important because we often need much faster thinking times in NPCs - and often we want them to be a bit “stupid” – so the player can win (after a good fight).

# Advanced AI

Finite State Machines

Pathfinding

Decision Trees

Command Hierarchy

Dead Reckoning

Emergent Behaviours

Influence Mapping

Level-of-Detail AI

Manager Task

Obstacle avoidance

Scripting

Terrain Analysis

Trigger systems

Advanced AI

# Advanced AI Techniques

Some of these techniques are used only in very high budget AAA titles.

Some are used for experimental indie titles

Some of these techniques aren't used at all yet, but show promise for future games

Even then they are used sparingly, with far more emphasis on more common AI techniques such as State Machines and navigation meshes.

# Advanced AI Techniques

## Bayesian Networks

perform complex reasoning when faced with uncertainty.  
States, features, or events in the game world are represented as nodes in a graph

## Blackboard Architecture

Post problem on a shared communication space, called the blackboard  
Expert systems then propose solutions

## Decision tree learning

Alter decision trees based on experience.  
Used in Black and White

# Advanced AI Techniques

## Fuzzy Logic

An extension of classical logic based on a probabilistic state

## Genetic Algorithms

Technique for search and optimisation based on evolutionary principles

## N-Gram Statistical Prediction

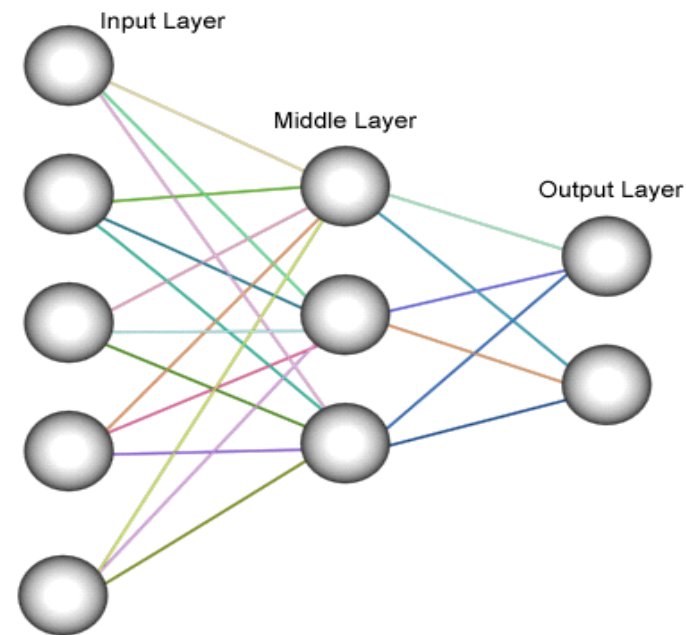
Statistical technique that can predict the next value in a sequence. Can be used to predict player patterns

# Advanced AI Techniques

## Neural Networks

series of identical nonlinear processing elements (analogous to neurons) connected together in a network by weights (analogous to synapses)

Can learn to respond based on inputs



# Advanced AI Techniques

## Perceptrons

A single-layer neural network

It can be used to learn simple Boolean decisions such as attack or don't attack

Used in Black and White

## Player modelling

Build a profile of a player's behaviour

## Production System

Capture expert knowledge in the form of rules in a database and an inference system



# Advanced AI Techniques

## Reinforcement Learning

Discover solutions via trial and error

## Reputation Systems

Model the way a player's reputation in the world develops and changes

## Smart Terrain

Put intelligence into seemingly inanimate objects.  
e.g. The sims

## Dynamic Difficulty Balancing

Automatically adjust the difficulty of the game to the players (ability and type of play)

# Summary

A lot of different AI techniques have been applied to games – these algorithms can be challenging to understand – they often have a strong mathematical foundation.

But AI is usually simpler to do in games than it is in the real-world (as the game world has a lot of constraints).

Players just need estimates of intelligence in NPC (and usually don't want enemies to be too clever anyway)

Often processing speed is a constraint in implementing AI

In the future we can expect more advanced techniques to be integrated (as processing speed improves)