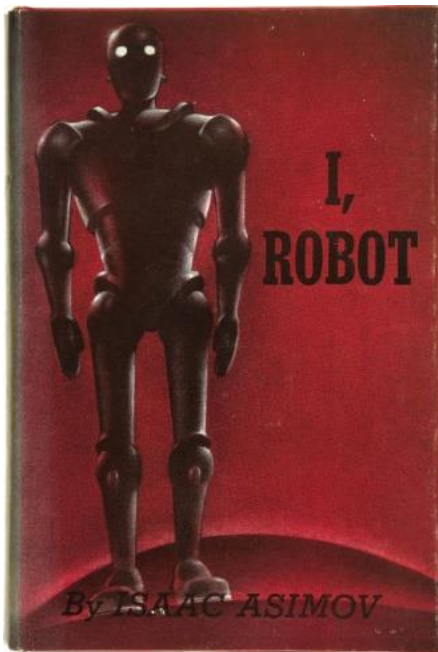


AI Tutorial 3: AI in Games Development



New Concepts

- ▶ What is AI for? Overview of motivation
- ▶ Decision Trees
- ▶ Rubber Banding
- ▶ Adaptive AI
- ▶ Machine Learning
- ▶ Engineering an AI System

What is AI for?

- ▶ Immerse the player - believable
- ▶ Challenge the player - intelligent
- ▶ Entertain the player - beatable

- ▶ “Make the player feel like a gaming god”

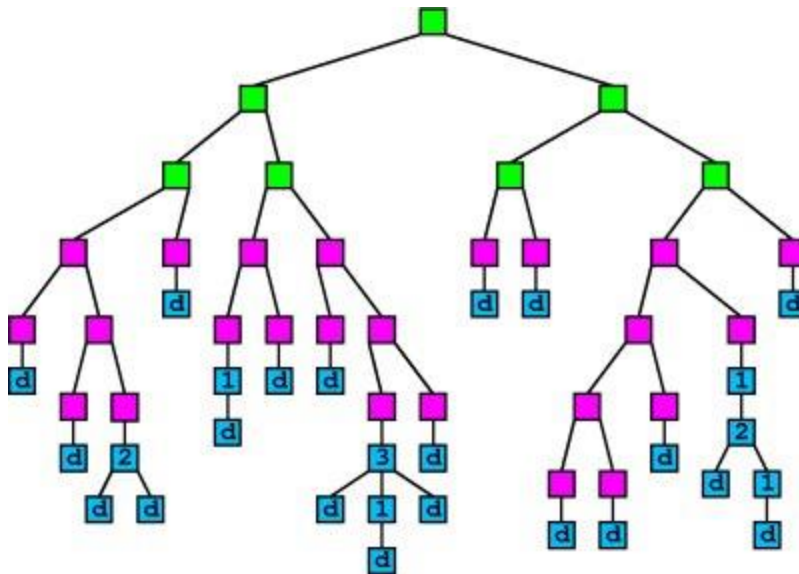
What is AI for?

- ▶ Must be reliable - millions of players, countless situations.
- ▶ Must be efficient - can't distract processor scheduling from other, more 'overt' immersion technologies (graphics, physics, sound, etc.)
- ▶ Randomness vs. Predictability
 - ▶ Pattern recognition and training on the part of the player (MMORPGs)
 - ▶ Repetitiveness and boredom (What do MMOs have that mitigates this problem?)

Decision Trees

Decision Trees

- ▶ Most common use of AI techniques: making decisions
- ▶ If a decision can be broken down into chains of binaries, can use decision tree for modelling
- ▶ Simple example is telephone banking



Decision Trees

- ▶ Needs to be applied with care; technically, everything is a DT
- ▶ Don't let that dissuade you from using them where appropriate
- ▶ But don't try and shoe-horn them into a problem they aren't optimal to solve

Embedded Information

Embedded Information

- ▶ Already discussed this specifically in the context of navigation
- ▶ If we can embed data about an environment into a map, why can't we apply this process elsewhere?
- ▶ We can.

Embedded Information

- ▶ AI information can be attached to world objects, not just hidden in the map.
- ▶ Simple, mechanical example: a +2 Sword can embed its bonus as a variable.
- ▶ Invisible objects can trigger behaviours.
- ▶ In physics, we talked about collision detections which don't elicit a collision resolution - this is what we mean.
- ▶ Many potential layers of additional data.

Interaction Scripts

Interaction Scripts

- ▶ Govern behaviour between two types of agent.
- ▶ Natural extension of our earlier discussion of interpreted state patterns being applied to Character Type/Character Behaviour structures
- ▶ Rabbit encounters Carrot: Eat
- ▶ Rabbit encounters Fox: Run like Hell

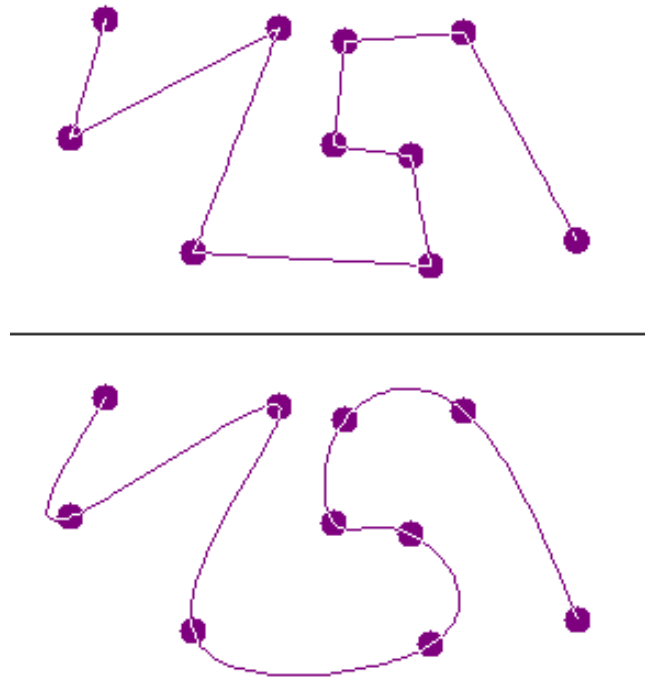
Interaction Scripts

- ▶ Since we can define AI types as collections of behaviours, it's notionally possible to define behaviours related to specific AI types
- ▶ Usually not in C++
- ▶ Instead, written in something accessible which can be interpreted at runtime e.g. Lua
- ▶ Requires detailed design work upfront

Splines and Rubber Bands

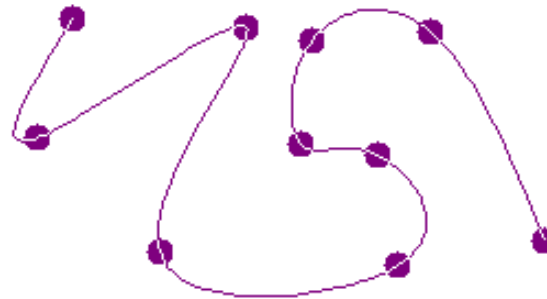
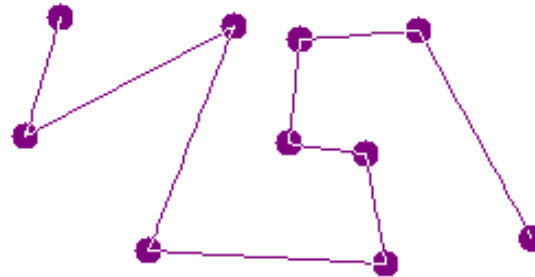
Spline following

- ▶ Already discussed splines in the context of path-finding
- ▶ What about games where the 'path' is reasonably well-defined and unlikely to significantly change?
- ▶ Track-based racing games



Spline following

- ▶ Principle remains the same
 - establish waypoints about the track, generate a curve which is defined by them and parameterisation
- ▶ Shorter splines for overtaking - this is always a reactive process
- ▶ Can introduce branches to splines for short-cuts, etc.



Rubber-banding



- ▶ AI in racing games (and any other game, really) cheats
- ▶ Rubber -banding is a means of ensuring that the AI is always filling it's main role - challenging the player

Rubber-banding



- ▶ It's no fun in a racing game if the player gets so far behind the other drivers that they can never catch up
- ▶ It's not that much fun if the player gets so far ahead of the pack that they can't threaten the player

Rubber-banding



- ▶ Rubber-banding accounts for this in a number of ways.
- ▶ Can be simple - NPCs slow down if too far ahead, speed up if too far behind
- ▶ Can be masked by design - Mario Kart doesn't give you a blue shell or a lightning bolt if you're already at the front of the line

Rubber-banding



- ▶ Conceptually, this is also the root of normalised difficulty in level-up games
- ▶ Consider any game with scaling difficulty - it is basically rubber-banding
- ▶ By scaling the difficulty of enemies around you (a la Dragon Age or Guild Wars 2) no content is ever considered trivial or 'un-challenging'

Scripted Events

Scripted Events

- ▶ Cut-scenes and set-pieces
- ▶ Player can't interact with it
- ▶ 100% reliable
- ▶ Can be complex and give wow-factor

Scripted Events

- ▶ Can lead to novel extensions of engine technology - consider the cut scenes in Starcraft 2, which use technology born out of Blizzard's 3D animation department
- ▶ Far cry from the in-game-cut-scene elements of Final Fantasy VII, where you had beautifully rendered CG FMV acting as an animated backdrop



Adaptive AI

The background of the slide features abstract, overlapping geometric shapes in various shades of blue, ranging from light sky blue to deep navy blue. These shapes are primarily located on the right side and bottom of the frame, creating a modern, tech-oriented aesthetic.

Welcome to the Future

- ▶ Adaptive AI is a thing
- ▶ But there's a difference between what researchers consider “adaptive AI”, and what games consider adaptive AI



What do games consider adaptive AI?

- ▶ Normally, it has to do with adaptive difficulty
- ▶ Less of a ham-fisted approach than level-capping in GW2 zones
- ▶ Normally appropriate in scenarios where the difficulty can't solely be summed up in mechanical numbers
 - ▶ The difference between heroic and normal raid content (mechanics and tactics change), not the difference between a shotgun doing 1-6 damage vs 2-8 damage

Adaptive AI

- ▶ System first needs to detect when the player is struggling
- ▶ Then, the system needs to dynamically change the AI in the game to compensate
- ▶ Generally, this is to make them easier to kill, or give them a harder time killing the player
- ▶ More subtle than simply capping values

Adaptive AI

- ▶ Recent title Alien: Isolation does this in reverse
- ▶ If the player is doing too well, the Alien adapts its strategies to try and catch the player out
- ▶ So, Adaptive AI to make the game harder

Machine Learning, and other animals

Machine Learning

- ▶ This is the process by which the parameters defining AI behaviour are refined based on 'experience'
- ▶ The AI tracks its parameterisation vs. its success or failure, determined by some metric
- ▶ It makes an informed (or non-informed) judgement about how to adapt the parameterisation based on success or failure
- ▶ In its simplest implementation, this is similar to what Alien: Isolation does - if the player favours one approach to evade the xenomorph, the xenomorph changes its tactics in an effort to counter that

Machine Learning

- ▶ Implementation in games is limited, as is application
- ▶ If the AI is too easy to beat (hasn't been 'trained' enough), players won't replay the game
- ▶ If the AI learns to counter everything a casual gamer does, no fun
- ▶ Neither of these scenarios is likely, but they're possible - also, all of these facts are true of the other methods we discuss after this

Genetic Algorithms

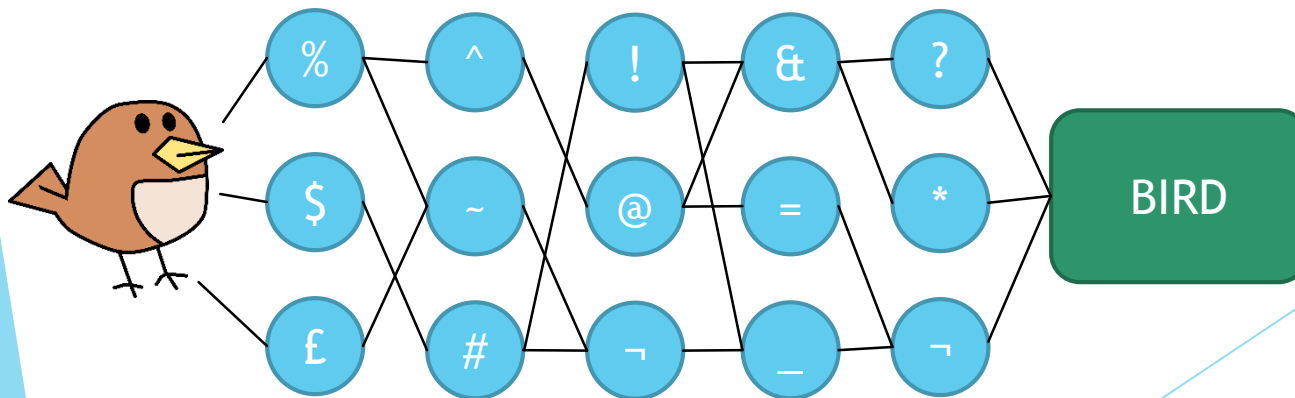
- ▶ Consider how we parameterised our agent's world state in our GOAP example
- ▶ Now imagine that those 1s and 0s were actually used to score how 'good' an NPC is (good is subjective - part of the problem - we need a metric for it). We'll call this their fitness, and the collection of 1s and 0s their type.
- ▶ Based on our metric for fitness, we assess NPCs with a variety of types. The least fit are less likely to 'pass' into the next generation, while the most fit are more likely to
- ▶ Over many generations, our population of NPCs statistically becomes awesome

Genetic Algorithms

- ▶ That was GA's in a nutshell
- ▶ We don't use them often in game technology for the reasons outlined when we discussed machine learning
- ▶ Computationally expensive to refine new generations, assuming we can even get the data to do so
- ▶ Has to be done offline
- ▶ Difficult to parameterise fitness when the NPC's purpose is fun, not survival
- ▶ Ultimately self-defeating

Neural Networks

- ▶ Training (tuning) a network of arithmetical operations to favour a certain output based on a certain type of input
- ▶ Allegedly used by Codemasters
- ▶ Really, doesn't see that much use in the games industry - though apparently some fighting games are messing around with them



General Observations

- ▶ All of the above are Artificial Learning Techniques
- ▶ Problem with learning techniques is three-fold
 - ▶ Game often needs to be complete before we can begin development on the AI, much less test it
 - ▶ Parameters involved often aren't human-readable (in a neural net, for example, we can't identify which of the arithmetical operations is making something misbehave, and changing any of them changes their effect on everything - don't exist in isolation)
 - ▶ Requires extensive QA testing to ensure that the final product behaves well, every time

Engineering an AI System

Understand the Requirements

- ▶ More than any other aspect of game engineering, competent AI programming hinges on understanding what the system needs to be able to do
- ▶ Your solutions should, where possible, be elegant
- ▶ But do not choose elegance over functionality; an ANN is 'elegant', but impractical in a great many cases

Understand the Requirements

- ▶ Those requirements can be expected to change during development cycle
- ▶ This is a good thing - not the same as feature creep
- ▶ A well-engineered AI system will be able to adapt to this - notice how all of the techniques we've focused on are intended to permit extensibility with minimal rewriting
- ▶ Important not to over-engineer the AI - most effective 'AI' is actually clever design; Mass Effect doesn't require natural-language-simulation, just VAs

Level of Detail

- ▶ Unlike Physics, AI can employ something akin to frustum culling
- ▶ You don't necessarily need all, sophisticated AI running all the time
- ▶ Your platform doesn't need to be processing the gunfight between a dozen Minutemen and a horde of bandits until the player is at risk of encountering them - in fact, the player's arrival may well trigger the encounter

Level of Detail

- ▶ In a less dramatic example, consider navigation of cars through a city
- ▶ Cars the player can see need AI which reflects the sophistication of a road network (stop lights, passing places, etc.)
- ▶ Cars the player can't see can pretty much do anything they want, so long as they're where you want them to be when the player's about to encounter them

Level of Detail

- ▶ Problem with using this approach too liberally is similar to popping with Mipmaps
- ▶ Think state oscillation, but where if you're 9 feet from the NPC it's a devious boss monster with tactics you'll take weeks to work out...
- ▶ ... But when you're 10 feet away, it's "a bit derpy"

Spread the Load

- ▶ Unlike physics, AI doesn't need updating every frame
- ▶ You can parcel your AI updates into per-frame batches, and handle them over the course of a second or more
- ▶ The player will never know
- ▶ In a more sophisticated approach, you can actually partition specific algorithmic instances (such as an expensive path-finding/planning operation in a civilisation simulator) to be divided across many frames

Empower Designers

- ▶ The more your designers can do without asking you to recode something, the better
- ▶ From the outset, engineer your AI system to be extensible and accessible
- ▶ Tools are your friend - and since they'll all be internal, they don't need to be fully-featured, just robust and understandable by people in the same building

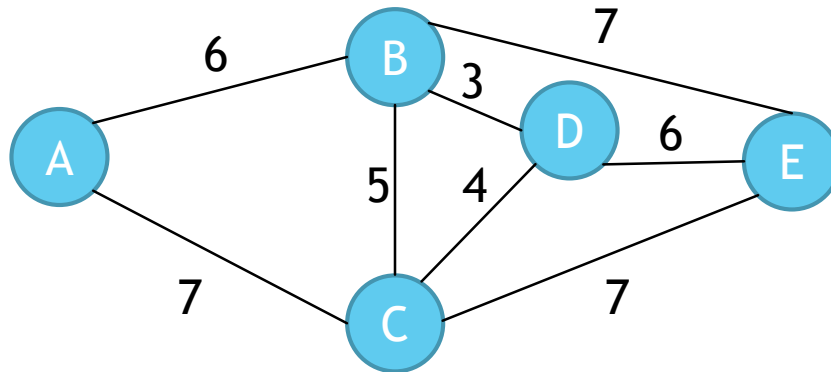
Cheat

- ▶ It's impossible to overestimate the value of cheating
- ▶ You control the world the game operates in - there are no parameters you can't manipulate
- ▶ You don't need to write a complex environment-assessment algorithm to work out what an NPC can see - you can set what they can see
- ▶ You don't need to write an accurate simulation of aiming to determine if an NPC's shot hits - you say whether or not that shot hits

Goal-Oriented Action Planning

Remember our graph?

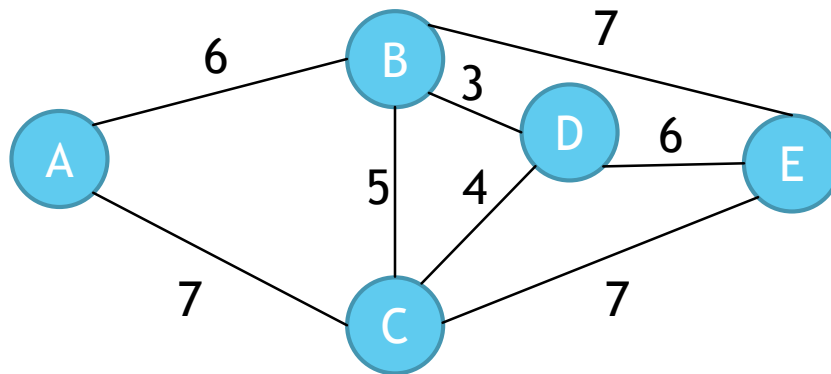
- ▶ Remember how we said the other day that we could represent our environment as a node graph?



- ▶ We add labels to the vertices, and costs to the edges.
- ▶ From here, we work out the cost of traversing the environment from one location to another...

Remember our graph?

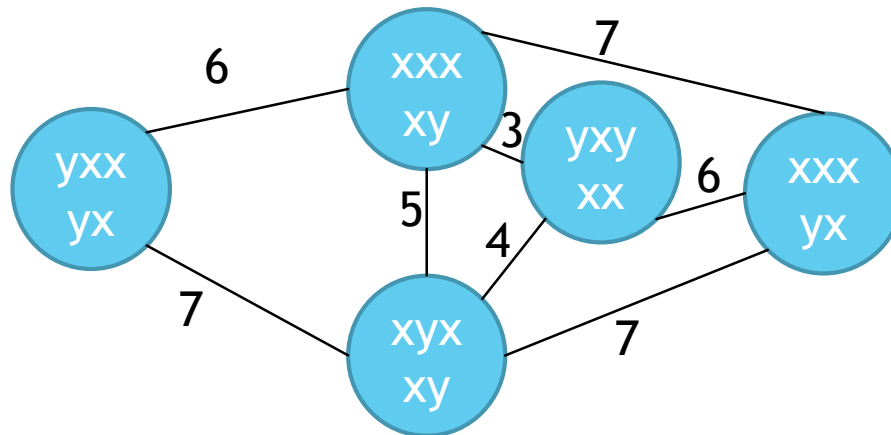
- ▶ What if our interest weren't geographical navigation but were, instead, manipulating the world into a configuration that suits us?



- ▶ Sort of like Dune, but with fewer sand-worms

Combining State Machines and Path-Finding

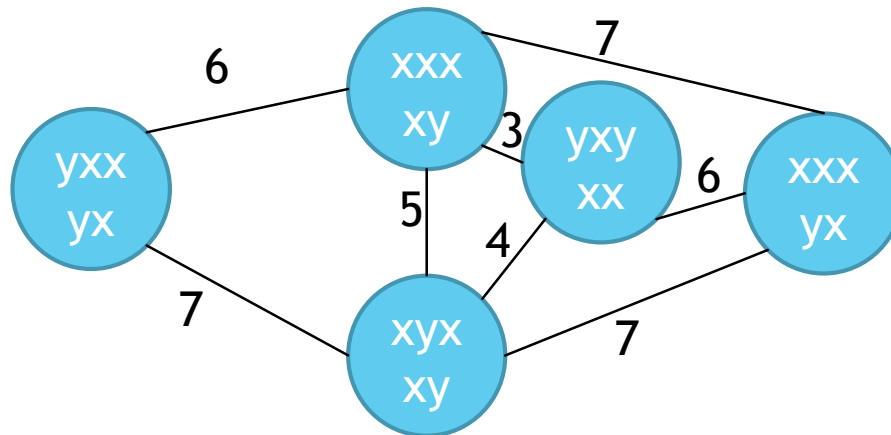
- ▶ Consider the agent's state (or the world's state!) as a set of binary values, eg $[0,1,0,0,1]$.
- ▶ Consider a target state $[1,1,1,1,0]$



- ▶ We can imagine a representation of all possible actions our agent can take to form a graph.
- ▶ These actions can be connected, or have conditions required to be met before they are selected

Combining State Machines and Path-Finding

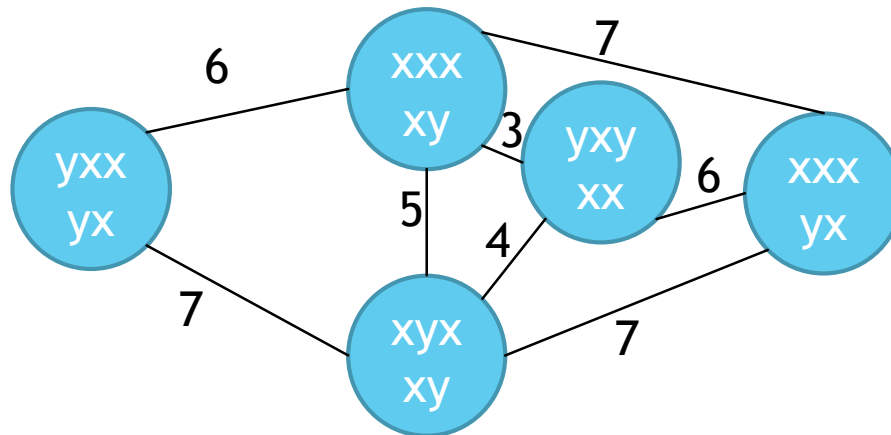
- ▶ Envision a scenario where these actions flip bits in our agent state.



- ▶ It follows that through searching the graph we can determine a minimum cost path to obtain our goal state
- ▶ This is a simplified view of the approach employed under Goal-Oriented Action Planning

Combining State Machines and Path-Finding

- ▶ Employed (and created) by industry
- ▶ Edges are contextual in GOAP-inspired systems - heuristic might not be suitable - Vanilla Dijkstra



- ▶ Often they will be defined/created during the search, as a function of what actions lead to preconditions being met
- ▶ This leads to potential edges between all vertices

Summary

- ▶ Developing code is an engineering job
- ▶ A good engineer chooses the right tool for each task.
- ▶ A good engineer uses scientific and mathematical ability to resolve actual issues.
- ▶ Occasionally, those good solutions can feed back into the science!