COMP250: Artificial Intelligence

4: Utility-Based AI

Utility

Utility theory

- How does an AI agent measure “goodness” or “usefulness” of the outcomes of its actions?
- **Utility theory** supposes that a given outcome can be assigned a **single number** measuring its **utility**
- Actions can then be **ranked** by utility, and one with the **highest** utility chosen
- Utility is sometimes called **reward**, **payoff**, **fitness**
- Multiply by -1 and we have **cost**

Utility — example

- A laptop costs £500 from Amazon or £450 from Bob's Computers
- Assuming everything else equal, where do you buy it from?

Utility — example

- A laptop costs £500 from Amazon or £450 from Bob's Computers
- Amazon offers next-day delivery, but delivery from Bob's Computers takes 4 weeks
- Assuming everything else equal, where do you buy it from?
- Utility is a **single number** — we essentially have to put a monetary value on the longer wait time

Weighting

- Utility is often formed of several **decision factors**
- In the laptop buying example: price and delivery time
- To get a utility value we can take a **weighted sum**

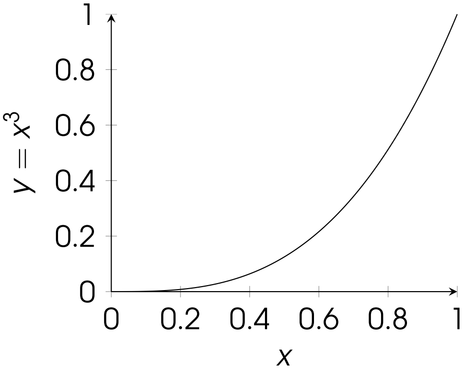
```
utility = WEIGHT_PRICE * price + WEIGHT_TIME * time;
```

- Here `WEIGHT_PRICE` and `WEIGHT_TIME` are constant values
- The values used will influence the agent's behaviour and so must be carefully tuned by the designer

Nonlinearity

- Decision factors are not always linear
- E.g. a difference of £1 is more significant for something that costs £5 than something that costs £500
- E.g. a difference of 1 HP is more significant if the agent is close to death than if it is at full health
- Therefore we may want to apply a **curve** mapping to decision factors

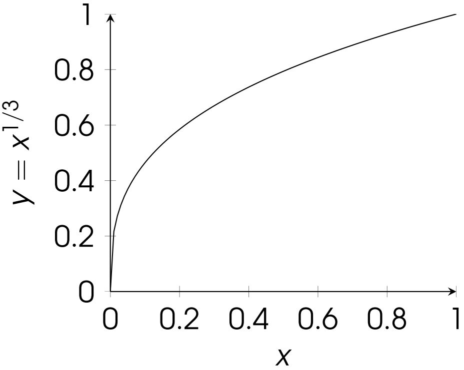
Polynomial curve



- Formula: $y = x^k$
- C#:

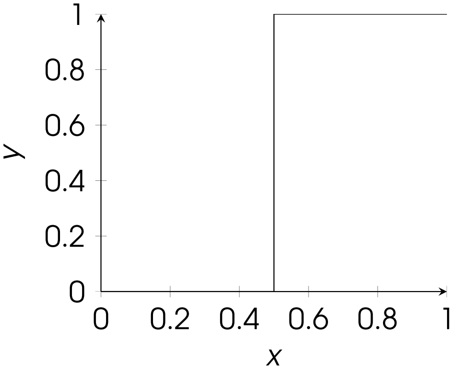
  `Mathf.Pow(x, k)`
- k is a constant: bigger k gives a steeper curve

Inverse polynomial curve



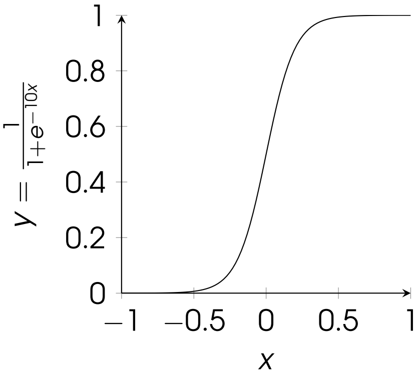
- Same formula as polynomial curve, but k is between 0 and 1
- k closer to 0 gives a steeper curve

Step function



- Formula:
  $$y = \begin{cases} 0 & \text{if } x < 0.5 \\ 1 & \text{if } x \geq 0.5 \end{cases}$$
- C#:
  ```
  (x < 0.5f) ? 0 : 1
  ```
- Models a **threshold** or **if-then** rule

Logistic function



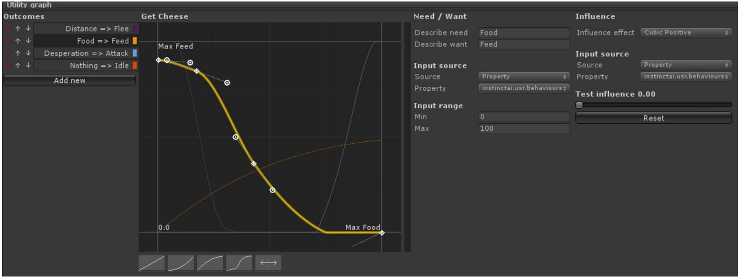
- Formula: $y = \frac{1}{1+e^{-kx}}$
- C#:

```
1 / (1 + Mathf.Exp(-k*x))
```

- Similar to step function but with a softer transition from "off" to "on"
- k is a constant: bigger k gives a steeper curve

Tweaking curves

- Adjusting utility curves is **more art than science**
- It's worth **experimenting** with different curve types to get the desired result
- **Graphical curve editors** are worth investigating
- E.g. InstinctAI asset for Unity:



Expected utility

- If the environment is **stochastic**, the outcome of an action (and hence its utility) may not be known with certainty
- Let $p(x)$ be the probability that a given action has utility x
- Then the **expected utility** is

$$\sum_x x \cdot p(x)$$

- That is, the sum of utility values weighted by their probabilities

Expected utility — example

- A slot machine pays out:
  - £1 with probability 0.05
  - £5 with probability 0.03
  - £10 with probability 0.02
  - Nothing with probability 0.9
- The expected payout is

$$1 \times 0.05 + 5 \times 0.03 + 10 \times 0.02 + 0 \times 0.9 = 0.4$$

  i.e. £0.40
- If it costs £1 to play the slot machine, the expected utility overall is $-$£0.60
- (Although the actual utility can range from $-$£1 to $+$£9)

Inertia

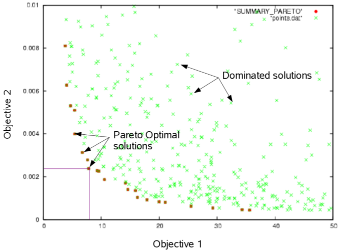
- Utilities will generally change rapidly as the state of the environment changes
- This could result in the agent **changing its mind** as to which action has the best utility
- Can force the agent to finish its current action before evaluating and choosing another
- Can give a utility bonus to sticking to the current action — this still allows the agent to change its mind if the current action's utility becomes very bad

Multi-objective optimisation

- Utility-based AI is **single-objective**: all decision factors must be combined into a single number
- An alternative is **multi-objective**: treat all decision factors as separate, and find an action that optimises all of them at once

Pareto optimality

- Consider the space of all possible solutions (e.g. actions or plans)
- A solution is **Pareto dominated** if there is some other solution that is better than it on **all** decision factors at once
- A solution is **Pareto optimal** if it is not dominated by any other solution



Single-objective vs multi-objective

- Multi-objective optimisation gets around the problem of having to tune weights for decision factors
- However, there are generally a large number of Pareto optimal solutions so we need some other method to tie-break between them

GOAP

GOAP

- **G**oal **O**riented **A**ction **P**lanning
- Originally developed for F.E.A.R. (2005), since used in several games
- A modified version of STRIPS planning (recall from last week) specifically for real-time planning in video games

GOAP

- Each agent has a **goal set**
  - Multiple goals with differing **priority**
  - Goals are like in STRIPS — sets of predicates that the agent wants to satisfy
- Each agent also has a set of **actions**
  - Like in STRIPS — actions have preconditions and postconditions
  - Unlike STRIPS, each action also has a **cost**

Action sets

- Different types of agent could have the **same goals** but **different action sets**
- This will result in those agents achieving those goals in **different ways**
- NB this doesn't have to be explicitly coded — it **emerges** from the GOAP system
- E.g. this was used by the F.E.A.R. team to quickly add new enemy types

Action sets

Soldier

	Action
1	AI/Actions/Attack
2	AI/Actions/AttackCrouch
3	AI/Actions/SuppressionFire
4	AI/Actions/SuppressionFireFromCover
5	AI/Actions/FlushOutWithGrenade
6	AI/Actions/AttackFromCover
7	AI/Actions/BlindFireFromCover
8	AI/Actions/AttackGrenadeFromCover
9	AI/Actions/AttackFromView
10	AI/Actions/DrawWeapon
11	AI/Actions/HolsterWeapon
12	AI/Actions/ReloadCrouch
13	AI/Actions/ReloadCovered
14	AI/Actions/InspectDisturbance
15	AI/Actions/LookAtDisturbance
16	AI/Actions/SurveyArea
17	AI/Actions/DodgeRoll
18	AI/Actions/DodgeShuffle
19	AI/Actions/DodgeCovered
20	AI/Actions/Uncover
21	AI/Actions/AttackMelee

Assassin

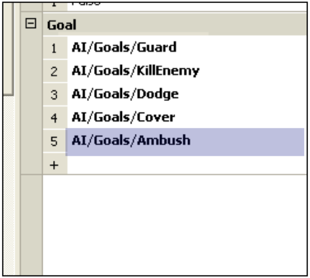
	Action
1	AI/Actions/Attack
2	AI/Actions/InspectDisturbance
3	AI/Actions/LookAtDisturbance
4	AI/Actions/SurveyArea
5	AI/Actions/AttackMeleeUncloaked
6	AI/Actions/TraverseBlockedDoor
7	AI/Actions/UseSmartObjectNodeMounted
8	AI/Actions/MountNodeUncloaked
9	AI/Actions/DismountNodeUncloaked
10	AI/Actions/TraverseLinkUncloaked
11	AI/Actions/AttackFromAmbush
12	AI/Actions/DodgeRollParanoid
13	AI/Actions/AttackLungeUncloaked
14	AI/Actions/LopeToTargetUncloaked
+	

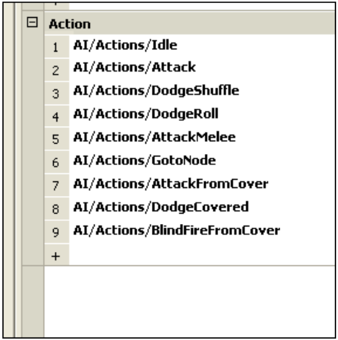
Rat

	Action
1	AI/Actions/Animate
2	AI/Actions/Idle
3	AI/Actions/GotoNode
4	AI/Actions/UseSmartObjectNode
+	

Layering

- Goal set allows different behaviours with different priorities to be **layered**
- E.g. enemy AI in F.E.A.R.:





Implementing GOAP

- An **abstracted** view of the game world is used for planning
- Represented as a fixed-length array (or struct) of values
- Predicates (preconditions, postconditions, goals) represented in terms of this array representation
- Most implementations also allow for **programmatic** preconditions (e.g. calling the pathfinding system to check availability of a path)

Implementing GOAP

- Not difficult to implement
- Open-source implementations do exist
- Not built into Unity or Unreal, but asset store packages are available

Finding the plan

- As in STRIPS, we can build a **tree** whose nodes are world states and edges are available actions
- Since actions have costs, we can use A* to find the lowest cost path to the goal
- Plan is a **queue** of actions that the agent then executes
- If the plan is interrupted or fails then the agent can **replan**

GOAP vs behaviour trees

- BT: Designer specifies “how”
- GOAP: Designer specifies “what” — “how” is in whatever system is used to implement actions (FSMs in F.E.A.R.; could use BTs or hand coding)
- Both: actions (tasks in BT) are modular and reusable between agents
- GOAP: goals are also modular and reusable
- BT: goals are not represented explicitly
- BT can be classified as **authored behaviour**
- GOAP can be classified as **computational intelligence**