Uncertainty and Utilities

Table of Contents

[Worst VS Average Case 2](#_Toc138878364)

[Expectimax Search 3](#_Toc138878365)

[Performance Improvements 4](#_Toc138878366)

[Pruning 4](#_Toc138878367)

[Depth Limited Search 5](#_Toc138878368)

[Probability Values 5](#_Toc138878369)

[Informed Probabilities 6](#_Toc138878370)

[Modelling Assumptions 6](#_Toc138878371)

[Other Types of Games 7](#_Toc138878372)

[Mixed Layer Types 7](#_Toc138878373)

[Multi-Agent Utilities 8](#_Toc138878374)

[Utilities 8](#_Toc138878375)

[What Utilities to Use 9](#_Toc138878376)

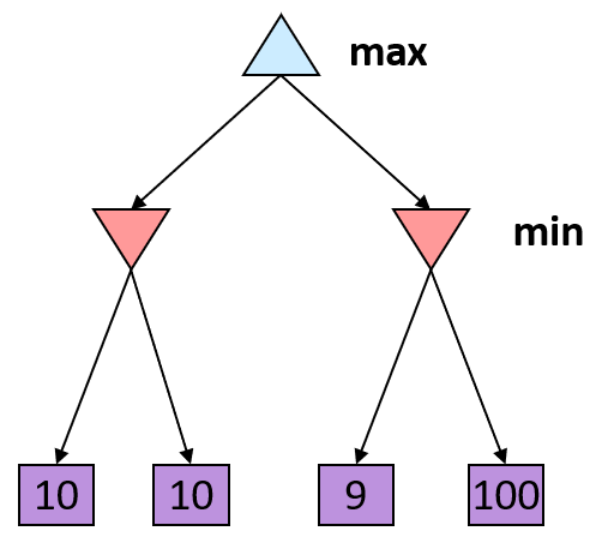
[Creating Utilities 9](#_Toc138878377)

[Preferences 10](#_Toc138878378)

[Rational Preferences 10](#_Toc138878379)

[Human Utilities 11](#_Toc138878380)

## Worst VS Average Case



Consider the minimax tree above. We know through minimax search that the outcome from this tree should be 10. If we try to reach the 100 mark, then our opponent will take us to the 9 mark. However, this is the **worst-case result**. What if our opponent is not playing optimally? What if uncertainty about the outcome becomes involved?

Suppose there is a 50% chance of our opponent going either way. Thus, for the left sub-tree, the expected value is 10 and for the right sub-tree the expected value is 54.5. In this scenario, we should choose the right sub-tree, even though minimax search tells us not to do that, purely because we have a **higher expected value**. The uncertain part of the equation is controlled by chance, not by our opponent. We are able to take this opportunity because we know that the adversary is not optimal.

## Expectimax Search

There are a few scenarios where we might not know for sure what the outcome of our actions will be.

* **Explicit Randomness** – These are situations where the game explicitly introduces randomness, such as a dice being rolled.
* **Unpredictable Opponents** – Sometimes, opponents might not play optimally.
* **Failed Actions** – In certain situations (not likely in board games), there is a possibility of an action failing. For example, a moving robot could slip on a certain surface.

In situations like this, using minimax search will give us worse results since we are being more pessimistic than we need to be. Instead, we should use **expectimax search**, which will give us the average results.



In expectimax search, instead of having a minimizer, we have a **chance node**. At these nodes, we calculate the **expected value** instead of the minimum one. Note that the actual value during gameplay will be different from this expected value, but over a large number of plays, the average value will be the expected value.

The pseudocode for expectimax search is provided below:

def value(state):

if the state if a terminal state: return the state’s utility

if the next agent is MAX: return max-value(state)

if the next agent is EXP: return exp-value(state)

def max-value(state):

initialize

for each successor of state:

return

def exp-value(state):

initialize

for each successor of state:

return

PSEUDOCODE

## Performance Improvements

### Pruning

Unlike in minimax search, we cannot perform **pruning** in expectimax seach. In pruning, we used the fact that the minimum value will be picked by our opponent. If we found that the minimum value down a sub-branch after partial exploration was already lower than the minimum value down a different sub-branch, we knew beforehand that we should not use that sub-branch. We cannot make this same prediction for expectimax search, since we are no longer dealing with minimum values, but rather with average ones. We cannot know if the average value down a sub-branch will be higher or lower than our current maximum value without fully exploring the sub-branch.

### Depth Limited Search

**Depth-Limited Search**, however, is a perfectly valid technique to use with expectimax search. Just like minimax search, we will have a prediction function which will predict the **expected outcome** from a sub-branch rather than the minimum outcome.

One interesting observation here is, due to the huge amount of uncertainty involved in a large search tree, making an incorrect move at any single point makes much less of a difference to our overall results. Thus, we can make one or two mistakes, which also reduces the pressure on the evaluation function to be perfect.

## Probability Values

In expectimax search, we are using a **probabilistic model**. The probabilities could follow a simple uniform distribution (such as a dice roll) or something that is far more sophisticated and requires a great deal of computation. For the moment, we will assume that the probability values will be provided for us.

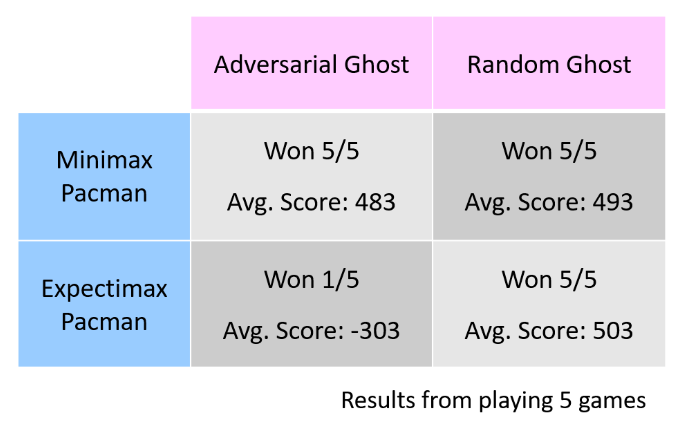
## Informed Probabilities

Suppose we know that our opponent is running a depth-2 minimax, using the results 80% of the time and moving randomly otherwise. In this scenario, we should use expectimax search since chance is involved.

However, to find the value of each chance node, we need to run a simulation of the opponent. This can quickly make things very difficult as we go deeper into the tree, since the number of branches will increase quickly. It gets even worse if we have an opponent that is simulating us as well, since we need to simulate them simulating us. Unlike in minimax search, where we know what the opponent will do, the existence of chance makes life difficult for us in expectimax search. Because of this, most of the time, we use a depth-limited expectimax search.

## Modelling Assumptions

When talking about chance and expected outcomes, we are making a few assumptions about the behaviour of our adversaries. What if we are optimistic and assume that the opponent makes moves based on chance when in reality, they are an optimal opponent? What if we assume that they are optimal but in reality, they make moves randomly? In each of these scenarios, we end up with suboptimal results.



Thus, it is important for us to make sure our assumptions are accurate reflections of reality to the best of our abilities.

## Other Types of Games

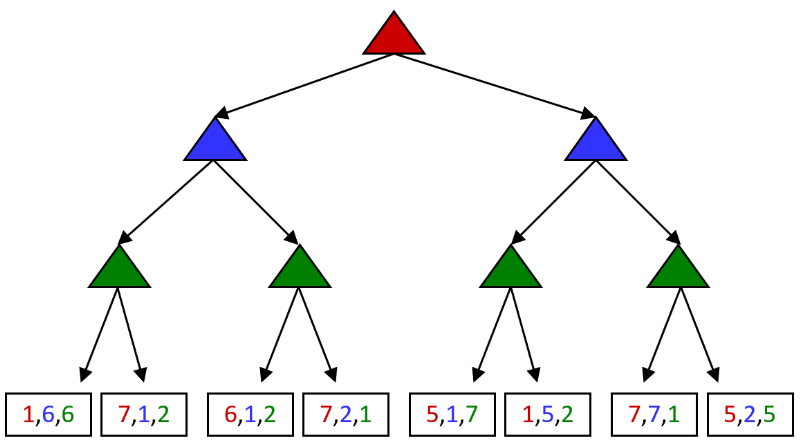
### Mixed Layer Types

For **mixed layer games**, the environment itself acts as an extra player which behaves randomly. Thus, we have our player which uses a maximizer, the adversary which uses a minimizer and a random agent, such as a dice roll, which uses a chance node. The search in this scenario is called an **expectiminimax**.

Games with more layers have a higher branching factor, adding to the computational complexity.

### Multi-Agent Utilities

We can also have non-zero-sum games or games that have multiple players. In this case, the terminal nodes are **tuples** of utility values with each player trying to maximize their own utility. This can give rise to cooperation, competition, etc.



## Utilities

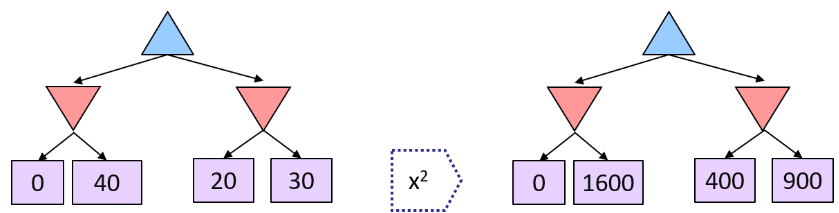
Think for a moment about why we are taking the expected value in cases where chance is involved and not just using minimax. If we have a possible outcome that has a very high value but a low probability, we will end up choosing that if we use minimax, even though it is very unlikely that we will ever get that outcome. Remember that our agent should be **maximizing the expected utility**.

When discussing utilities, we need to answer a few questions:

1. Where do the utilities come from?
2. How do we know that some utility even exists?
3. Why does averaging the utilities make sense?
4. What if the desired behaviour cannot be expressed as a utility?

### What Utilities to Use

For minimax search, all that matters is the order of the utility values, meaning that better states should have higher values. For the two situations below, we will end up with the same set of actions to take.



This property is called **insensitivity to monotonic transformations**.

For expectimax search however, this property does not hold and we will end up with a different set of actions. The **magnitudes** have meaning in this case.

### Creating Utilities

Utilities are functions that map some real state of the world to a number representing the agent’s preferences. The games we have looked at so far have been ones where we can simply assign a +1 or a -1 to certain outcomes to denote desirable and undesirable outcomes.

Not every situation is this intuitive though. For example, consider that we have the choice between getting 1 or 2 scoops of ice cream. Obviously, the choice with the higher number of scoops has a higher utility. However, that choice also has a chance of us dropping the ice cream leaving us with nothing. Thus, it is no longer sufficient to just say 2 scoops of ice cream is better than 1 scoop. We need to assign a value to each choice to denote exactly how much better it is so that we can calculated expected values.

### Preferences

The first step is to define our **preferences**, which are the different outcomes, or prizes, that we can get. On top of this, we need to consider situations where we have **lotteries**, where the prize is uncertain. These are denoted as , meaning we will get with the probability and we will get with the probability .

In the discussion ahead, we use to denote that is preferred over and we use to denote that it does not matter, we are indifferent.

### Rational Preferences

To avoid situations where our chosen preferences make it impossible to create a rational agent, we should follow some constraints.

* – Axiom of Transitivity
* – Axiom of Orderability
* – Axiom of Continuity
* – Axiom of Substitutability
* – Axiom of Monotonicity

If we have a set of preferences that abide by these axioms for all prizes, then the preferences are rational. This is turn implies behaviour that can be described as maximizing some expected utility, i.e., there exists some function such that

meaning values assigned by preserve the preferences of both prizes and lotteries.

This equation comes from the **Maximum Expected Utility** (MEU) principle, which simply states that we should choose the action that maximizes the expected utility.

Note that it is entirely possible to build a rational agent without ever representing or manipulating utilities and probabilities, for example by using a lookup table that tells us the correct move to make in each situation for simple games like Tic-Tac-Toe.

## Human Utilities

If we rescale the utility values to be between 0 and 1, the values become manageable without having any effect on the outcome. We do this by assigning the worst possible outcome to the value 0 and the best possible outcome to the value 1 and rescaling everything in between accordingly. Utility values are said to be **invariant under positive linear transformations**.

With deterministic prizes only (i.e., no lottery choices), the only thing that matters is the order. The magnitude does not matter.

One practical use case for the world we live in is trying to decide how much we are willing to pay to reduce the risk of a product literally killing someone. Suppose there is a one-millionth chance of someone dying. This is called the **micromort** value.

Another way this can also be described is using **Quality-Adjusted Life Years** (QALYs), which assigns a value based on how long you live along with the quality of your life. Thus, death at birth would be 0, a long happy life would be 1, dying in a car accident at age 30 would maybe be 0.4 and so on. This can be used in Trolley Problem situations, for example to decide whether we want to cure cancer for 3 people or cure the blindness of 5 people.

Utilities map states to real numbers, so we need to figure out how to calculate these numbers. The standard approach to assessing human utilities is to compare a prize to a standard lotter with has a best possible outcome with the value and a worst possible outcome with the value .

Suppose the situation is such that we need to spin a wheel with a nominal chance of us getting killed. We can avoid spinning the wheel altogether by paying some amount of money. How much are we willing to pay? Thus, we need to adjust the probability of death until a point where .

A similar setup can also be created for money, even though money itself does not behave like a utility function (since rich people literally do not care about money), having money or being in debt does. Given a lotter , the **expected monetary value** is given by while the utility is given by . Typically, , meaning that the utility of having a possibility of winning some money is usually lower than the utility of actually getting some money, even if the latter amount is lower. The (lower) amount of money that people are willing to accept for certain is called the **certainty equivalent**. This is because people are risk-averse. However, this behaviour changes when people are in debt, where they become more risk-prone. The difference between the expected monetary value and the certainty equivalent is the **insurance premium**. The entire insurance industry exists because people are willing to pay to reduce the risk of having to paying more in case of an accident.