Suppose you want to beat your friend, a world champion, at checkers. You're not very good yourself, but you happen to have a trick up your sleeve: a super powerful artificial intelligence (AI). You can just ask your AI for advice on which moves to make.

You're a bit worried, though. You have a complicated relationship with your AI, you don't trust its recommendations entirely. It has its own agenda, which sometimes might involve offering you bad advice. So, you'd like some way to be sure that the moves it recommends are actually good. At first thought, it's not clear that this can be done. If the AI proposes a move, how do you know there isn't a tricky response to it which will let your friend beat you? Maybe to convince you that a move is good, the AI would have to present you with a way of winning against every possible response by your opponent. You don't have time for that! Your friend would tire of waiting for your move and win by default.

Amazingly though, there is a way for the AI to quickly convince you that a given move is a good one, and for you to catch it if it's trying to lead you astray. That is, when your AI gives you a recommendation, after a quick back-and-forth with it, you can be 99.99999% certain that it really is giving you a winning move and not trying to trip you up.

To understand a little of how this is possible, let's take a brief trip through complexity theory.

Algorithms and PSPACE

An *algorithm* is a set of instructions which, when followed, solve a particular problem. Let's try to come up with an algorithm to check if a given move is a *winning move* for a game of checkers.

First let's define what we mean by a winning move.

Definition of a winning move (1):
A move is a winning move for you if
no matter what move your opponent makes in response,
the response is not a winning move for them.

This definition is circular: we've defined a winning move for you in terms of winning moves for your opponent. We can get a feel for its usefulness by expanding the self-reference one more level – that is, by substituting in the definition itself when we refer to it in the body of the definition.

Definition of a winning move (2):
A move is a winning move for you if
no matter what move your opponent makes in response,
there is a response to their response you can make
that is a winning move for you.

Let's write this definition a little more succinctly as

Definition of a winning move (2a):
A move is a winning move for you if
for every response
there is a response to the response
that is a winning move for you.

Let's expand this out one more level:

Definition of a winning move (3a):
A move is a winning move for you if
for every response
there is a response to the response so that
for every response to the response to the response
it is not a winning move for your opponent.

We'll assume that we're talking about checkers without kings (the rule that your pieces gain the ability to move backward once they make it to the other end of the board), so pieces can only move forward, and can no longer move once they've hit the end. Since both players start out with 12 pieces and each piece can only move at most 7 spaces forward, the total number of moves each player can make before all their pieces are stuck is at most $12 \cdot 7 = 84$. That means that if we expand out our definition of a winning move $2 \cdot 84 = 168$ times, it will "bottom out":

Definition of a winning move (168a):
A move is a winning move for you if
for every response
there is a response to the response so that
. . . (168 times)
for every $\underbrace{\text{response to the . . . to the response}}_{\text{167 times}}$
there is a $\underbrace{\text{response to the response to the ... to the response}}_{\text{168 times}}$
so that if both players make this particular chain of moves, you win.

This description of what a winning move also provides us with an algorithm for checking if a given move *is* a winning move. Namely, for every possible response, check that there is some response to the response, so that for every possible response . . . , the chain of moves results in you winning.

This algorithm has us examine every possible sequence of moves and responses. There are about 12^{168} such sequences, which is unimaginably huge: it's far bigger than the number of atoms in the universe. So this isn't exactly a practical algorithm.

Even though this algorithm takes a long time, you can arrange things so that the amount of work space – think of it like the scratch

paper used in doing calculations – which it uses is pretty small, on the order of one kilobyte. That is about the size of a 30 pixel by 30 pixel image.

PSPACE and interactive proofs

Problems like this, those which can be solved using a small amount of space, are called PSPACE problems. What does all this have to do with using our untrustworthy AI to help us win at checkers? Well, there's a surprising theorem which states that any PSPACE problem has a short *interactive proof*. That is, for any PSPACE problem, like finding winning moves in checkers, given a advisor proposing a solution, there's a short conversation you could have with them that would convince you of the correctness of their solution. Moreover, if the advisor was trying to trick you by giving you the wrong solution, you'd be able to catch them with overwhelmingly high probability.

This is an amazing fact! It says that even for very hard problems like picking winning moves in games, which you have no hope of solving on your own, you can safely accept advice from untrusted sources. Your AI, who might be trying to make you lose your checkers match, will be easily stymied by your ability to check its suggestions.