

What makes a move good?

There has been a lot of discussion about this, and it mostly goes nowhere concrete.

This is because the precise notion of a good move, or 'best play' is rather technical. This is unlike chess, checkers, tic tac toe, or any other traditional board game, where most adults at least intuitively understand what makes a move good or bad. This distinction is because of the simultaneous nature of Pokémon

Even in the simplest game states, the outcome of any move that you select is dependent on what move your opponent selects, which is unknown to you. This scenario in its most distilled form is known as a 50/50

Readers might also notice that Pokémon has other aspects which complicate analysis, the salient one being information. The species, stats, items, move-sets, and abilities of your opponent's Pokémon are unknown to us, and the most obvious problem this presents is that we are unable to 'look-ahead' with any certainty; even if we account for

Even if we had the time and computing resources to keep track of our opponent's options and the subsequent outcomes, we don't know their options (moves) to begin with! And if we don't know their stats we can't calculate the possible damage rolls. And if we don't know their ability we can't account for any possible side effects. The list goes on...

There is an art to deducing these unknowns based on information that is revealed over the course of a game. However, this 'imperfect information' aspect of Pokémon is very tricky to deal with programmatically, and it also requires that we have an even looser and more technical notion of best play than I am trying to introduce currently.

Therefore, I'm going to make a necessary decision, that in all the discussion that follows, we have perfect information.

This definition makes the following discussion much easier and the provable results much stronger. It is also not entirely unrealistic; Towards an endgame we may have figured out our opponent's team with a good deal of certainty

There is also uncertainty in rng. Fortunately, as we will see this is easily dealt with, and only poses a minor computational load

I guess before we should proceed, I should make it entirely clear what my motivations for this are

1. Define what a good move is, mathematically. Make it clear why this definition is valid. Use this definition and the results of game theory to make interesting and provable claims about this game (in the perfect information context)

2. Use this framework to program an engine. Modern chess engines have long surpassed human players and are used for analysis and for solving chess positions. I would like to create a perfect analogue of this for Pokémon. There have been many attempts to create mon playing agents, with very limited success. However, as far as I know none of these programs have been designed with the proper mathematical foundations are not provably guaranteed to find the optimal strategy in a given position

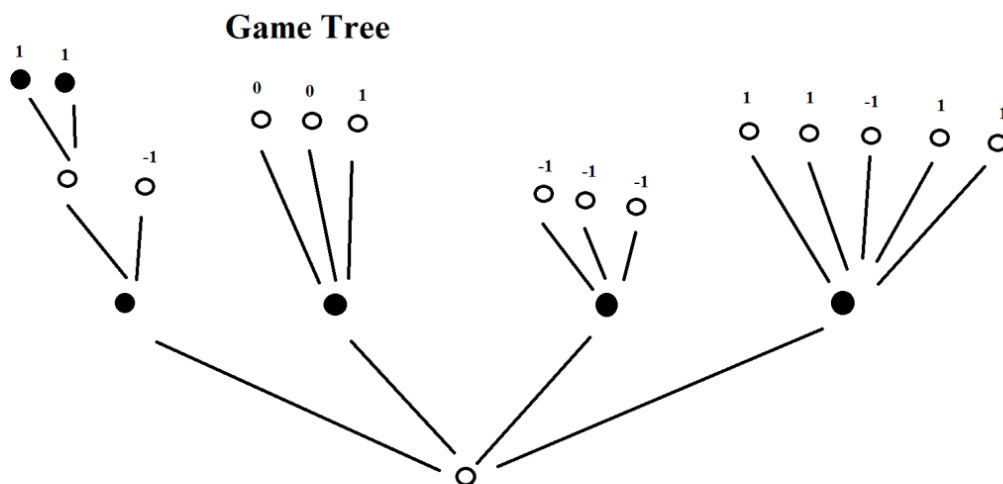
So, before we proceed talking about Pokémon in a theoretical perspective, we should talk about a traditional turn-based game

This tree is an abstract representation of a game. The dots, called vertices or nodes, represent a board state

The lines are called edges, and they represent moves or actions. They connect game states together just as playing a move causes the game state to progress

In this diagram the tree grows upwards as the game develops

The nodes are color coded to indicate who is to move, so at the start of the game it is white to move, with black to follow and vice versa.



The topmost nodes are called leaf nodes, or terminal states when thinking about game states. The number represents a score or reward that is given to the white player at the end of the game

All games are zero sum

This is not necessarily true for every game, but mons and every other game mentioned thus far are in this category. This definition is essential to attain some of the later results

Whatever reward that white gets, black will get the opposite reward. This ensures that both players are competing against each other while trying to maximize their own reward. It rules out cooperation; neither player has anything to gain buy maximizing the reward of their opponent

By convention, a win is 1, a loss is -1, and a draw is 0

Now if we closely consider the picture, we see that if both players are playing optimally, the white player should select the second option which will result in a draw. Any other move will result in a loss

Now if black is not playing optimal, it might be worthwhile to select the fourth option. Black has a way to force a win there, but they have many options so they are 'likely' to select a move that loses

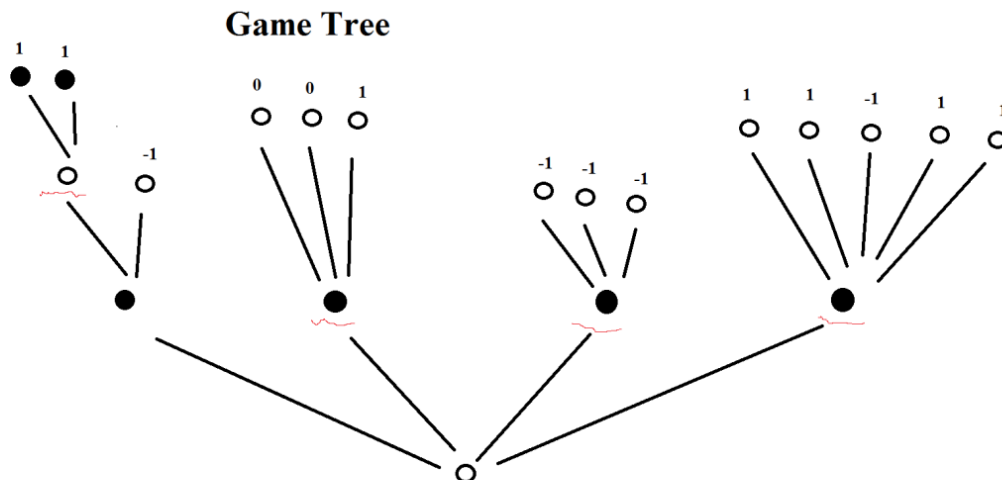
It is important to note that considerations like this are outside the scope of this discussion. We are concerned with the result of the game when both players optimize their rewards without mistake

Before we move on from game trees and address SM games, we will describe a simple algorithm for determining the best play, or expected reward of a game just as we did above

Simple is this case has a particular meaning. It is an algorithm so straightforward that a computer could easily be programmed to solve game trees of any size that could be stored on memory. The algorithm in question is called minimax, and it is the original solution concept defined by Von Neumann in the infancy of game theory

Each node of the tree can be thought of as the starting point, or root node, of a subgame. The minimax algorithm works by solving the entire tree recursively, working backwards from the leaf nodes all the way to the root. If a node is a leaf, it is already solved, and its expected reward is just the score. If a node points to only nodes that are already solved, then we employ a very simple heuristic:


If the current node is white to move, then best play would entail selecting the next node with the highest score; we are selecting the child node with the maximum score. If we are at a node that is black to move, the we should select the move with minimum score. In either case the current node now has the same value as the child node we selected.




To summarize, minimax is an algorithm to determine what move is best AND a precise mathematical notion of value of a game. In the example given, the game is a draw.

Now let's describe the simplest possible Pokémon game state that will challenge our attempts to describe optimal play

Imagine the white player has just a fast Heatran with lava plume vs a +2 Bisharp with sucker punch and knock off. Since both players are selecting their move at the same time, this game is best described with a table



				
			<b>Sucker</b>	<b>Knock</b>
<b>Lava Plume</b>		-1		1
<b>Rocks</b>		1		-1

**\*sucker only has 1 pp left somehow**

In the mons example there is no best move, in the sense that we had before. We could announce to our opponent that we, as the white player in the prior tree game, that we were going to select the second move, and our opponent would not be able to exploit this declared intention to squeeze out a reward better than the minimax value of 0. But now, no matter what move we select, our opponent could select a countering move in order to win the game! Colloquially this is known as a 50/50, and it may seem to you, as it did to me at one point, that even this simple example game is incompatible with any notion of best play

But there is something quite remarkable about the term 50/50. It suggests that players tacitly understand, on some level, that there is a probabilistic solution concept in this setting

The key idea is that of a mixed strategy. Instead of there existing a clear best move that should be selected every time, each move has a probability associated with it. Players make their selection by choosing randomly with respect to this probability distribution function on their available actions

Since we are now selecting moves randomly, we have to refer to the expected reward

We can call each square in our grid/matrix an event. Then the expected reward is the average reward of all events weighted by the probability of that even happening with respect to both players mixed strategies

Let's say both players adopt the mixed strategy of just 'flipping a coin'. They both will select either of their actions with equal probability. We can then calculate the expected reward for white with respect to this pair of mixed strategies

		<b>.5</b>	<b>.5</b>
		<b>Sucker</b>	<b>Knock</b>
<b>Lava Plume</b>		-1	1
<b>.5</b>		<b>.25</b>	<b>.25</b>
<b>Rocks</b>		1	-1
<b>.5</b>		<b>.25</b>	<b>.25</b>

Expected reward = 0

Now let's consider what would happen if black altered their mixed strategy while white keeps the same.

If we pick any other strategy, say sucker -> .3, knock -> .7 and carry out the calculation, we arrive at the same expected value

This same thing happens if black fixes their strategy and white tries something else. (Obviously by symmetry of the game)

So, these two mixed strategies, taken together, have a certain stability to them. Neither player improves their outcome by deviating from their mixed strategy. This is called nash equilibrium, and it is the primary solution concept for SM games like this

There are many comprehensive resources that explain just NE better than I can. But there are a few relevant points that I would like to touch on before expanding the scope of this discussion to mons as it is actually played (i.e. not this artificial one turn mini game)

Firstly, a Nash equilibrium does not require communication or cooperation to be valid. By definition, it consists of two specific strategies  $s_1, s_2$  together; It may seem that this arrangement falls apart if both sides do not 'cooperate'. This however is not the case

If white strolls up to the game and decides with just himself that he will play his half ( $s_1$ ) of the Nash equilibrium, he can be assured that no matter what black does, he is guaranteed an expected outcome no worse than the value that's associated to the pair of strategies like we calculated earlier.

His opponent can play however he would like, in a manner totally unknown to him, and white is still assured to break even like we calculated]

Now, if black decides that he is going to play the strategy ( $.7 = \text{sucker}, .3 = \text{knock}$ ), then white would miss out in some sense by playing the 50/50 strategy. You can check for yourself, but if white somehow knew that black was foolhardy like this then white could play his own strategy of ( $.3 = \text{plume}, .7 = \text{rocks}$ ) and he would win more often than not. With this pair of strategies white would have a positive expected reward.

However, this is not the point of NE. If you play your part of a Nash equilibrium strategy you are not guaranteed to get the best possible outcome regardless of what your opponent does. That is impossible; there is no such notion in the world of simultaneous games.

What it will do however is protect you from being exploited

It is very important that we draw a comparison to the tree game example. If black was foolish and was somehow certain to mess up the 'endgame' after white selects the 4th move, then white suffered an 'opportunity cost' by selecting the minimax 2nd option

But selecting the fourth option leaves you open to being exploited by a keen opponent and losing the game you could have drawn

I am hopeful that I've drawn some connection between the 'best play' aspect of the minimax algorithm and the '(un)exploitability' aspect of Nash Equilibrium

Now I'd like to take a step back for a sec and spell something out

Even though this simple matrix game is without any intrinsic 'variance' or 'rng', the solution to the game, the Nash Equilibrium strategy pair, is inherently probabilistic

playing this correctly means selecting moves randomly (but still in accordance to some strategy, I don't mean always uniformly random even though that happens to be the case in this example) and success of our play has to be measured on average, ideally with as many play-throughs/runs as possible

2.

There are some interesting theorems/facts about NE that we get to use. These apply to any matrix game regardless of how many options there are or what the rewards are

Most importantly, there always exists at least one NE!!!

Next, there may exist more than one NE. This is not a problem though. Just like in the previous discussion, both players are content with playing their half on any NE, and they are safe from being exploited. They are always assured their just reward [This is only true for 2 player, constant sum]

Lastly, in the case of zero-sum games, all NEs have the same expected reward (for both players)

These results should make you feel happy

If we are trying to discuss 'perfect play, etc', all this just means it actually exists and is just as valid as in more traditional games

But wait -

Everything I've said so far seems to only apply to these shitty one-shot matrix games which do not describe mons in full. So let's expand our definitions a bit to accommodate mons!

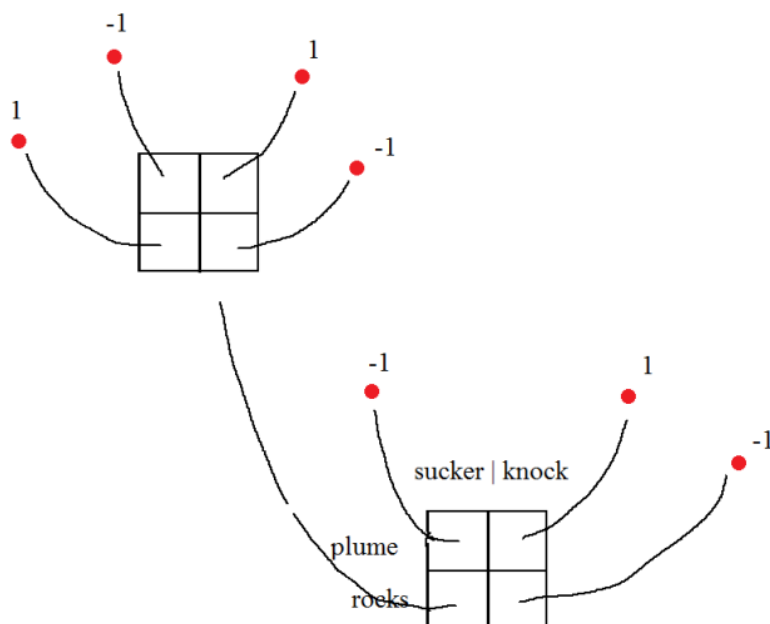
In mons, after both players have selected their actions, the game usually does not end, and rewards are given. Instead, the game progresses to the next state. This is where the tree makes its return

We should picture an example of this, but also as simple as possible

Let's use the sucker 50/50 example, but this time sucker has 2 pp ☹☹

All events are the same, except in the case of white getting the sucker turn right

if Heatran rocks and Bisharp suckers, black does not necessarily lose. Instead, the game progresses to the next turn and this new turn is identical to the 1 pp case



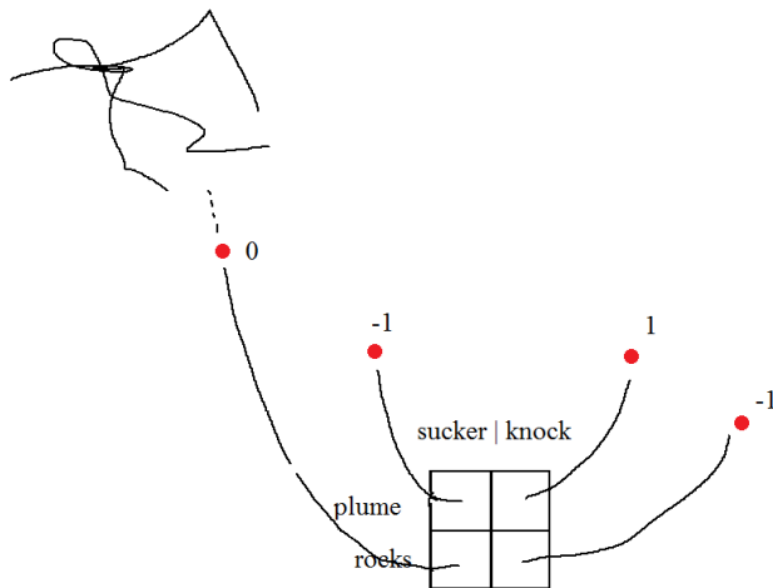
Now the question is what do we do on the first turn?

Well, before we'd play a NE strategy, and if all the entries in the matrix are numbers/rewards we can calculate this easily

[https://cgi.csc.liv.ac.uk/~rahul/bimatrix\\_solver/](https://cgi.csc.liv.ac.uk/~rahul/bimatrix_solver/)

however, at the root matrix one of the entries does not have a reward given to us a priori

BUT just like in the minimax algorithm we can solve this tree from leaves to root. The values will propagate down the tree and 'fill in the blank entries' so to speak



The second matrix further up the tree is solvable, with expected reward of 0

so, we can simply replace that branch with the value 0, and solve for the matrix game at the root

This matrix has entries -1, 1, 0, -1

Solving for Nash equilibrium in a general matrix game is actually somewhat involved. Fortunately, there exist many algorithms that can enumerate all the NE (though we only need to find one to calculate the value of that game)

This new game has an expected value of  $-1/3$  for white

with NE strategies of  $s_1 = (\text{plume} = 1/3, \text{rocks} = 2/3)$  and  $s_2 = (\text{sucker} = 2/3, \text{knock} = 1/3)$

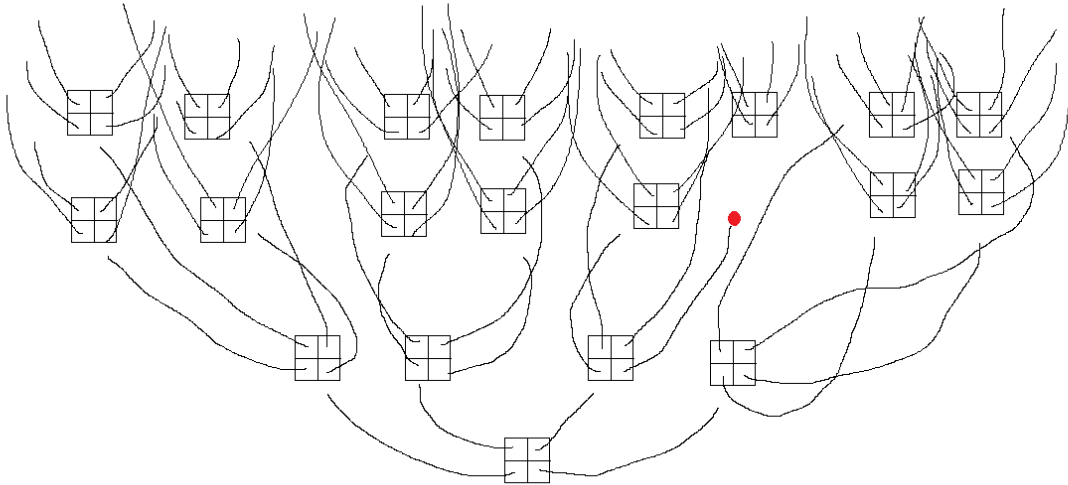
There is just one more relatively minor point before we have a complete picture of mons in the perfect information setting: RNG

The term 'variance' is sadly very common when describing misses, flinches, crits, etc because I feel that it not really disambiguated from the probabilities inherent in SM games



So how does rng fit into our literal picture and how does it affect our solution concept?

Not pictured: Even more nodes and edges



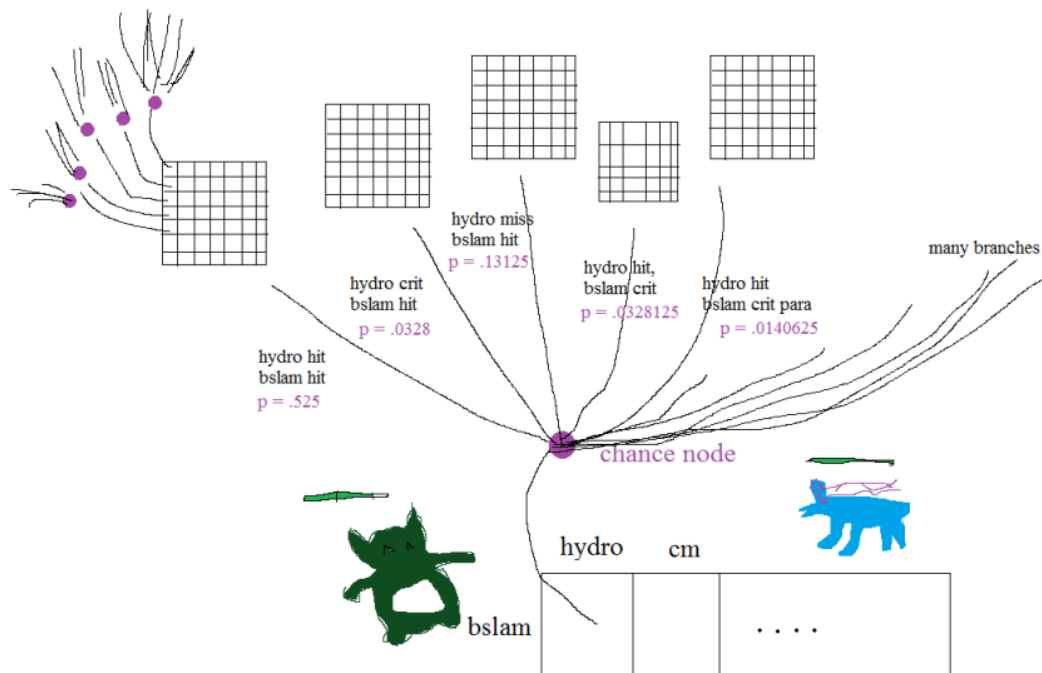
With rng in the picture, entries in a matrix no longer point directly to other matrices, like they do in the picture

Instead, they first point to a chance node. This chance node will have multiple edges pointing from it, and these edges are what point to the next matrix

each of these edges has an associated probability (so the probs. for all the edges from any chance node sums to 1)

and when we traverse up the game tree as the game is played out, one of these branches at the chance node is selected totally randomly. Neither player has any effect on this RNG process once they've selected their actions

Imagine a Body Slam Snorlax vs a Calm Mind Hydro Pump Suicune. Both players choose to attack each other. From this one single action pair, there are myriad possibilities that can arise from crits, misses, and paras alone



Fortunately, this is quite easy to accommodate into our solution algorithm

Each entry no longer has one matrix it points to, and thus no longer one value to inherit

Instead, the entry becomes the average value of all the matrix that its chance node points to, weighted by the probability of that event happening.

Essentially RNG only introduces more nodes and edges. It does not fundamentally change our game

And as long as we assume perfect information (i.e., both players revealed their types at the start of the battle), then the chance nodes, all the other nodes they point to, and the probability of these branches are ALL known to both players

After all, these nodes are just determined by the game mechanics

[I forgot to mention damage rolls, but these are not any different. It's only that there are 16 possible rolls for each regular attack, so I am not drawing those]

So what has this discussion achieved?

Most importantly, we've proven that perfect play exists in the perfect information case of Pokémon just as it exists in games like chess, although the simultaneous move aspect of the game means that this play takes the form of probabilistic strategies

Furthermore, given any two teams, we can refer to the objective expected outcome of the game

This gives precise meaning to the notion of matchup

It is not just a qualitative assessment, but actually a real number between -1 and 1, for a given player  
(The other player has the negative of that number as their expected value)

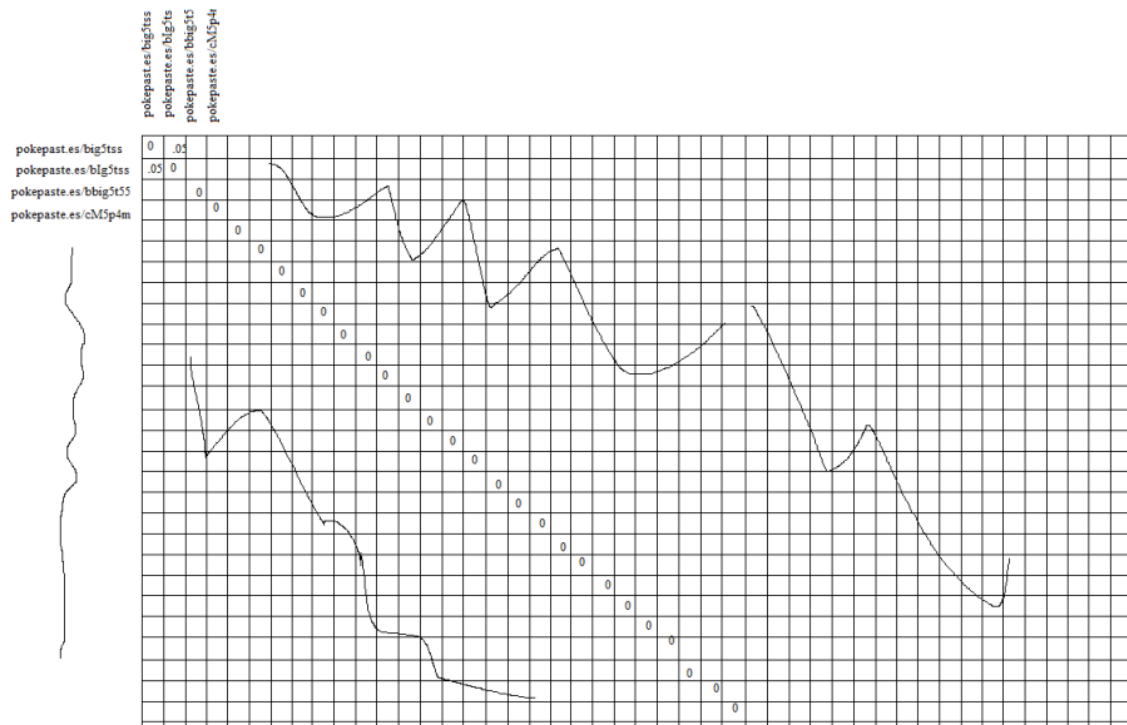
Now we are ready for the final point in this discussion

We can use these same mathematical results to discuss team building and the metagame

Now let's pretend for a moment that we can play perfectly

Then the question is: what teams should we bring?

since we can reduce any matchup to a concrete value, the process of loading something from the builder becomes its own matrix game



An action in this big game of games consists of loading up a certain team into battle

the entries are the NE value of that matchup

There are millions, billions of rows in this matrix. We have to consider every possible team. Any change in moves, item, nature counts as a distinct team. We are even considering ridiculous entries, like where Pokémon have no EVs or Caterpie squads

But more importantly, notice that this game is symmetric. Both players have the exact same actions, since they can both load any team they want. Additionally, the entries in the diagonal are all zero

The diagonal represents all the exact mirror matches. Clearly neither player has an advantage here, thus the expected value for both players is 0

also notice the symmetry around this line [mistake in picture]. If A vs B has expected score  $x$  for the first player, then B vs A has expected score  $-x$ .

There must also exist NE in this big meta-game, per the same theorems we used earlier