Pokemon ShowBot

Project report

By Denis Dekeyzer and Guillian Vansteene

Summary

Introduction

A. Pokemon recall of facts

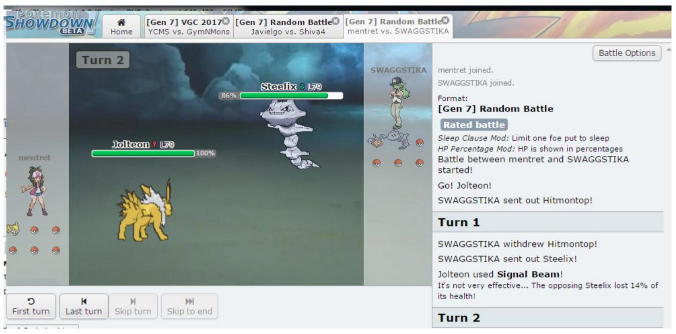
Who does not know Pokemon? These critters more Cute or less to capture, develop, exchange, and to fight them. Since their first appearance in 1996, the concept is all the rage in all formats: game video, playing cards, cartoon, manga, film, besides the astronomical amount of derivatives. With a gaming license several million copies sold Each year, the name is now part of popular culture from other major names.

Here we focus exclusively to video games, in its simplest form : the fights. No story, no exchange, level up, not nice or bad guys. Just the fighting.

Fighting in Pokemon has two facets. The first, known by a majority of the present simple: two Pokemon are fighting one against the other, they have guys who are stronger on each other (fire on plant water on fire, etc.) and a number attacks that can knock their opponent. The second addresses the issue of how much complex, each pokemon is defined by a set of parameters to the differentiate from others. Among others: a talent (comparable to a ability that triggers according to certain criteria), more or less high statistical nature (which affects statistics), an object which will influence the fight, etc.

It is obviously this second aspect that interests us, and that will dictate artificial intelligence of our bot. It would be too tedious to realize a course here on the pokemon strategy, we therefore stick to a strict minimum.

B. A combat simulator: Pokemon Showdown



mentret joined.
SWAGGSTIKA joined.
Format:
[Gen 7] Random Battle
Rated battle
Sleep Clause Mod: Limit one foe put to sleep
HP Percentage Mod: HP is shown in percentages
Battle between mentret and SWAGGSTIKA started!
Go! Jolteon!
SWAGGSTIKA sent out Hitmontop!

Turn 1

SWAGGSTIKA withdrew Hitmontop!
SWAGGSTIKA sent out Steelix!
Jolteon used **Signal Beam**!
It's not very effective... The opposing Steelix lost 14% of its health!

Turn 2

To make Pokemon battles, you need a simulator. The simulator allows create and manage all the parameters of the teams we create, the said parameters discussed above. It is accompanied by a matchmaking system that allows meet an opponent near our level as the battle format chosen.

There are ten on the web more or less driven, more or less active. Our choice is focused on Pokemon Showdown. This is an open source project completed especially by an American who became known under the pseudonym Zarel. He's regouping to this day the biggest players simulators community with between fifteen and twenty thousand permanently players.

This simulator offers many different fights formats, from classic (6v6 in preconceived team) to the most original (6v6 with Pokemon level 5 or totally random 1v1).

Classic formats are governed by a system of use: the *'Overused (* Or by example is the format that includes the most used Pokemon, while the *underused*
(UU) includes Pokemon less used, and so on. For our bot not depend on a preconceived team which could be more or less good, we turn to the format *random* . This is a 6v6 format where Pokemon are chosen randomly from a group of viable pokemons. They are graded according to their power and have a set of attacks, object, nature and talent to be chosen interesting for pokemon concerned. It is therefore a desired format as random, but nevertheless governed by certain rules to ensure a minimum balance.

C.The bot on a simulator

The idea of a bot on a simulator is not new, and we were able to build on projects already completed about it. We left quite naturally a python bot, a language that we control and fills quickly and Easy expectations.

The bot is divided into three distinct parts:

- Data Exchanges: methods of exchanges between the bot and the simulator.
- Game engine: Modeling fighting teams and Pokemon.
- Artificial intelligence: information processing and choice of club. Each party is practically

independent as the goal each time is different. We could also build on four databases provided by Showdown: a list of Pokemon and their characteristics (basic statistics, talents types), a list of moves and their details (power, purpose, type) a list of possible attacks for every pokemon in the format *random ,* and the picture of efficiency types. These were summers json adapted for ease of processing.

D. Glossary

- Stats: all statistics of a pokemon
  - life life
  - ATK: Physical attack
  - def: physical defense
  - Spa: Special Attack
  - spd: Special Defense
  - spe: Speed
- Type: Each Pokémon has one or two types, each attack has one. Types are more or less powerful against each other (see sources type table)
- Nature ; Nature acts on statistics, it increases and a lower one other.

- Talent (a pokemon): It is triggered in certain situations. It can act on a lot of aspects such as the effectiveness of an attack or a statistical pokemon. Each Pokemon has access to a limited list of talent that he individuals.

- Subject: In the same way that the talents they are activated in certain situations and can act on issues such as the effectiveness of an attack or statistics pokemon bringing the. All Pokémon have access to the same list of items
- Capacity, move, attack: Each Pokemon has up to four same capacity time. They are chosen from a set that pokemon can learn in Games. It may be physical, or special support. These past do not inflict damage directly.

- Switch: On each turn, you can choose to attack or change pokemon. This is called switch. A pokemon that switch does not work.
- Action: A pokemon can do an action per turn, either a switch or a stroke.
- Tour: Each turn, each of the two active Pokemon perform an action, the Quick goes first unless switch. The switch takes place before the move.
- KO: A pokemon knocked out is no longer able to fight. Six Pokemon KO and the match stop.

- Status: There are various problems of status, such as paralysis, burn, poisoning, gel or sleep. These impact the stats and actions pokemon who suffers.

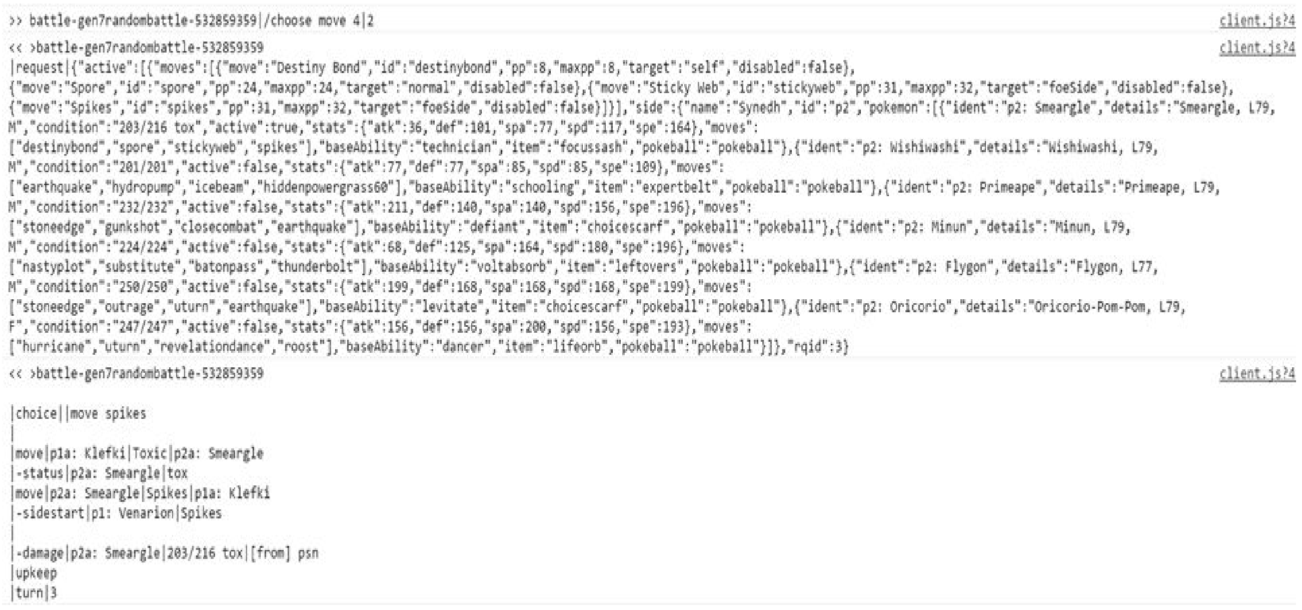
- Buff (or boost): multiplier applied to a stat. Ranging from 0.25 x to x 4. Applied by some attacks. Defaults to x 1

I. Data Exchange

A. Client connections - server

The first question we asked ourselves is naturally the way a dialogue with the bot simulator. It is available in web version, we are first turned to libraries to exchange with a page web. We threw our sights on Selenium, a simple library to use, which allows to retrieve a web page tags and interact with buttons and forms. However, many limitations have appeared to us, such as management several times or the need to recover the entire continuously webpage.

Digging the Git of the simulator project, we fell on the protocol client-server data exchange for this, and we discovered the extraordinary Websockets world. In short, it is a communication channel between a server and client event in form, ie without having to recover response from the server each time. The system is relatively new, and is accompanied python library asyncio necessary to make asynchronous (Which allows for multiple actions at the same time) in Python 3.5+. It was the answer many of our concerns because we do not need web window more limitation on the number of fights or performance loss.



```
>> battle-gen7randombattle-532859359|/choose move 4|2                                                    client.js?4
<< >battle-gen7randombattle-532859359                                                                     client.js?4
|request|{"active":[{"moves":[{"move":"Destiny Bond","id":"destinybond","pp":8,"maxpp":8,"target":"self","disabled":false},
{"move":"Spore","id":"spore","pp":24,"maxpp":24,"target":"normal","disabled":false},{"move":"Sticky Web","id":"stickyweb","pp":31,"maxpp":32,"target":"foeSide","disabled":false},
{"move":"Spikes","id":"spikes","pp":31,"maxpp":32,"target":"foeSide","disabled":false}]}],"side":{"name":"Synedh","id":"p2","pokemon":[{"ident":"p2: Smeargle","details":"Smeargle, L79,
M","condition":"203/216 tox","active":true,"stats":{"atk":36,"def":101,"spa":77,"spd":117,"spe":164},"moves":
["destinybond","spore","stickyweb","spikes"],"baseAbility":"technician","item":"focussash","pokeball":"pokeball"},{"ident":"p2: Wishiwashi","details":"Wishiwashi, L79,
M","condition":"201/201","active":false,"stats":{"atk":77,"def":77,"spa":85,"spd":85,"spe":109},"moves":
["earthquake","hydropump","icebeam","hiddenpowergrass60"],"baseAbility":"schooling","item":"expertbelt","pokeball":"pokeball"},{"ident":"p2: Primeape","details":"Primeape, L79,
M","condition":"232/232","active":false,"stats":{"atk":211,"def":140,"spa":140,"spd":156,"spe":196},"moves":
["stoneedge","gunkshot","closecombat","earthquake"],"baseAbility":"defiant","item":"choicescarf","pokeball":"pokeball"},{"ident":"p2: Minun","details":"Minun, L79,
M","condition":"224/224","active":false,"stats":{"atk":68,"def":125,"spa":164,"spd":180,"spe":196},"moves":
["nastyplot","substitute","batonpass","thunderbolt"],"baseAbility":"voltabsorb","item":"leftovers","pokeball":"pokeball"},{"ident":"p2: Flygon","details":"Flygon, L77,
M","condition":"250/250","active":false,"stats":{"atk":199,"def":168,"spa":168,"spd":168,"spe":199},"moves":
["stoneedge","outrage","uturn","earthquake"],"baseAbility":"levitate","item":"choicescarf","pokeball":"pokeball"},{"ident":"p2: Oricorio","details":"Oricorio-Pom-Pom, L79,
F","condition":"247/247","active":false,"stats":{"atk":156,"def":156,"spa":200,"spd":156,"spe":193},"moves":
["hurricane","uturn","revelationdance","roost"],"baseAbility":"dancer","item":"lifeorb","pokeball":"pokeball"}]},"rqid":3}
<< >battle-gen7randombattle-532859359                                                                     client.js?4

|choice||move spikes
|
|move|p1a: Klefki|Toxic|p2a: Smeargle
|-status|p2a: Smeargle|tox
|move|p2a: Smeargle|Spikes|p1a: Klefki
|-sidestart|p1: Venarion|Spikes
|
|-damage|p2a: Smeargle|203/216 tox|[from] psn
|upkeep
|turn|3
```

The first action to be performed when connecting to the server is websockets connection. The protocol for this is quite laborious:

- A connection to the server websockets, we receive a message under the format | challstr | <CHALLSTR> .
- There must then make a POST request to the server with the connection <CHALLSTR> formatted for the web (including whitespace characters and pipes) followed by the identifiers of the bot. This phase caused us many

worries since this doc under the javascript format and we went on a bot python.

- In the server response is a string < REPRESENTATIONS> it must recover.

- And finally sent to websockets server:

  / Trn <USERNAME>, 0, <REPRESENTATIONS>

Once these actions performed, it is (finally) possible to start fights.

B. incoming signals

Once started fighting, we receive each turn two websockets of from the server. Each information is in the same format : id_room | type_info | info

- The first is in the form | request | <REQUEST> , or < REQUEST> is an object json containing pokemon current information and other pokemons of the team.

- The second is the combat log. This is the detail of what happened at the turn previous. This socket actually contains a lot of information that we have not processed in full. The details of these is available on the protocol (See sources). Here are the ones we have addressed:

  - | Init | : start of a battle signal. Used to create the model. It is followed the fight id.

  - | Player | : If the following is the name of the bot, then recovers its id.
  - | Start | : Initial situation of a fight. Retrieves the name of opposing pokemon for the establishment of combat.
  - | Switch | : If the following is the id of the opponent, retrieve the name the opposing pokemon to treat it.
  - | -Boost | or | - unboost | : Lets find out if changes in stats.
  - | -Status | : Lets find out if there is a change in status.
  - | -Item | or | - enditem | : Allows to update the object of opposing pokemon.
  - | Turn | : Start a combat round.
  - | Upkeep | :
    - If tracking faint | <ID_BOT> => death of the current pokemon, triggers the switch phase.

    - If tracking faint | <ID_ADVERSAIRE> => death of the opposing Pokemon.
  - | Win | : Victory / defeat, after the battle.

There is almost no data on treatment in this part. It's about only to forward the information to the right features of the game engine. It is a rather tedious job in the sense that there is no way to do it "properly." This stage usually results in a series of conditions to meet the corresponding to the message sent.

The only action taken is the instantiation of fighting (see game / Battle Engine).

C.Signaux outgoing

There are quite a few outgoing signals, the shares of the remaining limited bot. They stay on the same pattern as the received messages, namely id_room | type_info | info. Here they are :

- Go search
  - | / Search gen7randombattle
  - Starts the matchmaking tool. We are in format *random* and 7 to the generation of Pokemon.

- Challenge a particular player
  - | / Challenge <PLAYER_NAME> gen7randombattle
  - Allows you to directly launch a fight against a player. Even format fight as above.

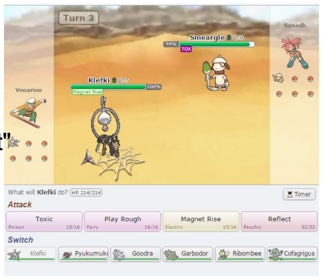
- Send a message
  - <ID_ROOM> | <MESSAGE>
  - Sends a message. The simulator includes a chat tool, we are not limited to fighting here, the message can be sent to anyone.

- selection move
  - <ID_COMBAT> | / choose move x | <TOWER>
  - Allows you to choose an attack from the four available (0 <x <5)
- selection switch
  - <ID_COMBAT> | / x switch choose | <TOWER>
  - Allows you to choose a Pokemon from six of the team (0 <x <7).
- Leaving a game
  - | / Leave <ID_COMBAT>
  - Exits a fight. Leave a battle does not mean losing it can join in too, although this feature has not interest in our case.

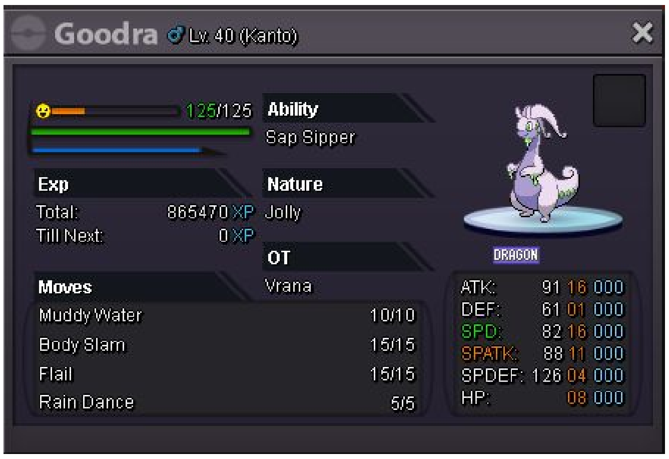
In practice, all sent messages with the same pattern, they are sent by the same function. We call it with different parameters to complete each part of the message.

II. Game engine

For the bot to react depending on the combat, must store some information, such as teams, Pokemon, the stats. In the source code, it is to each object that will be instantiated and filled. Each method is called only by the "parent" object.



A. Pokemon



Goodra ♂ Lv. 40 (Kanto)

125/125

Ability
Sap Sipper

Exp
Total: 865470 XP
Till Next: 0 XP

Nature
Jolly

OT
Vrana

DRAGON

Moves			
Muddy Water	10/10		
Body Slam	15/15		
Flail	15/15		
Rain Dance	5/5		

ATK:	91	16	000
DEF:	61	01	000
SPD:	82	16	000
SPATK:	88	11	000
SPDEF:	126	04	000
HP:		08	000

The objects "pokemon" include all specific information pokemon said: name, condition (KO or not), type, objects, potential, potential attacks talents stats, buffs.

There are two ways to create a pokemon. Once instantiated, or we knows the pokemon (this is one of those of bot) or you do not know (this is one of those of the adversary). The main difference comes in the information which are provided by the simulator.

If known pokemon, we know his attacks, his talent, his purpose, his stats. It is so easy to fill as much information. However, if it comes to that of the opponent, then we do not know all the information. We must then search the databases at our disposal that allow us to derive maximum information about the pokemon.

Team B.

The "team" objects contain a list of Pokemon. There is a maximum of six Pokémon per team.

Logically, it is possible to add pokemons (such as and as one discovers the opposing Pokemon), but also possible to delete. Indeed, some objects or moves used to change the appearance, the stats and nature a pokemon. In this case, it should be possible delete the old for the update.

The Pokemon KO are not removed, they can calculate the number of Pokemon remaining to the opponent when they have not all been revealed.

The method *active* retrieves the Pokémon currently in battle.

C.Battle

The object "Battle" manages everything about the fighting. First, it is he who directly retrieves information from incoming signals.

It is defined by its creation < BATTLE_ID> , recovered first websocket of fight. This allows to differentiate from other ongoing fighting, but it is also used in the outgoing signals. In the case of a fight, is the < ROOM_ID> .

It is saved to the bot team and that of the opponent, created instantiation, but also the current round.

Each turn, this object will retrieve the json corresponding to the WebRequest | Request | and treat. As mentioned specifically, it contains detailed information on the active pokemon and other information on the rest of the team bot. The request is sent at every turn, it allows us to update the entire team regularly. The first advantage of this system is to manage all changes forms following the use of objects, talents or abilities. It is also an opportunity to manage the reset boosts a pokemon. Indeed, these being reset if the pokemon is returned in his Poke Balls, they must be reset, and it is by this means that we do.

This is what also managed to update the opposing pokemon. At the beginning of each then fight with each switch, a different pokemon is set up, it is necessary to update the opposing team (add it if it is not already present), create the pokemon if he did not exist, etc.

Finally, it is also Here we will start searching for best action (cf. Artificial intelligence). Indeed, Only in this item found all information about the bot team and that of the opponent. When the search performed, is redirected to the corresponding output signal (move or switch).

III. Artificial intelligence

"And here is the heart of the house: the kitchen "Not at all.. Only the brain bot. We have already seen its members (data exchange) and body (the engine), here his head!

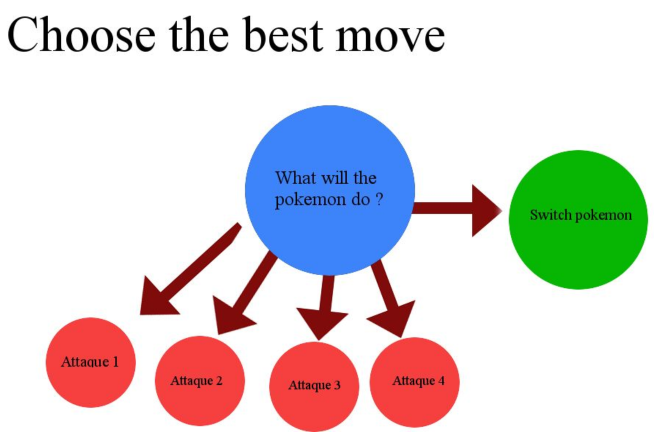
Each turn, a player has the choice between using an attack (a move) or change for another pokemon his team (a switch), if indeed there remain Pokemon. The delicacy AI comes from the leveling of this choice, to arrive to ensure that the bot can appreciate the true value of the dangerousness of the opposing pokemon compared to his.

We initially chose to work from the Minimax. It's about to assume that whatever the outcome of the round, we get to the end of the round with "Less" than that with which we left early. The idea is to make the choice the least bad to limit losses.

A. Choosing the best action

In reality pokemon, it is sometimes better to choose what will lose more to the opponent as a switch deals no loss to the opponent and can therefore be a more substantial loss in the long term if misused, while limiting losses for the current turn. That is why we have departed not evil of Minimax initial algorithm and created an algorithm specific to our needs.

So this is the balance between "I make the most possible damage to my opponent "and" I suffered the least this turn "must be found.



1. Best move

To calculate the efficiency of an attack, it would in theory apply the formula Complete damage pokemon. It is as follows:

Damage done = (((((((Level × 2 ÷ 5) + 2) × × Power Off [Spé] ÷ 50) ÷ Def [Spé]) × Mod1) + 2) × CC × Mod2 × R ÷ 100) × × STAB Type1 Type2 × × Mod3

Where DC, STAB, Mod1, Mod2 and Mod3 are values that change depending types, objects, nature, combat capabilities used (details in the source).

This is an extremely complicated formula to set up, because dependent on too many factors that the bot should have to manage. Moreover, the difference efficiency between the complete formula and a model is too low to justify such expenditure of time. The main difference is the inclusion of stats pokemon, these depending on the level, nature and other hidden features, still bringing other parameters to manage.

The first action to take for each attack is to differentiate attacks damage to those for a boost (called "status of attacks"). Indeed, the interest of first is quantifiable in terms of the damage, others will be so arbitrary.

For the damage of attacks, it is based on the initial damage of the attack, then applies a number of multipliers based on Pokemon:

- Application of the types table (see source)
- If the Pokemon is the same type as the attack, the damage is multiplied by 1.5
- Some objects change the damage of 1.2, 1.3 or 1.5 damage
- Some types or objects negate the damage as Levitation, which reduces undo the damage of ground attacks.
- Then you apply the multiplier boosts stats.

To the attacks of status, there are two types, volatile attacks, disappear when the pokemon back to his pokeball and persistent that inflict status problems. These are managed independently according stats and Pokemon types, each dependent on the characteristics of the defender pokemon.

Pour les attaques non persistantes, il s’agit de savoir si une attaque est rentable sur several turns. For example, an attack that makes my pokemon faster than the other will always profitable, as the next round, my pokemon attack before the other. They are also managed independently.

Once the effectiveness of an attack is determined, simply return the most interesting.

2. Best switch

The progress of the switch best is seemingly simple: for every pokemon in our team, look who is the most interesting in relation to the opponent. For know which one is the most interesting, comparing the attacks of two Pokemon, and takes one who does the most damage.

Here we have the theory. In practice there is one more parameter that must be matters is the speed of a pokemon. At every turn, the fastest pokemon acts first, and then the second is, if it is still able to fight. And this is all the nuance If you switch on a slower pokemon that the opponent, it will then take the attack of the tour being and that of the next round before finally can attack.

This is why we must qualify efficiency values: If I inflict significant amount of damage and I'm faster than my opponent, so I do not need consider its damage, since there is a big chance that he is gone for make. Conversely, if my opponent is faster than me and he has an attack that inflicts me a significant amount of damage, then there is a good chance that I Be there to inflict my damage, and they are no longer taken into account. However, if my opponent can not inflict significant damage, so my pokemon remains interesting because I would still be alive.

Here we have the processing performed to calculate the best switch. This is done two occasions: at the death of a pokemon of the team bot and each turn action should be performed. To be quite rigorous, the calculation should be slightly different from one case to another, because the pokemon from a switch always suffer an attack more. It is however something significant on many fights.

3. Attacking or change: you must choose

"So Heuu a good bot is a bot that attacks and switch, and Heuu bah bad bot is a bot that attacks and switch, but it is a bad bot."

This is actually the point that differentiates a good bot of a bad bot know when switcher switcher and when not. As a player, this decision is made on lot used: know the danger of a pokemon, know what her perfect counterattacks or its strengths, how to replicate, etc.

The bot has a huge advantage is the ability to say for sure if the pokemon opponent is dangerous or not. However, it is very complicated to know if dangerous enough to justify a switch, or if it is dangerous but we do not have a better answer for him than we have at the present moment. It is more so as some Pokémon have different behaviors related to attacks the bot can not manage properly.

Currently, the bot will switcher if two conditions are met:

- If there are more effective than current pokemon
- If the opposing pokemon is faster and causes extensive damage or if the pokemon bot inflicts little damage.

This is a behavior that is appropriate and responds to most situations that the bot may be, there is however some situations that only the deep learning can answer.

B. Parts of the Turn

1. Receive websockets
   - 1.1. Updating the bot team
   - 1.2. Update of the opposing team
2. Choosing the best action
   - 2.1. Choosing the best attack
     - 2.1.1. Course attacks
       - 2.1.1.1. Calculation types
       - 2.1.1.2. Application of modifiers
       - 2.1.1.3. Management status of attacks
     - 2.1.2. Back to the best attack
   - 2.2. Choosing the best pokemon which switcher
     - 2.2.1. Course of Pokemon
       - 2.2.1.1. Choosing the best attack pokemon
       - 2.2.1.2. Choosing the best attack the opposing pokemon
       - 2.2.1.3. Calculation of efficiency pokemon
     - 2.2.2. Back best pokemon
   - 2.3. Choice of switch or move
   - 2.4. Back of the best action
3. Launch of the action
4. Conclusion to perform according to the combat log
   - 4.1. If the current death pokemon
     - 4.1.1. Choosing the best switch
     - 4.1.2. Launched switch
   - 4.2. If death of the opposing pokemon
     - 4.2.1. Waiting for opponent
     - 4.2.2. Updating the opposing pokemon
   - 4.3. If opposing switch
     - 4.3.1. Updating the opposing pokemon
   - 4.4. If after the battle
     - 4.4.1. leave
5. End Turn

C.Vers the deep learning

We have seen, static AI has limits on switch because its decisions greatly depend on its implementation and the good will of its creator, unlike the calculation of efficiency for example, where the power values depend only a calculation. This is something that is resolvable by the deep learning. In Indeed, with the habit, it is possible to tell if a pokemon is good or not in relation to a other, and therefore to provide more relevant switches, both in their frequency to choose the pokemon best able to respond to a threat.

The main difficulty that we could not identify, and why we do not present deep learning of this model is that it is extremely complicated to deduct a ride if it was effective or not. For example, for attacks that inflict statutes problems like poison on the turn in progress, damage will be ridiculous (between -6 and -12% of the opposing life); but several turns and little pokemon is resistant, it is a damage amount which is far from negligible and can tip a situation of one against one in favor of the pokemon that placed this status.

The solution would be to retain the data until the change of two Pokemon and then make a conclusion based on the damage done both sides. It is a model that appears however complicated to implement because it takes remember all parameters of a combat log to reach a reliable conclusion.

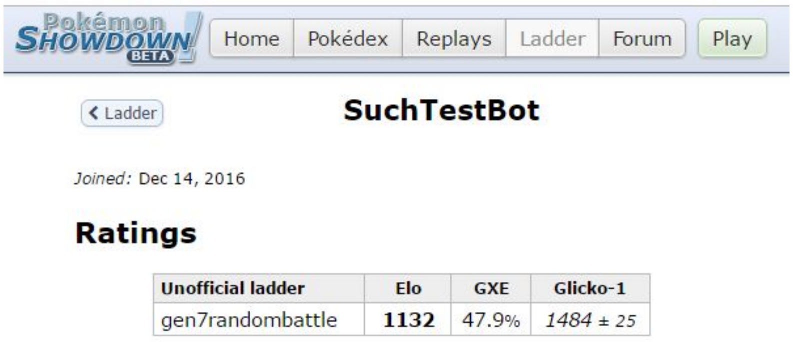
The deep learning also solve the problem based on Pokemon timing. We have seen the example of the above poison, but there are many other techniques, such as theft of life, health regeneration, or use skills to earn a bonus. This would require that the bot saves techniques on fights already made by good players and records the sequence attacks. It's something else ...

Conclusion

It was a pleasure working on this bot. We learned a lot, both pure programming level (we never touched the websockets nor asyncio well that it remains very dark) at level IA and workgroup.

It was also the opportunity to work on a subject that radically changes what We can offer traditional projects, especially in terms of study subject (the pokemons). It has allowed us to work on a subject that we liked and "fun" in working.

In the end and in the current state of things, the bot has a ratio of victories / defeats of about 55%, not counting the many stops due to bugs or massive stops (8 stops fighting simultaneous example). To give a figure for comparison, a "good" player is around 60-65%, so this is very far from being a bad score. The score below takes into account all the stops and covers about 1,000 fights.



SuchTestBot

Joined: Dec 14, 2016

Ratings

Unofficial ladder	Elo	GXE	Glicko-1
gen7randombattle	**1132**	47.9%	*1484 ± 25*

sources

Pokemon Showdown

pokemonshowdown.com

https://github.com/Zarel/Pokemon-Showdown

https://github.com/Zarel/Pokemon-Showdown-Client

Protocol websockets:

https://github.com/Zarel/Pokemon-Showdown/blob/master/PROTOCOL.md

Similar Project:

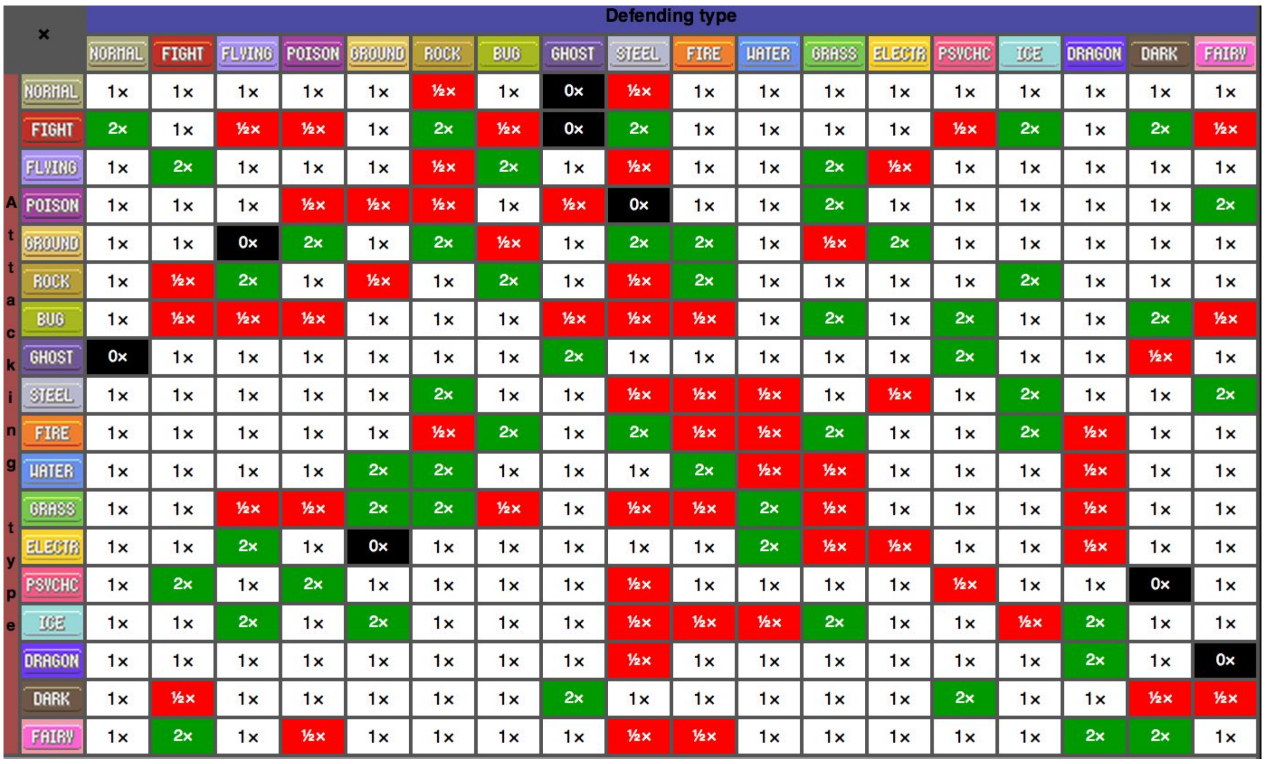
http://www.smogon.com/forums/threads/pokemon-showdown-ai-bot.3547689/

https://web.archive.org/web/20160416193733/http://nuggetbridge.com/blogs/entry/ 1548-sho
wdown-bot-pokemon-and-artificial-intelligence /

Pokémaths: damage calculations:

https://www.pokebip.com/page__jeuxvideo__guide_tactique_strategie_pokemon__formules_mathematiques.html

Table types:



×	Defending type																	
Attacking type	NORMAL	FIGHT	FLYING	POISON	GROUND	ROCK	BUG	GHOST	STEEL	FIRE	WATER	GRASS	ELECTR	PSYCHC	ICE	DRAGON	DARK	FAIRY
NORMAL	1×	1×	1×	1×	1×	½×	1×	0×	½×	1×	1×	1×	1×	1×	1×	1×	1×	1×
FIGHT	2×	1×	½×	½×	1×	2×	½×	0×	2×	1×	1×	1×	1×	½×	2×	1×	2×	½×
FLYING	1×	2×	1×	1×	1×	½×	2×	1×	½×	1×	1×	2×	½×	1×	1×	1×	1×	1×
POISON	1×	1×	1×	½×	½×	½×	1×	½×	0×	1×	1×	2×	1×	1×	1×	1×	1×	2×
GROUND	1×	1×	0×	2×	1×	2×	½×	1×	2×	2×	1×	½×	2×	1×	1×	1×	1×	1×
ROCK	1×	½×	2×	1×	½×	1×	2×	1×	½×	2×	1×	1×	1×	1×	2×	1×	1×	1×
BUG	1×	½×	½×	½×	1×	1×	1×	½×	½×	½×	1×	2×	1×	2×	1×	1×	2×	½×
GHOST	0×	1×	1×	1×	1×	1×	1×	2×	1×	1×	1×	1×	1×	2×	1×	1×	½×	1×
STEEL	1×	1×	1×	1×	1×	2×	1×	1×	½×	½×	½×	1×	½×	1×	2×	1×	1×	2×
FIRE	1×	1×	1×	1×	1×	½×	2×	1×	2×	½×	½×	2×	1×	1×	2×	½×	1×	1×
WATER	1×	1×	1×	1×	2×	2×	1×	1×	1×	2×	½×	½×	1×	1×	1×	½×	1×	1×
GRASS	1×	1×	½×	½×	2×	2×	½×	1×	½×	½×	2×	½×	1×	1×	1×	½×	1×	1×
ELECTR	1×	1×	2×	1×	0×	1×	1×	1×	1×	1×	2×	½×	½×	1×	1×	½×	1×	1×
PSYCHC	1×	2×	1×	2×	1×	1×	1×	1×	½×	1×	1×	1×	1×	½×	1×	1×	0×	1×
ICE	1×	1×	2×	1×	2×	1×	1×	1×	½×	½×	½×	2×	1×	1×	½×	2×	1×	1×
DRAGON	1×	1×	1×	1×	1×	1×	1×	1×	½×	1×	1×	1×	1×	1×	1×	2×	1×	0×
DARK	1×	½×	1×	1×	1×	1×	1×	2×	1×	1×	1×	1×	1×	2×	1×	1×	½×	½×
FAIRY	1×	2×	1×	½×	1×	1×	1×	1×	½×	½×	1×	1×	1×	1×	1×	2×	2×	1×