Poke-env

Sep 20, 2021

Contents

This project aims at providing a Python environment for interacting in pokemon showdown battles, with reinforcement learning in mind. Welcome to its documentation!

Poke-env offers a simple and clear API to manipulate Pokemons, Battles, Moves and many other pokemon showdown battle-related objects in Python. It also exposes an open ai gym interface to train reinforcement learning agents.

Warning: This module currently supports most gen 8 and 7 single battle formats. Support for doubles formats and gen 4-5-6 formats is preliminary. If you want a specific format to be supported, please open an issue.

CHAPTER 1

Table of contents

1.1 Getting started

This sections will guide you in installing `poke-env` and configuring a suited showdown server.

1.1.1 Installing `poke-env`

`poke-env` requires python >= 3.6 to be installed. It has a number of dependencies that are listed here and that will be installed automatically.

Installation can be performed via pip:

```
pip install poke-env
```

1.1.2 Configuring a showdown server

`poke-env` communicates with a pokemon showdown server. A public implementation of showdown is hosted here, and can be used to test your agents against real human players.

However, this implementation:

- Requires an internet connection at all time
- Has numerous performance limitation (move rate, number of concurrent battles...)
- Is not meant to be used to train agents

Therefore, it is recommended to host you own server. Fortunately, Pokemon Showdown is open-source and just requires Node.js v10+. `poke-env` used to maintain a custom and optimized fork, but its features have been merged in the official showdown implementation.

To get started, you will first need to install node v10+. Then, you can clone the pokemon showdown repo:

```
git clone https://github.com/smogon/pokemon-showdown.git
cd pokemon-showdown
npm install
cp config/config-example.js config/config.js
```

Everything is now almost ready to create your first agent: you just have to start the showdown server:

```
node pokemon-showdown start --no-security
```

Warning: The `--no-security` flag deactivates several important security features, so do not run a public server with this flag if you are not sure of what you are doing. This flag also removes most of showdown's rate limiting, authentication and throttling, which allows its usage to train AI agents effectively.

You should then get something like this:

```
NEW GLOBAL: global
NEW CHATROOM: lobby
NEW CHATROOM: staff
Worker 1 now listening on 0.0.0.0:8000
Test your server at http://localhost:8000
```

If that is the case, congratulations! You just launched your server! You can now refer to *Examples* to create your first agent.

1.1.3 Creating agents

In `poke-env`, agents are represented by instances of python classes inheriting from `Player`. This class incorporates everything that is needed to communicate with showdown servers, as well as many utilities designed to make creating agents easier.

To get started on creating an agent, we recommended taking a look at explained examples.

- Running agent: *Cross evaluating random players*
- Creating a first non-trivial agent: *Creating a simple max damage player*
- Using Reinforcement Learning to train an agent: *Reinforcement learning with the OpenAI Gym wrapper*
- Using teams and managing team preview in non-random formats: *Adapting the max player to gen 8 OU and managing team preview*
- Building a custom teambuilder: *Creating a custom teambuilder*

1.1.4 Configuring showdown players

`Player` instances need a player configuration corresponding to showdown accounts. By default, such configurations are automatically generated for each `Player`. These automatically generated configurations are compatible with servers bypassing authentication, such as the recommended fork mentionned above.

You can create custom configurations, for instance to use existing showdown accounts. To do so, use the `player_configuration` argument of `Player` constructors: you can pass in a `PlayerConfiguration`, which are named tuples with two arguments: an username and a password.

Users without authentication

If your showdown configuration does not require authentication, you can use any username and set the password to `None`.

```
from poke_env.player_configuration import PlayerConfiguration

# This will work on servers that do not require authentication, which is the
# case of the server launched in our 'Getting Started' section
my_player_config = PlayerConfiguration("my_username", None)
```

Users with authentication

If your showdown configuration uses authentication, the values of each `player_configuration` that you create must be defined in the server's authentication database. On pokemonshowdown.com, you can achieve this by registering an username.

```
from poke_env.player_configuration import PlayerConfiguration

# This object can be used with a player connecting to a server using authentication
# The user 'my_username' must exist and have 'super-secret-password' as his password
my_player_config = PlayerConfiguration("my_username", "super-secret-password")
```

1.1.5 Connecting your bots to showdown

`Player` instances need a server configuration pointing to a websocket endpoint and an authentication endpoint. By default, `Player` instances will use `LocalhostServerConfiguration`, which corresponds to the default configuration of local showdown servers.

You can set custom configurations by using the `server_configuration` argument of `Player` instances. It expects a `ServerConfiguration` object, which is a named tuple containing a server url and authentication url.

`poke-env` includes two ready-to-use `ServerConfiguration` objects: `LocalhostServerConfiguration` and `ShowdownServerConfiguration`.

The first one points to `locahost:8000` - the default endpoint for a local showdown server - whereas the second one points to `https://play.pokemonshowdown.com/`. Both use the same authentication endpoint, https://play.pokemonshowdown.com/action.php?.

If you use our custom fork of showdown, as mentionned in Getting Started, players do not need to authenticate to battle. This effectively skips authentication calls: your agents can access your server without an internet connection.

1.1.6 Custom server configuration

You can create your own server configuration if you want to connect your player to another server. You can do so like that:

```
from poke_env.server_configuration import ServerConfiguration

# If your server is accessible at my.custom.host:5432, and your authentication
# endpoint is authentication-endpoint.com/action.php?
my_server_config= ServerConfiguration(
    "my.custom.host:5432",
    "authentication-endpoint.com/action.php?"
```

(continues on next page)

(continued from previous page)

```
)

# You can now use my_server_config with a Player object
```

1.2 Examples

This page lists detailled examples demonstrating how to use this package. They are meant to cover basic use cases.

1.2.1 Cross evaluating random players

The corresponding complete source code can be found here.

Note: A similar example using gen 7 mechanics is available here.

The goal of this example is to demonstrate how to run existing agents locally, and how to easily measure the relative performance of multiple agents with the `cross_evaluate` utility function.

Note: This example uses `tabulate` ti format results. You can install it by running `pip install tabulate`.

Getting *something* to run

`poke-env` uses `asyncio` for concurrency: most of the functions used to run `poke-env` code are async functions. Using asyncio is therefore required.

Let's start by defining a `main` and some boilerplate code to run it with `asyncio`:

```
# -*- coding: utf-8 -*-
import asyncio

async def main():
    pass

if __name__ == "__main__":
    asyncio.get_event_loop().run_until_complete(main())
```

Creating random players

We can start by creating three players. Players (or agents) are the objects that control the decisions taken in battle: here, we create `RandomPlayer` s, which take decisions randomly. By default, `Player` instances will automatically use a generated username and try to connect to a showdown server hosted locally.

You can modify this behavior by using the `player_configuration` and `server_configuration` parameters of the constructor of `Player` objects, during initialization.

By default, players play the `gen8randombattle` format. You can specify another battle format by passing a `battle_format` parameter. If you choose to play a format that requires teams, you'll also need to define it with the `team` parameter. You can refer to *Adapting the max player to gen 8 OU and managing team preview* for an example using a custom team and format.

```
...
from poke_env.player.random_player import RandomPlayer

async def main():
    # We create three random players
    players = [
        RandomPlayer(max_concurrent_battles=10) for _ in range(3)
    ]

...
```

Note: This example supposes that you use a local showdown server that does not require authentication.

These players will play battles in the `gen8randombattle` battle format, connect to a locally hosted server, and play up to 10 battles simultaneously.

Cross evaluating players

Now that our players are defined, we can evaluate them: every player will play 20 games against every other player, for a total of 60 battles.

To do so, we can use the helper function `cross_evaluate`:

```
...
from poke_env.player.utils import cross_evaluate

async def main():
    ...
    cross_evaluation = await cross_evaluate(players, n_challenges=20)

...
```

Finally, we can display the results in a nice table:

```
...
from tabulate import tabulate

async def main():
    ...
    # Defines a header for displaying results
    table = [["-"] + [p.username for p in players]]

    # Adds one line per player with corresponding results
    for p_1, results in cross_evaluation.items():
        table.append([p_1] + [cross_evaluation[p_1][p_2] for p_2 in results])

    # Displays results in a nicely formatted table.
    print(tabulate(table))

...
```

Running the whole file should take a couple of seconds and print something similar to this:

```
--------------  --------------  --------------  --------------
-               RandomPlayer 1  RandomPlayer 2  RandomPlayer 3
RandomPlayer 1                  0.53            0.52
RandomPlayer 2  0.47                            0.5
RandomPlayer 3  0.48            0.5
--------------  --------------  --------------  --------------
```

If you want to create a custom player, take a look at the *Creating a simple max damage player* example.

If you want to jump into Reinforcement Learning, take a look at the *Reinforcement learning with the OpenAI Gym wrapper* example.

1.2.2 Creating a simple max damage player

The corresponding complete source code can be found here.

Note: A similar example using gen 7 mechanics is available here.

The goal of this example is to explain how to create a first custom agent. This agent will follow simple rules:

- If the active pokemon can attack, it will attack and use the move with the highest base power
- Otherwise, it will perform a random switch

Creating a player

The player that we are going to implement does not need to be trained: we can therefore directly inherit from the `Player` class.

Let's create the base class:

```
# -*- coding: utf-8 -*-
from poke_env.player.player import Player

class MaxDamagePlayer(Player):
    pass
```

`Player`'s has one abstract method, `choose_move`. Once implemented, we will be able to instantiate and use our player.

Creating a `choose_move` method

Method signature

The signature of `choose_move` is `choose_move(self, battle: Battle) -> str`: it takes a `Battle` object representing the game state as argument, and returns a move order encoded as a string. This move order must be formatted according to the showdown protocol. Fortunately, `poke-env` provides utility functions allowing us to directly format such orders from `Pokemon` and `Move` objects.

We therefore have to take care of two things: first, reading the information we need from the `battle` parameter. Then, we have to return a properly formatted response, corresponding to our move order.

Selecting a move

The `battle` parameter is an object of type `Battle` which encodes the agent's current knowledge of the game state. It offers several properties that make accessing the game state easy. Some of the most notable are `active_pokemon`, `available_moves`, `available_switches`, `opponent_active_pokemon`, `opponent_team` and `team`.

In this example, we are going to use `available_moves`: it returns a list of `Move` objects which are available this turn.

We can therefore test if at least one move can be used with `if battle.available_moves:`. We are interested in the base power of available_moves, which can be accessed with the `base_power` property of `Move` objects.

```
class MaxDamagePlayer(Player):
    def choose_move(self, battle):
        # If the player can attack, it will
        if battle.available_moves:
            # Finds the best move among available ones
            best_move = max(battle.available_moves, key=lambda move: move.base_power)
```

Returning a choice

Now that we have selected a move, we need to return a corresponding order, which takes the form of a string. Fortunately, `Player` provides a method designed to craft such strings directly: `create_order`. It takes a `Pokemon` (for switches) or `Move` object as argument, and returns a string corresponding to the order. Additionally, you can use its `mega`, `z_move` and `dynamax` parameters to mega evolve, use a z-move, dynamax or gigantamax, if possible this turn.

We also have to return an order corresponding to a random switch if the player cannot attack. `Player` objects incorporate a `choose_random_move` method, which we will use if no attacking move is available.

```
class MaxDamagePlayer(Player):
    def choose_move(self, battle):
        # If the player can attack, it will
        if battle.available_moves:
            # Finds the best move among available ones
            best_move = max(battle.available_moves, key=lambda move: move.base_power)
            return self.create_order(best_move)

    # If no attack is available, a random switch will be made
    else:
        return self.choose_random_move(battle)
```

Running and testing our agent

We can now test our agent by making it battle a random agent. The complete code is:

```
# -*- coding: utf-8 -*-
import asyncio
import time

from poke_env.player.player import Player
from poke_env.player.random_player import RandomPlayer
```

(continues on next page)

(continued from previous page)

```
class MaxDamagePlayer(Player):
    def choose_move(self, battle):
        # If the player can attack, it will
        if battle.available_moves:
            # Finds the best move among available ones
            best_move = max(battle.available_moves, key=lambda move: move.base_power)
            return self.create_order(best_move)

        # If no attack is available, a random switch will be made
        else:
            return self.choose_random_move(battle)


async def main():
    start = time.time()

    # We create two players.
    random_player = RandomPlayer(
        battle_format="gen8randombattle",
    )
    max_damage_player = MaxDamagePlayer(
        battle_format="gen8randombattle",
    )

# Now, let's evaluate our player
await max_damage_player.battle_against(random_player, n_battles=100)

print(
    "Max damage player won %d / 100 battles [this took %f seconds]"
    % (
        max_damage_player.n_won_battles, time.time() - start
    )
)


if __name__ == "__main__":
    asyncio.get_event_loop().run_until_complete(main())
```

Running it should take a couple of seconds and print something similar to this:

```
Max damage player won 92 / 100 battles [this took 6.320682 seconds]
```

If you want to use Reinforcement Learning, take a look at the *Reinforcement learning with the OpenAI Gym wrapper* example.

1.2.3 Reinforcement learning with the OpenAI Gym wrapper

The corresponding complete source code can be found here.

Note: A similar example using gen 7 mechanics is available here.

The goal of this example is to demonstrate how to use the open ai gym interface proposed by `EnvPlayer`, and to train a simple deep reinforcement learning agent comparable in performance to the `MaxDamagePlayer` we created in *Creating a simple max damage player*.

Note: This example necessitates keras-rl (compatible with Tensorflow 1.X) or keras-rl2 (Tensorflow 2.X), which implement numerous reinforcement learning algorithms and offer a simple API fully compatible with the Open AI Gym API. You can install them by running `pip install keras-rl` or `pip install keras-rl2`. If you are unsure, `pip install keras-rl2` is recommended.

Warning: `keras-rl2` version 1.0.4 seems to be causing problems with this example. While we are trying to find a workaround, please try using version 1.0.3 with python version 3.6.

Implementing rewards and observations

The open ai gym API provides *rewards* and *observations* for each step of each episode. In our case, each step corresponds to one decision in a battle and battles correspond to episodes.

Defining observations

Observations are embeddings of the current state of the battle. They can be an arbitrarily precise description of battle states, or a very simple representation. In this example, we will create embedding vectors containing:

- the base power of each available move
- the damage multiplier of each available move against the current active opponent pokemon
- the number of non fainted pokemons in our team
- the number of non fainted pokemons in the opponent's team

To define our observations, we will create a custom `embed_battle` method. It takes one argument, a `Battle` object, and returns our embedding.

Defining rewards

Rewards are signals that the agent will use in its optimization process (a common objective is optimizing a discounted total reward). `EnvPlayer` objects provide a helper method, `reward_computing_helper`, that can help defining simple symmetric rewards that take into account fainted pokemons, remaining hp, status conditions and victory.

We will use this method to define the following reward:

- Winning corresponds to a positive reward of 30
- Making an opponent's pokemon faint corresponds to a positive reward of 1
- Making an opponent lose % hp corresponds to a positive reward of %
- Other status conditions are ignored

Conversly, negative actions lead to symettrically negative rewards: losing is a reward of -30 points, etc.

To define our rewards, we will create a custom `compute_reward` method. It takes one argument, a `Battle` object, and returns the reward.

Defining our custom class

Our player will play the `gen8randombattle` format. We can therefore inheritate from `Gen8EnvSinglePlayer`.

```
# -*- coding: utf-8 -*-
from poke_env.player.env_player import Gen8EnvSinglePlayer

class SimpleRLPlayer(Gen8EnvSinglePlayer):
    def embed_battle(self, battle):
        # -1 indicates that the move does not have a base power
        # or is not available
        moves_base_power = -np.ones(4)
        moves_dmg_multiplier = np.ones(4)
        for i, move in enumerate(battle.available_moves):
            moves_base_power[i] = move.base_power / 100 # Simple rescaling to␣
↪facilitate learning
            if move.type:
                moves_dmg_multiplier[i] = move.type.damage_multiplier(
                    battle.opponent_active_pokemon.type_1,
                    battle.opponent_active_pokemon.type_2,
                )

        # We count how many pokemons have not fainted in each team
        remaining_mon_team = len([mon for mon in battle.team.values() if mon.
↪fainted]) / 6
        remaining_mon_opponent = (
            len([mon for mon in battle.opponent_team.values() if mon.fainted]) / 6
        )

        # Final vector with 10 components
        return np.concatenate(
            [moves_base_power, moves_dmg_multiplier, [remaining_mon_team, remaining_
↪mon_opponent]]
        )

    def compute_reward(self, battle) -> float:
        return self.reward_computing_helper(
            battle,
            fainted_value=2,
            hp_value=1,
            victory_value=30,
        )

...
```

Instanciating a player

Now that our custom class is defined, we can instantiate our RL player.

```
...
env_player = SimpleRLPlayer(battle_format="gen8randombattle")
...
```

Creating a DQN with keras-rl

We have defined observations and rewards. We can now build a model that will control our player. In this section, we will implement the DQN algorithm using keras-rl.

Defining a base model

We build a simple keras sequential model. Our observation vectors have 10 components; our model will therefore accept inputs of dimension 10.

The output of the model must map to the environment's action space. The action space can be accessed through the `action_space` property. Each action correspond to one order: a switch or an attack, with additional options for dynamaxing, mega-evolving and using z-moves.

```
...
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Sequential

# Output dimension
n_action = len(env_player.action_space)

model = Sequential()
model.add(Dense(128, activation="elu", input_shape=(1, 10,)))

# Our embedding have shape (1, 10), which affects our hidden layer dimension and␣
↪output dimension
# Flattening resolve potential issues that would arise otherwise
model.add(Flatten())
model.add(Dense(64, activation="elu"))
model.add(Dense(n_action, activation="linear"))
...
```

Defining the DQN

Now that we have a model, we can build the DQN agent. This agent combines our model with a *policy* and a *memory*. The *memory* is an object that will store past actions and define samples used during learning. The *policy* describes how actions are chosen during learning.

We will use a simple memory containing 10000 steps, and an epsilon greedy policy.

For more information regarding keras-rl, please refer to their documentation.

```
...
from rl.agents.dqn import DQNAgent
from rl.memory import SequentialMemory
from rl.policy import LinearAnnealedPolicy, EpsGreedyQPolicy
from tensorflow.keras.optimizers import Adam

memory = SequentialMemory(limit=10000, window_length=1)

# Simple epsilon greedy
policy = LinearAnnealedPolicy(
    EpsGreedyQPolicy(),
    attr="eps",
    value_max=1.0,
```

(continues on next page)

(continued from previous page)

```
    value_min=0.05,
    value_test=0,
    nb_steps=10000,
)

# Defining our DQN
dqn = DQNAgent(
    model=model,
    nb_actions=18,
    policy=policy,
    memory=memory,
    nb_steps_warmup=1000,
    gamma=0.5,
    target_model_update=1,
    delta_clip=0.01,
    enable_double_dqn=True,
)

dqn.compile(Adam(lr=0.00025), metrics=["mae"])
...
```

Training the model

Accessing the open AI Gym environment interface requires interacting with env players in the main thread without preventing other asynchronous operations from happening. The easiest way to do that is to use the `play_against` method of `EnvPlayer` instances.

This method accepts three arguments:

- `env_algorithm`: the function that will control the player. It must accept a first `player` argument, and can optionally take other arguments
- `opponent`: another `Player` that will be faced by the `env_player`
- `env_algorithm_kwargs`: a dictionary containing other objects that will be passed to `env_algorithm`

To train our agent, we will create a custom `dqn_training` function. In addition to the player, it will accept two additional arguments: `dqn` and `nb_steps`. We can pass it in a call to `play_against` as the `env_algorithm` argument.

```
...
from poke_env.player.random_player import RandomPlayer

def dqn_training(player, dqn, nb_steps):
    dqn.fit(player, nb_steps=nb_steps)

    # This call will finished eventual unfinshed battles before returning
    player.complete_current_battle()

opponent = RandomPlayer(battle_format="gen8randombattle")

# Training
env_player.play_against(
    env_algorithm=dqn_training,
    opponent=opponent,
    env_algorithm_kwargs={"dqn": dqn, "nb_steps": 100000},
```

(continues on next page)

(continued from previous page)

```
)
...
```

Evaluating the model

Similarly to the training function above, we can define an evaluation function.

```
...
def dqn_evaluation(player, dqn, nb_episodes):
    # Reset battle statistics
    player.reset_battles()
    dqn.test(player, nb_episodes=nb_episodes, visualize=False, verbose=False)

    print(
        "DQN Evaluation: %d victories out of %d episodes"
        % (player.n_won_battles, nb_episodes)
    )

# Ths code of MaxDamagePlayer is not reproduced for brevity and legibility
# It can be found in the complete code linked above, or in the max damage example
second_opponent = MaxDamagePlayer(battle_format="gen8randombattle")

# Evaluation
print("Results against random player:")
env_player.play_against(
    env_algorithm=dqn_evaluation,
    opponent=opponent,
    env_algorithm_kwargs={"dqn": dqn, "nb_episodes": 100},
)

print("\nResults against max player:")
env_player.play_against(
    env_algorithm=dqn_evaluation,
    opponent=second_opponent,
    env_algorithm_kwargs={"dqn": dqn, "nb_episodes": 100},
)
...
```

Running the whole file should take a couple of minutes and print something similar to this:

```
Training for 10000 steps ...
Interval 1 (0 steps performed)
10000/10000 [==============================] - 96s 10ms/step - reward: 0.6307
done, took 96.233 seconds
Results against random player:
DQN Evaluation: 97 victories out of 100 episodes

Results against max player:
DQN Evaluation: 65 victories out of 100 episodes
```

1.2.4 Adapting the max player to gen 8 OU and managing team preview

The corresponding complete source code can be found here.

Note: A similar example using gen 7 mechanics is available here.

This example adapts *Creating a simple max damage player* to the gen 8 overused format. In particular, it shows how to specify a team and manage team preview.

Team Preview management

We start with the `MaxDamagePlayer` from *Creating a simple max damage player*, and add a team preview method.

```
class MaxDamagePlayer(Player):
    # Same method as in previous examples
    def choose_move(self, battle):
        # If the player can attack, it will
        if battle.available_moves:
            # Finds the best move among available ones
            best_move = max(battle.available_moves, key=lambda move: move.base_power)
            return self.create_order(best_move)

        # If no attack is available, a random switch will be made
        else:
            return self.choose_random_move(battle)

    def teampreview(self, battle):
        ...
```

`teampreview` takes a `Battle` object as argument, and returns a team preview order. These orders are strings of the form `/team abcdef`, where `abcdef` is a sequence of integers from 1 to 6, which designates the pokemons in your team and determines the order in which they are ordered: in particular, the first integer defines your lead.

You can access your team with `Battle.team` and your opponent's team with `Battle.opponent_team`.

The order of the keys in `Battle.team` is the same as the order that showdown is considering: if you want to lead with the second pokemon in your team, returning `/team 213456` would work.

Here, we are going to evaluate how good of a lead each pokemon we have is, and return the one we deem to be best. To do that, we are going to need an evaluation function.

We define it as follows: we evaluate the performance of a pokemon against another one as the difference between the effectiveness of the first pokemon and the second's pokemon types. Here is an implementation:

```
def teampreview_performance(mon_a, mon_b):
    # We evaluate the performance on mon_a against mon_b as its type advantage
    a_on_b = b_on_a = -np.inf
    for type_ in mon_a.types:
        if type_:
            a_on_b = max(a_on_b, type_.damage_multiplier(*mon_b.types))
    # We do the same for mon_b over mon_a
    for type_ in mon_b.types:
        if type_:
            b_on_a = max(b_on_a, type_.damage_multiplier(*mon_a.types))
    # Our performance metric is the different between the two
    return a_on_b - b_on_a
```

We can now use it in our `teampreview` method:

```
def teampreview(self, battle):
    mon_performance = {}

    # For each of our pokemons
    for i, mon in enumerate(battle.team.values()):
        # We store their average performance against the opponent team
        mon_performance[i] = np.mean([
            teampreview_performance(mon, opp)
            for opp in battle.opponent_team.values()
        ])

    # We sort our mons by performance
    ordered_mons = sorted(mon_performance, key = lambda k: -mon_performance[k])

    # We start with the one we consider best overall
    # We use i + 1 as python indexes start from 0
    #  but showdown's indexes start from 1
    return "/team " + ''.join([str(i + 1) for i in ordered_mons])
```

This method sends our pokemons ordered by their average estimated performance against the opponent team.

Specifying a team

To specify a team, you have two main options: you can either provide a `str` describing your team, or a `Teambuilder` object. This example will focus on the first option; if you want to learn more about using teambuilders, please refer to *Creating a custom teambuilder* and *The teambuilder object and related classes*.

The easiest way to specify a team in `poke-env` is to copy-paste a showdown team. You can use showdown's teambuilder and export it directly.

Alternatively, you can use showdown's packed formats, which correspond to the actual string sent by the showdown client to the server.

Here is an example team, both in showdown and packed formats:

Packed format

Showdown format

Attributing a team to an agent

To attribute a team to an agent, you need to pass a `team` argument to the agent's constructor. This argument can either be a `Teambuilder` object, or the string describing your team. Here is an example:

```
team_1 = """
Goodra (M) @ Assault Vest
Ability: Sap Sipper
EVs: 248 HP / 252 SpA / 8 Spe
Modest Nature
IVs: 0 Atk
- Dragon Pulse
- Flamethrower
- Sludge Wave
- Thunderbolt
```

(continues on next page)

(continued from previous page)

```
Sylveon (M) @ Leftovers
Ability: Pixilate
EVs: 248 HP / 244 Def / 16 SpD
Calm Nature
IVs: 0 Atk
- Hyper Voice
- Mystical Fire
- Protect
- Wish

Cinderace (M) @ Life Orb
Ability: Blaze
EVs: 252 Atk / 4 SpD / 252 Spe
Jolly Nature
- Pyro Ball
- Sucker Punch
- U-turn
- High Jump Kick

Toxtricity (M) @ Throat Spray
Ability: Punk Rock
EVs: 4 Atk / 252 SpA / 252 Spe
Rash Nature
- Overdrive
- Boomburst
- Shift Gear
- Fire Punch

Seismitoad (M) @ Leftovers
Ability: Water Absorb
EVs: 252 HP / 252 Def / 4 SpD
Relaxed Nature
- Stealth Rock
- Scald
- Earthquake
- Toxic

Corviknight (M) @ Leftovers
Ability: Pressure
EVs: 248 HP / 80 SpD / 180 Spe
Impish Nature
- Defog
- Brave Bird
- Roost
- U-turn
"""
team_2 = """
Togekiss @ Leftovers
Ability: Serene Grace
EVs: 248 HP / 8 SpA / 252 Spe
Timid Nature
IVs: 0 Atk
- Air Slash
- Nasty Plot
- Substitute
- Thunder Wave
```

(continues on next page)

(continued from previous page)

```
Galvantula @ Focus Sash
Ability: Compound Eyes
EVs: 252 SpA / 4 SpD / 252 Spe
Timid Nature
IVs: 0 Atk
- Sticky Web
- Thunder Wave
- Thunder
- Energy Ball

Cloyster @ King's Rock
Ability: Skill Link
EVs: 252 Atk / 4 SpD / 252 Spe
Adamant Nature
- Icicle Spear
- Rock Blast
- Ice Shard
- Shell Smash

Sandaconda @ Focus Sash
Ability: Sand Spit
EVs: 252 Atk / 4 SpD / 252 Spe
Jolly Nature
- Stealth Rock
- Glare
- Earthquake
- Rock Tomb

Excadrill @ Focus Sash
Ability: Sand Rush
EVs: 252 Atk / 4 SpD / 252 Spe
Adamant Nature
- Iron Head
- Rock Slide
- Earthquake
- Rapid Spin

Cinccino @ King's Rock
Ability: Skill Link
EVs: 252 Atk / 4 Def / 252 Spe
Jolly Nature
- Bullet Seed
- Knock Off
- Rock Blast
- Tail Slap
"""

# We create two players.
random_player = RandomPlayer(
    battle_format="gen8ou",
    team=team_1,
    max_concurrent_battles=10,
)
max_damage_player = MaxDamagePlayer(
    battle_format="gen8ou",
    team=team_2,
```

(continues on next page)

(continued from previous page)

```
    max_concurrent_battles=10,
)
```

Warning: Parsing team can be sensitive to case or spaces. If you encounter errors, make sure that the string your are passing does not contain any unexpected characters.

Warning: Team parsing is a recent feature, and may contain unexpected bugs. If you encounter one, please do not hesitate to open an issue.

Running and testing our agent

We can now test our agent. To do so, we can use the `cross_evaluate` function from `poke_env.player.utils` or the `battle_against` method from `Player`.

```
import asyncio
import numpy as np

from poke_env.player.player import Player
from poke_env.player.random_player import RandomPlayer


class MaxDamagePlayer(Player):
    def choose_move(self, battle):
        # If the player can attack, it will
        if battle.available_moves:
            # Finds the best move among available ones
            best_move = max(battle.available_moves, key=lambda move: move.base_power)
            return self.create_order(best_move)

        # If no attack is available, a random switch will be made
        else:
            return self.choose_random_move(battle)

    def teampreview(self, battle):
        mon_performance = {}

        # For each of our pokemons
        for i, mon in enumerate(battle.team.values()):
            # We store their average performance against the opponent team
            mon_performance[i] = np.mean(
                [
                    teampreview_performance(mon, opp)
                    for opp in battle.opponent_team.values()
                ]
            )

        # We sort our mons by performance
        ordered_mons = sorted(mon_performance, key=lambda k: -mon_performance[k])

        # We start with the one we consider best overall
        # We use i + 1 as python indexes start from 0
```

(continues on next page)

(continued from previous page)

```
        #  but showdown's indexes start from 1
        return "/team " + "".join([str(i + 1) for i in ordered_mons])


def teampreview_performance(mon_a, mon_b):
    # We evaluate the performance on mon_a against mon_b as its type advantage
    a_on_b = b_on_a = -np.inf
    for type_ in mon_a.types:
        if type_:
            a_on_b = max(a_on_b, type_.damage_multiplier(*mon_b.types))
    # We do the same for mon_b over mon_a
    for type_ in mon_b.types:
        if type_:
            b_on_a = max(b_on_a, type_.damage_multiplier(*mon_a.types))
    # Our performance metric is the different between the two
    return a_on_b - b_on_a


async def main():
    team_1 = """
Goodra (M) @ Assault Vest
Ability: Sap Sipper
EVs: 248 HP / 252 SpA / 8 Spe
Modest Nature
IVs: 0 Atk
- Dragon Pulse
- Flamethrower
- Sludge Wave
- Thunderbolt

Sylveon (M) @ Leftovers
Ability: Pixilate
EVs: 248 HP / 244 Def / 16 SpD
Calm Nature
IVs: 0 Atk
- Hyper Voice
- Mystical Fire
- Protect
- Wish

Cinderace (M) @ Life Orb
Ability: Blaze
EVs: 252 Atk / 4 SpD / 252 Spe
Jolly Nature
- Pyro Ball
- Sucker Punch
- U-turn
- High Jump Kick

Toxtricity (M) @ Throat Spray
Ability: Punk Rock
EVs: 4 Atk / 252 SpA / 252 Spe
Rash Nature
- Overdrive
- Boomburst
- Shift Gear
- Fire Punch
```

(continues on next page)

(continued from previous page)

```
Seismitoad (M) @ Leftovers
Ability: Water Absorb
EVs: 252 HP / 252 Def / 4 SpD
Relaxed Nature
- Stealth Rock
- Scald
- Earthquake
- Toxic

Corviknight (M) @ Leftovers
Ability: Pressure
EVs: 248 HP / 80 SpD / 180 Spe
Impish Nature
- Defog
- Brave Bird
- Roost
- U-turn
"""
    team_2 = """
Togekiss @ Leftovers
Ability: Serene Grace
EVs: 248 HP / 8 SpA / 252 Spe
Timid Nature
IVs: 0 Atk
- Air Slash
- Nasty Plot
- Substitute
- Thunder Wave

Galvantula @ Focus Sash
Ability: Compound Eyes
EVs: 252 SpA / 4 SpD / 252 Spe
Timid Nature
IVs: 0 Atk
- Sticky Web
- Thunder Wave
- Thunder
- Energy Ball

Cloyster @ King's Rock
Ability: Skill Link
EVs: 252 Atk / 4 SpD / 252 Spe
Adamant Nature
- Icicle Spear
- Rock Blast
- Ice Shard
- Shell Smash

Sandaconda @ Focus Sash
Ability: Sand Spit
EVs: 252 Atk / 4 SpD / 252 Spe
Jolly Nature
- Stealth Rock
- Glare
- Earthquake
- Rock Tomb
```

(continues on next page)

(continued from previous page)

```
Excadrill @ Focus Sash
Ability: Sand Rush
EVs: 252 Atk / 4 SpD / 252 Spe
Adamant Nature
- Iron Head
- Rock Slide
- Earthquake
- Rapid Spin

Cinccino @ King's Rock
Ability: Skill Link
EVs: 252 Atk / 4 Def / 252 Spe
Jolly Nature
- Bullet Seed
- Knock Off
- Rock Blast
- Tail Slap
"""

# We create two players.
random_player = RandomPlayer(
    battle_format="gen8ou",
    team=team_1,
    max_concurrent_battles=10,
)
max_damage_player = MaxDamagePlayer(
    battle_format="gen8ou",
    team=team_2,
    max_concurrent_battles=10,
)

# Now, let's evaluate our player
await max_damage_player.battle_against(random_player, n_battles = 100)

print(
    "Max damage player won %d / 100 battles"
    % max_damage_player.n_won_battles
)


if __name__ == "__main__":
    asyncio.get_event_loop().run_until_complete(main())
```

Running it should take a couple of seconds and print something similar to this:

```
Max damage player won 99 / 100 battles
```

If you want to use Reinforcement Learning, take a look at *Reinforcement learning with the OpenAI Gym wrapper* example.

If you want to create a custom teambuilder, take a look at *Creating a custom teambuilder*.

1.2.5 Creating a custom teambuilder

The corresponding complete source code can be found here.

Note: A similar example using gen 7 mechanics is available here.

In *Adapting the max player to gen 8 OU and managing team preview*, we chose a team by passing a *str* containing the team we want to use as a showdown format team.

However, we might want to use different teams in different battles with the same agent, and use more complex mechanisms to generate and select teams. `Teambuilder` objects are meant for specifying teams in such a custom fashion. This example demonstrates how to build a simple custom `Teambuilder`: we will specify a pool of teams, and each game will be played using a team randomly selected from the pool.

Creating a custom `Teambuilder`

Class definition

`Teambuilder` objects need to implement one method, `yield_team`, which will be called before each battle starts to define the team to use. This method must return a showdown packed-formatted string. In this example, we will use built-in helper functions to simplify this process.

Our custom `Teambuilder` will be initialized with a list of showdown formatted teams, and will use one of these team randomly for each battle.

We therefore need to convert showdown formatted teams to the packed-formatted string required by showdown's protocol. We can do that in two steps:

- Convert showdown formatted teams to lists of `TeambuilderPokemon` objects. These objects are used internally by `poke-env` to describe pokemons used in a team in a flexible way. You can read more about them in *The teambuilder object and related classes*. This can be accomplished with `Teambuilder`'s `parse_showdown_team` method.
- Convert this list of `TeambuilderPokemon` objects into the required formatted string. This can be achieved with `Teambuilder`'s `join_team` method.

All in all, we get the following `Teambuilder`:

```
import numpy as np

from poke_env.teambuilder.teambuilder import Teambuilder


class RandomTeamFromPool(Teambuilder):
    def __init__(self, teams):
        self.teams = [self.join_team(self.parse_showdown_team(team)) for team in␣
↪teams]

    def yield_team(self):
        return np.random.choice(self.teams)
```

Instanciation

We can instantiate it as follows:

```
team_1 = """
Goodra (M) @ Assault Vest
Ability: Sap Sipper
```

(continues on next page)

(continued from previous page)

```
EVs: 248 HP / 252 SpA / 8 Spe
Modest Nature
IVs: 0 Atk
- Dragon Pulse
- Flamethrower
- Sludge Wave
- Thunderbolt

Sylveon (M) @ Leftovers
Ability: Pixilate
EVs: 248 HP / 244 Def / 16 SpD
Calm Nature
IVs: 0 Atk
- Hyper Voice
- Mystical Fire
- Protect
- Wish

Cinderace (M) @ Life Orb
Ability: Blaze
EVs: 252 Atk / 4 SpD / 252 Spe
Jolly Nature
- Pyro Ball
- Sucker Punch
- U-turn
- High Jump Kick

Toxtricity (M) @ Throat Spray
Ability: Punk Rock
EVs: 4 Atk / 252 SpA / 252 Spe
Rash Nature
- Overdrive
- Boomburst
- Shift Gear
- Fire Punch

Seismitoad (M) @ Leftovers
Ability: Water Absorb
EVs: 252 HP / 252 Def / 4 SpD
Relaxed Nature
- Stealth Rock
- Scald
- Earthquake
- Toxic

Corviknight (M) @ Leftovers
Ability: Pressure
EVs: 248 HP / 80 SpD / 180 Spe
Impish Nature
- Defog
- Brave Bird
- Roost
- U-turn
"""

team_2 = """
Togekiss @ Leftovers
```

(continues on next page)

(continued from previous page)

```
Ability: Serene Grace
EVs: 248 HP / 8 SpA / 252 Spe
Timid Nature
IVs: 0 Atk
- Air Slash
- Nasty Plot
- Substitute
- Thunder Wave

Galvantula @ Focus Sash
Ability: Compound Eyes
EVs: 252 SpA / 4 SpD / 252 Spe
Timid Nature
IVs: 0 Atk
- Sticky Web
- Thunder Wave
- Thunder
- Energy Ball

Cloyster @ King's Rock
Ability: Skill Link
EVs: 252 Atk / 4 SpD / 252 Spe
Adamant Nature
- Icicle Spear
- Rock Blast
- Ice Shard
- Shell Smash

Sandaconda @ Focus Sash
Ability: Sand Spit
EVs: 252 Atk / 4 SpD / 252 Spe
Jolly Nature
- Stealth Rock
- Glare
- Earthquake
- Rock Tomb

Excadrill @ Focus Sash
Ability: Sand Rush
EVs: 252 Atk / 4 SpD / 252 Spe
Adamant Nature
- Iron Head
- Rock Slide
- Earthquake
- Rapid Spin

Cinccino @ King's Rock
Ability: Skill Link
EVs: 252 Atk / 4 Def / 252 Spe
Jolly Nature
- Bullet Seed
- Knock Off
- Rock Blast
- Tail Slap
"""

custom_builder = RandomTeamFromPool([team_1, team_2])
```

Our `custom_builder` can now be used! To use a `Teambuilder` with a given `Player`, just pass it in its constructor, with the `team` keyword.

```
from poke_env.player.random_player import RandomPlayer

player_1 = RandomPlayer(
    battle_format="gen8ou",
    team=custom_builder,
    max_concurrent_battles=10,
)
player_2 = RandomPlayer(
    battle_format="gen8ou",
    team=custom_builder,
    max_concurrent_battles=10,
)
```

Launching battles

Now that we have two players with custom teambuilders, we can make them battle!

```
await player_1.battle_against(player_2, n_battles=5)
```

The complete example looks like that:

```
# -*- coding: utf-8 -*-
import asyncio
import numpy as np

from poke_env.player.random_player import RandomPlayer
from poke_env.teambuilder.teambuilder import Teambuilder


class RandomTeamFromPool(Teambuilder):
    def __init__(self, teams):
        self.teams = [self.join_team(self.parse_showdown_team(team)) for team in␣
↪teams]

    def yield_team(self):
        return np.random.choice(self.teams)


team_1 = """
Goodra (M) @ Assault Vest
Ability: Sap Sipper
EVs: 248 HP / 252 SpA / 8 Spe
Modest Nature
IVs: 0 Atk
- Dragon Pulse
- Flamethrower
- Sludge Wave
- Thunderbolt

Sylveon (M) @ Leftovers
Ability: Pixilate
EVs: 248 HP / 244 Def / 16 SpD
Calm Nature
```

(continues on next page)

(continued from previous page)

```
IVs: 0 Atk
- Hyper Voice
- Mystical Fire
- Protect
- Wish

Cinderace (M) @ Life Orb
Ability: Blaze
EVs: 252 Atk / 4 SpD / 252 Spe
Jolly Nature
- Pyro Ball
- Sucker Punch
- U-turn
- High Jump Kick

Toxtricity (M) @ Throat Spray
Ability: Punk Rock
EVs: 4 Atk / 252 SpA / 252 Spe
Rash Nature
- Overdrive
- Boomburst
- Shift Gear
- Fire Punch

Seismitoad (M) @ Leftovers
Ability: Water Absorb
EVs: 252 HP / 252 Def / 4 SpD
Relaxed Nature
- Stealth Rock
- Scald
- Earthquake
- Toxic

Corviknight (M) @ Leftovers
Ability: Pressure
EVs: 248 HP / 80 SpD / 180 Spe
Impish Nature
- Defog
- Brave Bird
- Roost
- U-turn
"""

team_2 = """
Togekiss @ Leftovers
Ability: Serene Grace
EVs: 248 HP / 8 SpA / 252 Spe
Timid Nature
IVs: 0 Atk
- Air Slash
- Nasty Plot
- Substitute
- Thunder Wave

Galvantula @ Focus Sash
Ability: Compound Eyes
EVs: 252 SpA / 4 SpD / 252 Spe
```

(continues on next page)

(continued from previous page)

```
Timid Nature
IVs: 0 Atk
- Sticky Web
- Thunder Wave
- Thunder
- Energy Ball

Cloyster @ King's Rock
Ability: Skill Link
EVs: 252 Atk / 4 SpD / 252 Spe
Adamant Nature
- Icicle Spear
- Rock Blast
- Ice Shard
- Shell Smash

Sandaconda @ Focus Sash
Ability: Sand Spit
EVs: 252 Atk / 4 SpD / 252 Spe
Jolly Nature
- Stealth Rock
- Glare
- Earthquake
- Rock Tomb

Excadrill @ Focus Sash
Ability: Sand Rush
EVs: 252 Atk / 4 SpD / 252 Spe
Adamant Nature
- Iron Head
- Rock Slide
- Earthquake
- Rapid Spin

Cinccino @ King's Rock
Ability: Skill Link
EVs: 252 Atk / 4 Def / 252 Spe
Jolly Nature
- Bullet Seed
- Knock Off
- Rock Blast
- Tail Slap
"""

custom_builder = RandomTeamFromPool([team_1, team_2])


async def main():
    # We create two players
    player_1 = RandomPlayer(
        battle_format="gen8ou",
        team=custom_builder,
        max_concurrent_battles=10,
    )
    player_2 = RandomPlayer(
        battle_format="gen8ou",
        team=custom_builder,
```

(continues on next page)

(continued from previous page)

```
        max_concurrent_battles=10,
    )

    await player_1.battle_against(player_2, n_battles=5)


if __name__ == "__main__":
    asyncio.get_event_loop().run_until_complete(main())
```

1.2.6 Connecting to showdown and challenging humans

The corresponding complete source code can be found here.

The goal of this example is to demonstrate how to run an agent on showdown, and how to challenge human players.

Connecting your agent to showdown

To connect an agent to a showdown server hosted online, you must specify a matching server configuration.

A configuration pointing towards play.pokemonshowdown.com is available in `poke_env.server_configuration` and can be used directly. To specify a different server, see *Configuring a showdown server*.

To connect to play.pokemonshowdown.com, you also need an account for your agent to use. The following snippets assumes that the account `bot_username` exists, and can be accessed with `bot_password`.

```
from poke_env.player.random_player import RandomPlayer
from poke_env.player_configuration import PlayerConfiguration
from poke_env.server_configuration import ShowdownServerConfiguration

# We create a random player
player = RandomPlayer(
    player_configuration=PlayerConfiguration("bot_username", "bot_password"),
    server_configuration=ShowdownServerConfiguration,
)
```

Challenging a human player

Now that your agent is configured to access showdown, you can use it to challenge any specific user connected on showdown. To do so, you just need their username. The following snippet will make your agent challenge user `your_username` for one battle.

```
await player.send_challenges("your_username", n_challenges=1)
```

Accepting challenges from human players

You can use the `accept_challenges` method to automatically accept challenges from a player. To do so, run:

```
# Replace opp_username with None to accept challenges from any player
await player.accept_challenges('opp_username', 1)
```

Passing `None` instead of a username will make the agent accept challenges from any player.

```
# Replace opp_username with None to accept challenges from any player
await player.accept_challenges('opp_username', 1)
```

Playing on the ladder

Finally, you can use the `ladder` method to play games on the ladder.

```
# Play five games on the ladder
await player.ladder(5)
```

After playing games on the ladder, you may receive rating information. You can access them with the `Battle.rating` and `Battle.opponent_rating` methods:

```
# Print the rating of the player and its opponent after each battle
for battle in player.battles.values():
    print(battle.rating, battle.opponent_rating)
```

A complete example source code is:

```
# -*- coding: utf-8 -*-
import asyncio

from poke_env.player.random_player import RandomPlayer
from poke_env.player_configuration import PlayerConfiguration
from poke_env.server_configuration import ShowdownServerConfiguration


async def main():
    # We create a random player
    player = RandomPlayer(
        player_configuration=PlayerConfiguration("bot_username", "bot_password")
        server_configuration=ShowdownServerConfiguration,
    )

    # Sending challenges to 'your_username'
    await player.send_challenges("your_username", n_challenges=1)

    # Accepting one challenge from any user
    await player.accept_challenges(None, 1)

    # Accepting three challenges from 'your_username'
    await player.accept_challenges('your_username', 3)

    # Playing 5 games on the ladder
    await player.ladder(5)

    # Print the rating of the player and its opponent after each battle
    for battle in player.battles.values():
        print(battle.rating, battle.opponent_rating)


if __name__ == "__main__":
    asyncio.get_event_loop().run_until_complete(main())
```

1.3 Module documentation

1.3.1 Battle objects

class poke_env.environment.abstract_battle.**AbstractBattle**(*battle_tag: str, username: str, logger: logging.Logger*)

Bases: abc.ABC

MESSAGES_TO_IGNORE = {'-anim', '-block', '-burst', '-center', '-combine', '-crit', '-f

POKEMON_CLASS

alias of *poke_env.environment.pokemon.Gen8Pokemon*

active_pokemon

all_active_pokemons

available_moves

available_switches

battle_tag

Returns The battle identifier.

Return type str

can_mega_evolve

can_z_move

dynamax_turns_left

Returns How many turns of dynamax are left. None if dynamax is not active

Return type int, optional

end_turn(*turn: int*) → None

fields

Returns A dict mapping fields to the turn they have been activated.

Return type Dict[*Field*, int]

finished

Returns A boolean indicating whether the battle is finished.

Return type Optional[bool]

force_switch

get_pokemon(*identifier: str, force_self_team: bool = False, details: str = '', request: Optional[dict] = None*) → poke_env.environment.pokemon.Pokemon

Returns the Pokemon object corresponding to given identifier. Can force to return object from the player's team if force_self_team is True. If the Pokemon object does not exist, it will be created. Details can be given, which is necessary to initialize alternate forms (eg. alolan pokemons) properly.

Parameters

- **identifier** (*str*) – The identifier to use to retrieve the pokemon.
- **force_self_team** (*bool, optional, defaults to False*) – Wheter to force returning a Pokemon from the player's team. Defaults to False.

- **details** (*str, optional*) – Detailled information about the pokemon. Defaults to ''.

Returns The corresponding pokemon object.

Return type *Pokemon*

Raises **AssertionError** – If the team has too many pokemons, as determined by the team-size component of battle initialisation.

logger

lost

Returns If the battle is finished, a boolean indicating whether the battle is lost. Otherwise None.

Return type Optional[bool]

max_team_size

Returns The maximum acceptable size of the team to return in teampreview, if applicable.

Return type int, optional

maybe_trapped

move_on_next_request

Returns Wheter the next received request should yield a move order directly. This can happen when a switch is forced, or an error is encountered.

Return type bool

opponent_active_pokemon

opponent_can_dynamax

opponent_dynamax_turns_left

Returns How many turns of dynamax are left for the opponent's pokemon. None if dynamax is not active

Return type Optional[int]

opponent_rating

Opponent's rating after the end of the battle, if it was received.

Returns The opponent's rating after the end of the battle.

Return type int, optional

opponent_role

Returns Opponent's role in given battle. p1/p2

Return type str, optional

opponent_side_conditions

Returns

The opponent's side conditions. Keys are SideCondition objects, values are: - the number of layers of the side condition is the side condition is

stackable

- the turn where the SideCondition was setup otherwise

Return type Dict[*SideCondition*, int]

opponent_team
During teampreview, keys are not definitive: please rely on values.

Returns The opponent's team. Keys are identifiers, values are pokemon objects.

Return type Dict[str, *Pokemon*]

opponent_username

Returns The opponent's username, or None if unknown.

Return type str, optional.

player_role

Returns Player's role in given battle. p1/p2

Return type str, optional

player_username

Returns The player's username.

Return type str

players

Returns The pair of players' usernames.

Return type Tuple[str, str]

rating
Player's rating after the end of the battle, if it was received.

Returns The player's rating after the end of the battle.

Return type int, optional

rqid
Should not be used.

Returns The last request's rqid.

Return type Tuple[str, str]

side_conditions

Returns

The player's side conditions. Keys are SideCondition objects, values are: - the number of layers of the side condition is the side condition is

stackable

- the turn where the SideCondition was setup otherwise

Return type Dict[*SideCondition*, int]

team

Returns The player's team. Keys are identifiers, values are pokemon objects.

Return type Dict[str, *Pokemon*]

team_size

Returns The number of Pokemon in the player's team.

Return type int

teampreview

Returns Wheter the battle is awaiting a teampreview order.

Return type bool

trapped

turn

Returns The current battle turn.

Return type int

weather

Returns The battle's weather or None if no weather is active.

Return type Optional[*Weather*]

won

Returns If the battle is finished, a boolean indicating whether the battle is won. Otherwise None.

Return type Optional[bool]

class poke_env.environment.battle.**Battle**(*battle_tag: str*, *username: str*, *logger: logging.Logger*)

Bases: *poke_env.environment.abstract_battle.AbstractBattle*

active_pokemon

Returns The active pokemon

Return type Optional[*Pokemon*]

all_active_pokemons

Returns A list containing all active pokemons and/or Nones.

Return type List[Optional[*Pokemon*]]

available_moves

Returns The list of moves the player can use during the current move request.

Return type List[*Move*]

available_switches

Returns The list of switches the player can do during the current move request.

Return type List[*Pokemon*]

can_dynamax

Returns Wheter of not the current active pokemon can dynamax

Return type bool

can_mega_evolve

Returns Wheter of not the current active pokemon can mega evolve.

Return type bool

can_z_move

Returns Wheter of not the current active pokemon can z-move.

Return type bool

force_switch

Returns A boolean indicating whether the active pokemon is forced to switch out.

Return type Optional[bool]

static from_format(*format_: str*, *battle_tag: str*, *username: str*, *logger: logging.Logger*) → poke_env.environment.abstract_battle.AbstractBattle

Returns A GenXBattle instance, depending on the format received

Return type *Battle*

maybe_trapped

Returns A boolean indicating whether the active pokemon is maybe trapped by the opponent.

Return type bool

opponent_active_pokemon

Returns The opponent active pokemon

Return type *Pokemon*

opponent_can_dynamax

Returns Wheter of not opponent's current active pokemon can dynamax

Return type bool

trapped

Returns A boolean indicating whether the active pokemon is trapped, either by the opponent or as a side effect of one your moves.

Return type bool

class poke_env.environment.battle.**Gen4Battle**(*battle_tag: str*, *username: str*, *logger: logging.Logger*)

Bases: *poke_env.environment.battle.Battle*

POKEMON_CLASS

alias of *poke_env.environment.pokemon.Gen4Pokemon*

class poke_env.environment.battle.**Gen5Battle**(*battle_tag: str*, *username: str*, *logger: logging.Logger*)

Bases: *poke_env.environment.battle.Battle*

POKEMON_CLASS

alias of *poke_env.environment.pokemon.Gen5Pokemon*

class poke_env.environment.battle.**Gen6Battle**(*battle_tag: str*, *username: str*, *logger: logging.Logger*)

Bases: *poke_env.environment.battle.Battle*

POKEMON_CLASS

alias of *poke_env.environment.pokemon.Gen6Pokemon*

class poke_env.environment.battle.**Gen7Battle**(*battle_tag: str*, *username: str*, *logger: logging.Logger*)

Bases: *poke_env.environment.battle.Battle*

POKEMON_CLASS

alias of *poke_env.environment.pokemon.Gen7Pokemon*

class poke_env.environment.battle.**Gen8Battle**(*battle_tag: str*, *username: str*, *logger: logging.Logger*)

Bases: *poke_env.environment.battle.Battle*

POKEMON_CLASS

alias of *poke_env.environment.pokemon.Gen8Pokemon*

class poke_env.environment.double_battle.**DoubleBattle**(*battle_tag: str*, *username: str*, *logger: logging.Logger*)

Bases: *poke_env.environment.abstract_battle.AbstractBattle*

EMPTY_TARGET_POSITION = 0

OPPONENT_1_POSITION = 1

OPPONENT_2_POSITION = 2

POKEMON_1_POSITION = -1

POKEMON_2_POSITION = -2

active_pokemon

Returns The active pokemon, always at least one is not None

Return type List[Optional[*Pokemon*]]

all_active_pokemons

Returns A list containing all active pokemons and/or Nones.

Return type List[Optional[*Pokemon*]]

available_moves

Returns A list of two lists of moves the player can use during the current move request for each Pokemon.

Return type List[List[*Move*]]

available_switches

Returns The list of two lists of switches the player can do during the current move request for each active pokemon

Return type List[List[*Pokemon*]]

can_dynamax

Returns Wheter of not the current active pokemon can dynamax

Return type List[bool]

can_mega_evolve

Returns Whether of not either current active pokemon can mega evolve.

Return type List[bool]

can_z_move

Returns Wheter of not the current active pokemon can z-move.

Return type List[bool]

force_switch

Returns A boolean indicating whether the active pokemon is forced to switch out.

Return type List[bool]

get_possible_showdown_targets(*move: poke_env.environment.move.Move*, *pokemon: poke_env.environment.pokemon.Pokemon*, *dynamax=False*) → List[int]

Given move of an ALLY Pokemon, returns a list of possible Pokemon Showdown targets for it. This is smart enough so that it figures whether the Pokemon is already dynamaxed.

Parameters

- **move** (Move) – Move instance for which possible targets should be returned
- **dynamax** – whether given move also STARTS dynamax for its user

Returns a list of integers indicating Pokemon Showdown targets: -1, -2, 1, 2 or self.EMPTY_TARGET_POSITION that indicates "no target"

Return type List[int]

maybe_trapped

Returns A boolean indicating whether either active pokemon is maybe trapped by the opponent.

Return type List[bool]

opponent_active_pokemon

Returns The opponent active pokemon, always at least one is not None

Return type List[Optional[*Pokemon*]]

opponent_can_dynamax

Returns Wheter of not opponent's current active pokemon can dynamax

Return type List[bool]

trapped

Returns A boolean indicating whether either active pokemon is trapped by the opponent.

Return type List[bool]

1.3.2 The move object

class poke_env.environment.move.**DynamaxMove**(*parent: poke_env.environment.move.Move*)

Bases: *poke_env.environment.move.Move*

BOOSTS_MAP = {<PokemonType.BUG: 1>: {'spa': -1}, <PokemonType.DARK: 2>: {'spd': -1

SELF_BOOSTS_MAP = {<PokemonType.FIGHTING: 6>: {'atk': 1}, <PokemonType.FLYING: 8>:

TERRAIN_MAP = {<PokemonType.ELECTRIC: 4>: <Field.ELECTRIC_TERRAIN: 2>, <PokemonType.F

WEATHER_MAP = {<PokemonType.FIRE: 7>: <Weather.SUNNYDAY: 8>, <PokemonType.ICE: 12>:

accuracy

Returns The move's accuracy (0 to 1 scale).

Return type float

base_power

Returns The move's base power.

Return type int

boosts

Returns Boosts conferred to the target by using the move.

Return type Optional[Dict[str, float]]

breaks_protect

Returns Whether the move breaks proect-like defenses.

Return type bool

crit_ratio

Returns The move's crit ratio. If the move is guaranteed to crit, returns 6.

Return type

damage

Returns The move's fix damages. Can be an int or 'level' for moves such as Seismic Toss.

Return type Union[int, str]

defensive_category

Returns Move's defender category.

Return type *MoveCategory*

expected_hits

Returns Expected number of hits, between 1 and 5. Equal to n_hits if n_hits is constant.

Return type float

force_switch

Returns Whether this move forces switches.

Return type bool

heal

Returns Proportion of the user's HP recovered.

Return type float

is_protect_counter

Returns Wheter this move increments a mon's protect counter.

Return type int

is_protect_move

Returns Wheter this move is a protect-like move.

Return type int

n_hit

Returns How many hits this move lands. Tuple of the form (min, max).

Return type Tuple

priority

Returns Move priority.

Return type int

recoil

Returns Proportion of the move's damage inflicted as recoil.

Return type float

self_boost

Returns Boosts applied to the move's user.

Return type Dict[str, int]

status

Returns The status inflicted by the move.

Return type Optional[*Status*]

terrain

Returns Terrain started by the move.

Return type Optional[*Field*]

weather

Returns Weather started by the move.

Return type Optional[*Weather*]

class poke_env.environment.move.**EmptyMove**(*move_id*)
Bases: *poke_env.environment.move.Move*

class poke_env.environment.move.**Gen4Move**(*move: str = ''*, *move_id: Optional[str] = None*)
Bases: *poke_env.environment.move.Move*

class poke_env.environment.move.**Gen5Move**(*move: str = ''*, *move_id: Optional[str] = None*)
Bases: *poke_env.environment.move.Move*

class poke_env.environment.move.**Gen6Move**(*move: str = ''*, *move_id: Optional[str] = None*)
Bases: *poke_env.environment.move.Move*

class poke_env.environment.move.**Gen7Move**(*move: str = ''*, *move_id: Optional[str] = None*)
Bases: *poke_env.environment.move.Move*

class poke_env.environment.move.**Gen8Move**(*move: str = ''*, *move_id: Optional[str] = None*)
Bases: *poke_env.environment.move.Move*

class poke_env.environment.move.**Move**(*move: str = ''*, *move_id: Optional[str] = None*)
Bases: object

accuracy

Returns The move's accuracy (0 to 1 scale).

Return type float

base_power

Returns The move's base power.

Return type int

boosts

Returns Boosts conferred to the target by using the move.

Return type Optional[Dict[str, float]]

breaks_protect

Returns Whether the move breaks proect-like defenses.

Return type bool

can_z_move

Returns Wheter there exist a z-move version of this move.

Return type bool

category

Returns The move category.

Return type *MoveCategory*

crit_ratio

Returns The move's crit ratio. If the move is guaranteed to crit, returns 6.

Return type

current_pp

Returns Remaining PP.

Return type int

damage

Returns The move's fix damages. Can be an int or 'level' for moves such as Seismic Toss.

Return type Union[int, str]

deduced_target

Returns Move deduced target, based on Move.target and showdown's request messages.

Return type str, optional

defensive_category

Returns Move's defender category.

Return type *MoveCategory*

drain

Returns Ratio of HP of inflicted damages, between 0 and 1.

Return type float

dynamaxed

Returns The dynamaxed version of the move.

Return type *DynamaxMove*

entry

Should not be used directly.

Returns The data entry corresponding to the move

Return type dict

expected_hits

Returns Expected number of hits, between 1 and 5. Equal to n_hits if n_hits is constant.

Return type float

flags

This property is not well defined, and may be missing some information. If you need more information on some flag, please open an issue in the project.

Returns Flags associated with this move. These can come from the data or be custom.

Return type Set[str]

force_switch

Returns Whether this move forces switches.

Return type bool

heal

Returns Proportion of the user's HP recovered.

Return type float

id

Returns Move id.

Return type str

ignore_ability

Returns Whether the move ignore its target's ability.

Return type bool

ignore_defensive

Returns Whether the opponent's stat boosts are ignored.

Return type bool

ignore_evasion

Returns Wheter the opponent's evasion is ignored.

Return type bool

ignore_immunity

Returns Whether the opponent's immunity is ignored, or a list of ignored immunities.

Return type bool or set of Types

is_empty

Returns Whether the move is an empty move.

Return type bool

static is_id_z(*id_*) → bool

static is_max_move(*id_*) → bool

is_protect_counter

Returns Wheter this move increments a mon's protect counter.

Return type int

is_protect_move

Returns Wheter this move is a protect-like move.

Return type int

is_side_protect_move

Returns Wheter this move is a side-protect move.

Return type int

is_z

Returns Whether the move is a z move.

Return type bool

max_pp

Returns The move's max pp.

Return type int

n_hit

Returns How many hits this move lands. Tuple of the form (min, max).

Return type Tuple

no_pp_boosts

Returns Whether the move uses PPs.

Return type bool

non_ghost_target

Returns True for curse.

Return type bool

priority

Returns Move priority.

Return type int

pseudo_weather

Returns Pseudo-weather activated by this move.

Return type str

recoil

Returns Proportion of the move's damage inflicted as recoil.

Return type float

request_target

Returns Target information sent by showdown in a request message, if any.

Return type str, optional

retrieve_id

Retrieve the id of a move based on its full name.

Parameters **move_name** (*str*) – The string to convert into a move id.

Returns The corresponding move id.

Return type str

secondary

Returns Secondary effects. At this point, the precise content of this property is not too clear.

Return type Optional[Dict]

self_boost

Returns Boosts applied to the move's user.

Return type Dict[str, int]

self_destruct

Returns Move's self destruct consequences.

Return type Optional[str]

self_switch

Returns What kind of self swtich this move implies for the user.

Return type Optional[str]

should_be_stored

side_condition

Returns Side condition inflicted by the move.

Return type Optional[str]

sleep_usable

Returns Whether the move can be user by a sleeping pokemon.

Return type bool

slot_condition

Returns Which slot condition is started by this move.

Return type Optional[str]

stalling_move

Returns Showdown classification of the move as a stalling move.

Return type bool

status

Returns The status inflicted by the move.

Return type Optional[*Status*]

steals_boosts

Returns Whether the move steals its target's boosts.

Return type bool

target

Returns

Move target. Possible targets (copied from PS codebase):

- adjacentAlly - Only relevant to Doubles or Triples, the move only targets an ally of the user.
- adjacentAllyOrSelf - The move can target the user or its ally.

- adjacentFoe - The move can target a foe, but not (in Triples) a distant foe.
- all - The move targets the field or all Pokémon at once.
- allAdjacent - The move is a spread move that also hits the user's ally.
- allAdjacentFoes - The move is a spread move.
- allies - The move affects all active Pokémon on the user's team.
- allySide - The move adds a side condition on the user's side.
- allyTeam - The move affects all unfainted Pokémon on the user's team.
- any - The move can hit any other active Pokémon, not just those adjacent.
- foeSide - The move adds a side condition on the foe's side.
- normal - The move can hit one adjacent Pokémon of your choice.
- randomNormal - The move targets an adjacent foe at random.
- scripted - The move targets the foe that damaged the user.
- self - The move affects the user of the move.

Return type str

terrain

Returns Terrain started by the move.

Return type Optional[*Field*]

thaws_target

Returns Whether the move thaws its target.

Return type bool

type

Returns Move type.

Return type *PokemonType*

use() → None

use_target_offensive

Returns Whether the move uses the target's offensive statistics.

Return type bool

volatile_status

Returns Volatile status inflicted by the move.

Return type Optional[str]

weather

Returns Weather started by the move.

Return type Optional[*Weather*]

z_move_boost

Returns Boosts associated with the z-move version of this move.

Return type Dict[str, int]

z_move_effect

Returns Effects associated with the z-move version of this move.

Return type Optional[str]

z_move_power

Returns Base power of the z-move version of this move.

Return type int

1.3.3 Other environment objects

Effect

This module defines the Effect class, which represents in-game effects.

class poke_env.environment.effect.**Effect**

Bases: enum.Enum

Enumeration, represent an effect a Pokemon can be affected by.

AFTERMATH = 3

AFTER_YOU = 2

AQUA_RING = 4

AROMATHERAPY = 5

AROMA_VEIL = 6

ATTRACT = 7

AUTOTOMIZE = 8

BAD_DREAMS = 9

BANEFUL_BUNKER = 10

BATTLE_BOND = 11

BIDE = 12

BIND = 13

BURN_UP = 14

CELEBRATE = 15

CHARGE = 16

CLAMP = 17

CONFUSION = 18

COURT_CHANGE = 19

CRAFTY_SHIELD = 20

CURSE = 21

CUSTAP_BERRY = 22

DANCER = 23

DESTINY_BOND = 24

```
DISABLE = 25
DISGUISE = 26
DOOM_DESIRE = 27
DYNAMAX = 28
EERIE_SPELL = 29
ELECTRIC_TERRAIN = 30
EMBARGO = 31
EMERGENCY_EXIT = 32
ENCORE = 33
ENDURE = 34
FAIRY_LOCK = 35
FEINT = 36
FIRE_SPIN = 37
FLASH_FIRE = 38
FLOWER_VEIL = 39
FOCUS_BAND = 40
FOCUS_ENERGY = 41
FORESIGHT = 42
FOREWARN = 43
FUTURE_SIGHT = 44
GRAVITY = 50
GRUDGE = 51
GUARD_SPLIT = 52
GULP_MISSILE = 53
G_MAX_CENTIFERNO = 45
G_MAX_CHI_STRIKE = 46
G_MAX_ONE_BLOW = 47
G_MAX_RAPID_FLOW = 48
G_MAX_SANDBLAST = 49
HEALER = 56
HEAL_BELL = 54
HEAL_BLOCK = 55
HYDRATION = 57
HYPERSPACE_FURY = 58
HYPERSPACE_HOLE = 59
ICE_FACE = 60
```

```
ILLUSION = 61
IMMUNITY = 62
IMPRISON = 63
INFESTATION = 64
INGRAIN = 65
INNARDS_OUT = <class 'enum.auto'>
INSOMNIA = 66
IRON_BARBS = 67
LASER_FOCUS = 68
LEECH_SEED = 69
LIGHTNING_ROD = 70
LIMBER = 71
LIQUID_OOZE = 72
LOCK_ON = 73
MAGMA_STORM = 74
MAGNET_RISE = 75
MAGNITUDE = 76
MAT_BLOCK = 77
MAX_GUARD = 78
MIMIC = 79
MIMICRY = 80
MIND_READER = 81
MIRACLE_EYE = 82
MIST = 83
MISTY_TERRAIN = 84
MUMMY = 85
NEUTRALIZING_GAS = 86
NIGHTMARE = 87
NO_RETREAT = 88
OBLIVIOUS = 89
OCTOLOCK = 90
OWN_TEMPO = 91
PASTEL_VEIL = 92
PERISH0 = 93
PERISH1 = 94
PERISH2 = 95
```

```
PERISH3 = 96
PHANTOM_FORCE = 97
POLTERGEIST = 98
POWDER = 99
POWER_CONSTRUCT = 100
POWER_SPLIT = 101
POWER_TRICK = 102
PROTECT = 103
PROTECTIVE_PADS = 104
PSYCHIC_TERRAIN = 105
PURSUIT = 106
QUASH = 107
QUICK_CLAW = 108
QUICK_GUARD = 109
REFLECT = 110
RIPEN = 111
ROUGH_SKIN = 112
SAFEGUARD = 113
SAFETY_GOGGLES = 114
SAND_TOMB = 115
SCREEN_CLEANER = 116
SHADOW_FORCE = 117
SHED_SKIN = 118
SKETCH = 119
SKILL_SWAP = 120
SKY_DROP = 121
SLOW_START = 122
SMACK_DOWN = 123
SNAP_TRAP = 124
SNATCH = 125
SPEED_SWAP = 126
SPITE = 127
STICKY_HOLD = 128
STICKY_WEB = 129
STOCKPILE = 130
STOCKPILE1 = 131
```

STOCKPILE2 = 132

STOCKPILE3 = 133

STORM_DRAIN = 134

STRUGGLE = 135

SUBSTITUTE = 136

SUCTION_CUPS = 137

SWEET_VEIL = 138

SYMBIOSIS = 139

SYNCHRONIZE = 140

TAR_SHOT = 141

TAUNT = 142

TELEKINESIS = 143

TELEPATHY = 144

THROAT_CHOP = 145

THUNDER_CAGE = 146

TORMENT = 147

TRAPPED = 148

TRICK = 149

TYPEADD = 150

TYPECHANGE = 151

TYPE_CHANGE = 152

UPROAR = 153

VITAL_SPIRIT = 154

WANDERING_SPIRIT = 155

WATER_BUBBLE = 156

WATER_VEIL = 157

WHIRLPOOL = 158

WIDE_GUARD = 159

WIMP_OUT = 160

WRAP = 161

YAWN = 162

breaks_protect

Returns Wheter this effects breaks protect-like states.

Return type bool

from_showdown_message = <function Effect.from_showdown_message>

is_action_countable

Returns Wheter it is useful to keep track of the number of times this effect has been activated.

Return type bool

is_turn_countable

Returns Wheter it is useful to keep track of the number of turns this effect has been active for.

Return type bool

Field

This module defines the Field class, which represents a battle field.

class poke_env.environment.field.**Field**

Bases: enum.Enum

Enumeration, represent a non null field in a battle.

ELECTRIC_TERRAIN = 2

GRASSY_TERRAIN = 3

GRAVITY = 4

HEAL_BLOCK = 5

MAGIC_ROOM = 6

MISTY_TERRAIN = 7

MUD_SPORT = 8

MUD_SPOT = 9

PSYCHIC_TERRAIN = 10

TRICK_ROOM = 11

WATER_SPORT = 12

WONDER_ROOM = 13

from_showdown_message = <function Field.from_showdown_message>

is_terrain

Wheter this field is a terrain.

Move category

This module defines the MoveCategory class, which represents a move category.

class poke_env.environment.move_category.**MoveCategory**

Bases: enum.Enum

Enumeration, represent a move category.

PHYSICAL = 1

SPECIAL = 2

STATUS = 3

Pokemon gender

This module defines the PokemonGender class, which represents the gender of a Pokemon.

class poke_env.environment.pokemon_gender.**PokemonGender**
Bases: enum.Enum

Enumeration, represent a pokemon's gender.

FEMALE = 1

MALE = 2

NEUTRAL = 3

from_request_details = <function PokemonGender.from_request_details>

Pokemon Type

This module defines the PokemonType class, which represents a Pokemon type. PokemonTypes are mainly associated with Pokemons and moves.

class poke_env.environment.pokemon_type.**PokemonType**
Bases: enum.Enum

A Pokemon type

This enumeration represents pokemon types. Each type is an instance of this class, whose name corresponds to the upper case spelling of its english name (ie. FIRE).

BUG = 1

DARK = 2

DRAGON = 3

ELECTRIC = 4

FAIRY = 5

FIGHTING = 6

FIRE = 7

FLYING = 8

GHOST = 9

GRASS = 10

GROUND = 11

ICE = 12

NORMAL = 13

POISON = 14

PSYCHIC = 15

ROCK = 16

STEEL = 17

WATER = 18

damage_multiplier(*type_1: poke_env.environment.pokemon_type.PokemonType*, *type_2: Optional[PokemonType] = None*) → float

Computes the damage multiplier from this type on a pokemon with types *type_1* and, optionally, *type_2*.

Parameters

- **type_1** (PokemonType) – The first type of the target.
- **type_2** (PokemonType, *optional*) – The second type of the target. Defaults to None.

Returns The damage multiplier from this type on a pokemon with types *type_1* and, optionally, *type_2*.

Return type float

from_name = <function PokemonType.from_name>

Side condition

This module defines the SideCondition class, which represents a in-battle side condition.

class poke_env.environment.side_condition.**SideCondition**

Bases: enum.Enum

Enumeration, represent a in-battle side condition.

AURORA_VEIL = 2

FIRE_PLEDGE = 3

GRASS_PLEDGE = 9

G_MAX_CANNONADE = 4

G_MAX_STEELSURGE = 5

G_MAX_VINE_LASH = 6

G_MAX_VOLCALITH = 7

G_MAX_WILDFIRE = 8

LIGHT_SCREEN = 10

LUCKY_CHANT = 11

MIST = 12

REFLECT = 13

SAFEGUARD = 14

SPIKES = 15

STEALTH_ROCK = 16

STICKY_WEB = 17

TAILWIND = 18

TOXIC_SPIKES = 19

WATER_PLEDGE = 20

from_showdown_message = <function SideCondition.from_showdown_message>

Status

This module defines the Status class, which represents statuses a pokemon can be afflicted with.

class poke_env.environment.status.**Status**
Bases: enum.Enum

Enumeration, represent a status a pokemon can be afflicted with.

BRN = 1

FNT = 2

FRZ = 3

PAR = 4

PSN = 5

SLP = 6

TOX = 7

Weather

This module defines the Weather class, which represents a in-battle weather.

class poke_env.environment.weather.**Weather**
Bases: enum.Enum

Enumeration, represent a non null weather in a battle.

DELTASTREAM = 3

DESOLATELAND = 2

HAIL = 4

PRIMORDIALSEA = 5

RAINDANCE = 6

SANDSTORM = 7

SUNNYDAY = 8

from_showdown_message = <function Weather.from_showdown_message>

Z Crystal

This module contains objects related ot z-crystal management. It should not be used directly.

1.3.4 The player object and related subclasses

- *Env player*
- *Player*
- *Random Player*

- *Trainable player*
- *Player network interface*

Env player

This module defines a player class exposing the Open AI Gym API.

class poke_env.player.env_player.**EnvPlayer**(*player_configuration: Optional[poke_env.player_configuration.PlayerConfiguration] = None*, ***, *avatar: Optional[int] = None*, *battle_format: Optional[str] = None*, *log_level: Optional[int] = None*, *server_configuration: Optional[poke_env.server_configuration.ServerConfiguration] = None*, *start_listening: bool = True*, *start_timer_on_battle_start: bool = False*, *team: Union[str, poke_env.teambuilder.teambuilder.Teambuilder, None] = None*)

Bases: *poke_env.player.player.Player*, gym.core.Env, abc.ABC

Player exposing the Open AI Gym Env API. Recommended use is with play_against.

MAX_BATTLE_SWITCH_RETRY = 10000

PAUSE_BETWEEN_RETRIES = 0.001

action_space
Returns the action space of the player. Must be implemented by subclasses.

choose_move(*battle: poke_env.environment.abstract_battle.AbstractBattle*) → poke_env.player.battle_order.BattleOrder
Abstract method to choose a move in a battle.

Parameters **battle** (AbstractBattle) – The battle.

Returns The move order.

Return type str

close() → None
Unimplemented. Has no effect.

complete_current_battle() → None
Completes the current battle by forfeiting.

compute_reward(*battle: poke_env.environment.abstract_battle.AbstractBattle*) → float
Returns a reward for the given battle.

The default implementation corresponds to the default parameters of the reward_computing_helper method.

Parameters **battle** (AbstractBattle) – The battle for which to compute the reward.

Returns The computed reward.

Return type float

embed_battle(*battle: poke_env.environment.abstract_battle.AbstractBattle*) → Any
Abstract method for embedding battles.

Parameters **battle** (AbstractBattle) – The battle whose state is being embedded

Returns The computed embedding

Return type Any

play_against(*env_algorithm: Callable*, *opponent: poke_env.player.player.Player*, *env_algorithm_kwargs=None*)
Executes a function controlling the player while facing opponent.

The env_algorithm function is executed with the player environment as first argument. It exposes the open ai gym API.

Additional arguments can be passed to the env_algorithm function with env_algorithm_kwargs.

Battles against opponent will be launched as long as env_algorithm is running. When env_algorithm returns, the current active battle will be finished randomly if it is not already.

Parameters

- **env_algorithm** (*callable*) – A function that controls the player. It must accept the player as first argument. Additional arguments can be passed with the env_algorithm_kwargs argument.
- **opponent** (Player) – A player against with the env player will player.
- **env_algorithm_kwargs** – Optional arguments to pass to the env_algorithm. Defaults to None.

render(*mode='human'*) → None
A one line rendering of the current state of the battle.

reset() → Any
Resets the internal environment state. The current battle will be set to an active unfinished battle.

Returns The observation of the new current battle.

Return type Any

Raies EnvironmentError

reward_computing_helper(*battle: poke_env.environment.abstract_battle.AbstractBattle*, ***, *fainted_value: float = 0.0*, *hp_value: float = 0.0*, *number_of_pokemons: int = 6*, *starting_value: float = 0.0*, *status_value: float = 0.0*, *victory_value: float = 1.0*) → float
A helper function to compute rewards.

The reward is computed by computing the value of a game state, and by comparing it to the last state.

State values are computed by weighting different factor. Fainted pokemons, their remaining HP, inflicted statuses and winning are taken into account.

For instance, if the last time this function was called for battle A it had a state value of 8 and this call leads to a value of 9, the returned reward will be 9 - 8 = 1.

Consider a single battle where each player has 6 pokemons. No opponent pokemon has fainted, but our team has one fainted pokemon. Three opposing pokemons are burned. We have one pokemon missing half of its HP, and our fainted pokemon has no HP left.

The value of this state will be:

- **With fainted value: 1, status value: 0.5, hp value: 1:** = - 1 (fainted) + 3 * 0.5 (status) - 1.5 (our hp) = -1
- **With fainted value: 3, status value: 0, hp value: 1:** = - 3 + 3 * 0 - 1.5 = -4.5

Parameters

- **battle** (AbstractBattle) – The battle for which to compute rewards.
- **fainted_value** (*float*) – The reward weight for fainted pokemons. Defaults to 0.
- **hp_value** (*float*) – The reward weight for hp per pokemon. Defaults to 0.
- **number_of_pokemons** (*int*) – The number of pokemons per team. Defaults to 6.
- **starting_value** (*float*) – The default reference value evaluation. Defaults to 0.
- **status_value** (*float*) – The reward value per non-fainted status. Defaults to 0.
- **victory_value** (*float*) – The reward value for winning. Defaults to 1.

Returns The reward.

Return type float

seed(*seed=None*) → None
Sets the numpy seed.

step(*action: int*) → Tuple
Performs action in the current battle.

Parameters **action** (*int*) – The action to perform.

Returns A tuple containing the next observation, the reward, a boolean indicating wheter the episode is finished, and additional information

Return type tuple

class poke_env.player.env_player.**Gen4EnvSinglePlayer**(*player_configuration: Optional[poke_env.player_configuration.PlayerConfigurat = None, *, avatar: Optional[int] = None, battle_format: Optional[str] = None, log_level: Optional[int] = None, server_configuration: Optional[poke_env.server_configuration.ServerConfigurati = None, start_listening: bool = True, start_timer_on_battle_start: bool = False, team: Union[str, poke_env.teambuilder.teambuilder.Teambuilder, None] = None*)

Bases: *poke_env.player.env_player.EnvPlayer*

action_space
The action space for gen 7 single battles.

The conversion to moves is done as follows:

0 <= action < 4: The actionth available move in battle.available_moves is executed.

4 <= action < 10 The action - 4th available switch in battle.available_switches is executed.

class poke_env.player.env_player.**Gen5EnvSinglePlayer**(*player_configuration: Optional[poke_env.player_configuration.PlayerConfigurat = None*, ***, *avatar: Optional[int] = None*, *battle_format: Optional[str] = None*, *log_level: Optional[int] = None*, *server_configuration: Optional[poke_env.server_configuration.ServerConfigurati = None*, *start_listening: bool = True*, *start_timer_on_battle_start: bool = False*, *team: Union[str, poke_env.teambuilder.teambuilder.Teambuilder, None] = None*)

Bases: *poke_env.player.env_player.Gen4EnvSinglePlayer*

class poke_env.player.env_player.**Gen6EnvSinglePlayer**(*player_configuration: Optional[poke_env.player_configuration.PlayerConfigurat = None*, ***, *avatar: Optional[int] = None*, *battle_format: Optional[str] = None*, *log_level: Optional[int] = None*, *server_configuration: Optional[poke_env.server_configuration.ServerConfigurati = None*, *start_listening: bool = True*, *start_timer_on_battle_start: bool = False*, *team: Union[str, poke_env.teambuilder.teambuilder.Teambuilder, None] = None*)

Bases: *poke_env.player.env_player.EnvPlayer*

action_space

The action space for gen 7 single battles.

The conversion to moves is done as follows:

0 <= action < 4: The actionth available move in battle.available_moves is executed.

4 <= action < 8: The action - 8th available move in battle.available_moves is executed, with mega-evolution.

8 <= action < 14 The action - 8th available switch in battle.available_switches is executed.

class poke_env.player.env_player.**Gen7EnvSinglePlayer**(*player_configuration: Optional[poke_env.player_configuration.PlayerConfigurat = None, *, avatar: Optional[int] = None, battle_format: Optional[str] = None, log_level: Optional[int] = None, server_configuration: Optional[poke_env.server_configuration.ServerConfigurati = None, start_listening: bool = True, start_timer_on_battle_start: bool = False, team: Union[str, poke_env.teambuilder.teambuilder.Teambuilder, None] = None*)

Bases: *poke_env.player.env_player.EnvPlayer*

action_space

The action space for gen 7 single battles.

The conversion to moves is done as follows:

0 <= action < 4: The actionth available move in battle.available_moves is executed.

4 <= action < 8: The action - 4th available move in battle.available_moves is executed, with z-move.

8 <= action < 12: The action - 8th available move in battle.available_moves is executed, with mega-evolution.

12 <= action < 18 The action - 12th available switch in battle.available_switches is executed.

class poke_env.player.env_player.**Gen8EnvSinglePlayer**(*player_configuration: Optional[poke_env.player_configuration.PlayerConfigurat = None, *, avatar: Optional[int] = None, battle_format: Optional[str] = None, log_level: Optional[int] = None, server_configuration: Optional[poke_env.server_configuration.ServerConfigurati = None, start_listening: bool = True, start_timer_on_battle_start: bool = False, team: Union[str, poke_env.teambuilder.teambuilder.Teambuilder, None] = None*)

Bases: *poke_env.player.env_player.EnvPlayer*

action_space

The action space for gen 7 single battles.

The conversion to moves is done as follows:

0 <= action < 4: The actionth available move in battle.available_moves is executed.

4 <= action < 8: The action - 4th available move in battle.available_moves is executed, with z-move.

8 <= action < 12: The action - 8th available move in battle.available_moves is executed, with mega-evolution.

12 <= action < 16: The action - 12th available move in battle.available_moves is executed, while dynamaxing.

16 <= action < 22 The action - 16th available switch in battle.available_switches is executed.

Player

This module defines a base class for players.

class poke_env.player.player.**Player**(*player_configuration: Optional[poke_env.player_configuration.PlayerConfiguration] = None*, ***, *avatar: Optional[int] = None*, *battle_format: str = 'gen8randombattle'*, *log_level: Optional[int] = None*, *max_concurrent_battles: int = 1*, *server_configuration: Optional[poke_env.server_configuration.ServerConfiguration] = None*, *start_timer_on_battle_start: bool = False*, *start_listening: bool = True*, *team: Union[str, poke_env.teambuilder.teambuilder.Teambuilder, None] = None*)

Bases: *poke_env.player.player_network_interface.PlayerNetwork*, abc.ABC

Base class for players.

DEFAULT_CHOICE_CHANCE = 0.001

MESSAGES_TO_IGNORE = {'', 'expire', 't:'}

accept_challenges(*opponent: Union[str, List[str], None]*, *n_challenges: int*) → None

Let the player wait for challenges from opponent, and accept them.

If opponent is None, every challenge will be accepted. If opponent if a string, all challenges from player with that name will be accepted. If opponent is a list all challenges originating from players whose name is in the list will be accepted.

Up to n_challenges challenges will be accepted, after what the function will wait for these battles to finish, and then return.

Parameters

- **opponent** (*None, str or list of str*) – Players from which challenges will be accepted.
- **n_challenges** (*int*) – Number of challenges that will be accepted

battle_against(*opponent: poke_env.player.player.Player*, *n_battles: int*) → None

Make the player play n_battles against opponent.

This function is a wrapper around send_challenges and accept challenges.

Parameters

- **opponent** (Player) – The opponent to play against.
- **n_battles** (*int*) – The number of games to play.

battles

choose_default_move(**args*, ***kwargs*) → poke_env.player.battle_order.DefaultBattleOrder
Returns showdown's default move order.

This order will result in the first legal order - according to showdown's ordering - being chosen.

choose_move(*battle: poke_env.environment.abstract_battle.AbstractBattle*) → poke_env.player.battle_order.BattleOrder
Abstract method to choose a move in a battle.

Parameters **battle** (AbstractBattle) – The battle.

Returns The move order.

Return type str

choose_random_doubles_move(*battle: poke_env.environment.double_battle.DoubleBattle*) → poke_env.player.battle_order.BattleOrder

choose_random_move(*battle: poke_env.environment.abstract_battle.AbstractBattle*) → poke_env.player.battle_order.BattleOrder
Returns a random legal move from battle.

Parameters **battle** (AbstractBattle) – The battle in which to move.

Returns Move order

Return type str

choose_random_singles_move(*battle: poke_env.environment.battle.Battle*) → poke_env.player.battle_order.BattleOrder

static create_order(*order: Union[poke_env.environment.move.Move, poke_env.environment.pokemon.Pokemon], mega: bool = False, z_move: bool = False, dynamax: bool = False, move_target: int = 0*) → poke_env.player.battle_order.BattleOrder
Formats an move order corresponding to the provided pokemon or move.

Parameters

- **order** (*Move or Pokemon*) – Move to make or Pokemon to switch to.
- **mega** (*bool*) – Whether to mega evolve the pokemon, if a move is chosen.
- **z_move** (*bool*) – Whether to make a zmove, if a move is chosen.
- **dynamax** (*bool*) – Whether to dynamax, if a move is chosen.
- **move_target** (*int*) – Target Pokemon slot of a given move

Returns Formatted move order

Return type str

format

format_is_doubles

ladder(*n_games*)
Make the player play games on the ladder.

n_games defines how many battles will be played.

Parameters **n_games** (*int*) – Number of battles that will be played

n_finished_battles

n_lost_battles

n_tied_battles

n_won_battles

random_teampreview(*battle: poke_env.environment.abstract_battle.AbstractBattle*) → str
Returns a random valid teampreview order for the given battle.

Parameters **battle** (AbstractBattle) – The battle.

Returns The random teampreview order.

Return type str

reset_battles() → None
Resets the player's inner battle tracker.

send_challenges(*opponent: str*, *n_challenges: int*, *to_wait: Optional[asyncio.locks.Event] = None*) → None
Make the player send challenges to opponent.

opponent must be a string, corresponding to the name of the player to challenge.

n_challenges defines how many challenges will be sent.

to_wait is an optional event that can be set, in which case it will be waited before launching challenges.

Parameters

- **opponent** (*str*) – Player username to challenge.
- **n_challenges** (*int*) – Number of battles that will be started
- **to_wait** (*Event, optional.*) – Optional event to wait before launching challenges.

teampreview(*battle: poke_env.environment.abstract_battle.AbstractBattle*) → str
Returns a teampreview order for the given battle.

This order must be of the form /team TEAM, where TEAM is a string defining the team chosen by the player. Multiple formats are supported, among which '3461' and '3, 4, 6, 1' are correct and indicate leading with pokemon 3, with pokemons 4, 6 and 1 in the back in single battles or leading with pokemons 3 and 4 with pokemons 6 and 1 in the back in double battles.

Please refer to Pokemon Showdown's protocol documentation for more information.

Parameters **battle** (AbstractBattle) – The battle.

Returns The teampreview order.

Return type str

win_rate

Random Player

This module defines a random players baseline

class poke_env.player.random_player.**RandomPlayer**(*player_configuration: Optional[poke_env.player_configuration.PlayerConfiguration] = None*, ***, *avatar: Optional[int] = None*, *battle_format: str = 'gen8randombattle'*, *log_level: Optional[int] = None*, *max_concurrent_battles: int = 1*, *server_configuration: Optional[poke_env.server_configuration.ServerConfiguration] = None*, *start_timer_on_battle_start: bool = False*, *start_listening: bool = True*, *team: Union[str, poke_env.teambuilder.teambuilder.Teambuilder, None] = None*)

Bases: *poke_env.player.player.Player*

choose_move(*battle*) → poke_env.player.battle_order.BattleOrder

Abstract method to choose a move in a battle.

Parameters **battle** (AbstractBattle) – The battle.

Returns The move order.

Return type str

Trainable player

Warning: This class is experimental and currently not tested. Use at your own risk. We recommend using EnvPlayer instead.

This module defines a random players baseline

class poke_env.player.trainable_player.**TrainablePlayer**(*player_configuration: poke_env.player_configuration.PlayerConfiguration*, ***, *avatar: Optional[int] = None*, *battle_format: str*, *log_level: Optional[int] = None*, *max_concurrent_battles: int = 1*, *model=None*, *server_configuration: poke_env.server_configuration.ServerConfiguration*, *start_timer_on_battle_start: bool = False*, *start_listening: bool = True*, *team: Union[str, poke_env.teambuilder.teambuilder.Teambuilder, None] = None*)

Bases: *poke_env.player.player.Player*, abc.ABC

This is an experimental API.

action_to_move(*action*, *battle: poke_env.environment.abstract_battle.AbstractBattle*)

battle_to_state(*battle: poke_env.environment.abstract_battle.AbstractBattle*)

choose_move(*battle: poke_env.environment.abstract_battle.AbstractBattle*) → poke_env.player.battle_order.BattleOrder
Abstract method to choose a move in a battle.

Parameters **battle** (AbstractBattle) – The battle.

Returns The move order.

Return type str

static init_model()

n_replays

replay(*battle_history: Dict[KT, VT]*)

state_to_action(*state: numpy.array*, *battle: poke_env.environment.abstract_battle.AbstractBattle*)

train_against(*opponent: poke_env.player.trainable_player.TrainablePlayer*, *n_battles=100*, *train_opponent: bool = False*)

training_data

Player network interface

This module defines a base class for communicating with showdown servers.

class poke_env.player.player_network_interface.**PlayerNetwork**(*player_configuration: poke_env.player_configuration.PlayerConfig*, **, avatar: Optional[int] = None*, *log_level: Optional[int] = None*, *server_configuration: poke_env.server_configuration.ServerConfig*, *start_listening: bool = True*)

Bases: abc.ABC

Network interface of a player.

Responsible for communicating with showdown servers. Also implements some higher level methods for basic tasks, such as changing avatar and low-level message handling.

listen() → None
Listen to a showdown websocket and dispatch messages to be handled.

logged_in
Event object associated with user login.

Returns The logged-in event

Return type Event

logger
Logger associated with the player.

Returns The logger.

Return type Logger

stop_listening() → None

username
The player's username.

Returns The player's username.

Return type str

websocket_url
The websocket url.

It is derived from the server url.

Returns The websocket url.

Return type str

1.3.5 The pokemon object

class poke_env.environment.pokemon.**Gen4Pokemon**(*, *species: Optional[str] = None*, *request_pokemon: Optional[Dict[str, Any]] = None*, *details: Optional[str] = None*)
Bases: *poke_env.environment.pokemon.Pokemon*

MOVE_CLASS
alias of *poke_env.environment.move.Gen4Move*

class poke_env.environment.pokemon.**Gen5Pokemon**(*, *species: Optional[str] = None*, *request_pokemon: Optional[Dict[str, Any]] = None*, *details: Optional[str] = None*)
Bases: *poke_env.environment.pokemon.Pokemon*

MOVE_CLASS
alias of *poke_env.environment.move.Gen5Move*

class poke_env.environment.pokemon.**Gen6Pokemon**(*, *species: Optional[str] = None*, *request_pokemon: Optional[Dict[str, Any]] = None*, *details: Optional[str] = None*)
Bases: *poke_env.environment.pokemon.Pokemon*

MOVE_CLASS
alias of *poke_env.environment.move.Gen6Move*

class poke_env.environment.pokemon.**Gen7Pokemon**(*, *species: Optional[str] = None*, *request_pokemon: Optional[Dict[str, Any]] = None*, *details: Optional[str] = None*)
Bases: *poke_env.environment.pokemon.Pokemon*

MOVE_CLASS
alias of *poke_env.environment.move.Gen7Move*

class poke_env.environment.pokemon.**Gen8Pokemon**(*, *species: Optional[str] = None*, *request_pokemon: Optional[Dict[str, Any]] = None*, *details: Optional[str] = None*)
Bases: *poke_env.environment.pokemon.Pokemon*

MOVE_CLASS
alias of *poke_env.environment.move.Gen8Move*

class poke_env.environment.pokemon.**Pokemon**(*, *species: Optional[str] = None*, *request_pokemon: Optional[Dict[str, Any]] = None*, *details: Optional[str] = None*)

Bases: object

MOVE_CLASS
alias of *poke_env.environment.move.Gen8Move*

ability

Returns The pokemon's ability. None if unknown.

Return type str, optional

active

Returns Boolean indicating whether the pokemon is active.

Return type bool

available_moves_from_request(*request: Dict[KT, VT]*) → List[poke_env.environment.move.Move]

available_z_moves
Caution: this property is not properly tested yet.

Returns The set of moves that pokemon can use as z-moves.

Return type List[*Move*]

base_stats

Returns The pokemon's base stats.

Return type Dict[str, int]

boosts

Returns The pokemon's boosts.

Return type Dict[str, int]

current_hp

Returns The pokemon's current hp. For your pokemons, this is the actual value. For opponent's pokemon, this value depends on showdown information: it can be on a scale from 0 to 100 or on a pixel scale.

Return type int

current_hp_fraction

Returns The pokemon's current remaining hp fraction.

Return type float

damage_multiplier(*type_or_move: Union[poke_env.environment.pokemon_type.PokemonType, poke_env.environment.move.Move]*) → float
Returns the damage multiplier associated with a given type or move on this pokemon.

This method is a shortcut for PokemonType.damage_multiplier with relevant types.

Parameters **type_or_move** (PokemonType *or* Move) – The type or move of interest.

Returns The damage multiplier associated with given type on the pokemon.

Return type float

effects

Returns A dict mapping the effects currently affecting the pokemon and the associated counter.

Return type Dict[*Effect*, int]

fainted

Returns Wheter the pokemon has fainted.

Return type bool

first_turn

Returns Wheter this is this pokemon's first action since its last switch in.

Return type bool

gender

Returns The pokemon's gender.

Return type *PokemonGender*, optional

height

Returns The pokemon's height, in meters.

Return type float

is_dynamaxed

Returns Whether the pokemon is currently dynamaxed

Return type bool

item

Returns The pokemon's item.

Return type Optional[str]

level

Returns The pokemon's level.

Return type int

max_hp

Returns The pokemon's max hp. For your pokemons, this is the actual value. For opponent's pokemon, this value depends on showdown information: it can be on a scale from 0 to 100 or on a pixel scale.

Return type int

moves

Returns A dictionary of the pokemon's known moves.

Return type Dict[str, *Move*]

must_recharge

Returns A boolean indicating whether the pokemon must recharge.

Return type bool

pokeball

Returns The pokeball in which is the pokemon.

Return type Optional[str]

possible_abilities

Returns The list of possible abilities for this pokemon.

Return type List[str]

preparing

Returns Whether this pokemon is preparing a multi-turn move.

Return type bool

protect_counter

Returns How many protect-like moves where used in a row by this pokemon.

Return type int

revealed

Returns Whether this pokemon has appeared in the current battle.

Return type bool

shiny

Returns Whether this pokemon is shiny.

Return type bool

species

Returns The pokemon's species.

Return type Optional[str]

stats

Returns The pokemon's stats, as a dictionary.

Return type Dict[str, Optional[int]]

status

Returns The pokemon's status.

Return type Optional[*Status*]

status_counter

Returns The pokemon's status turn count. Only counts TOXIC and SLEEP statuses.

Return type int

type_1

Returns The pokemon's first type.

Return type *PokemonType*

type_2

Returns The pokemon's second type.

Return type Optional[*PokemonType*]

types

Returns The pokemon's types, as a tuple.

Return type Tuple[*PokemonType*, Optional[*PokemonType*]]

weight

Returns The pokemon's weight, in kilograms.

Return type float

1.3.6 The teambuilder object and related classes

Base Teambuilder

This module defines the Teambuilder abstract class, which represents objects yielding Pokemon Showdown teams in the context of communicating with Pokemon Showdown.

class poke_env.teambuilder.teambuilder.**Teambuilder**

Bases: abc.ABC

Teambuilder objects allow the generation of teams by Player instances.

They must implement the yield_team method, which must return a valid packed-formatted showdown team every time it is called.

This format is a custom format decribed in Pokemon's showdown protocol documentation: https://github.com/smogon/pokemon-showdown/blob/master/PROTOCOL.md#team-format

This class also implements a helper function to convert teams from the classical showdown team text format into the packed-format.

static join_team(*team: List[poke_env.teambuilder.teambuilder_pokemon.TeambuilderPokemon]*) → str

Converts a list of TeambuilderPokemon objects into the corresponding packed showdown team format.

Parameters **team** (*list of TeambuilderPokemon*) – The list of TeambuilderPokemon objects that form the team.

Returns The formatted team string.

Return type str

static parse_showdown_team(*team: str*) → List[poke_env.teambuilder.teambuilder_pokemon.TeambuilderPokemon]

Converts a showdown-formatted team string into a list of TeambuilderPokemon objects.

This method can be used when using teams built in the showdown teambuilder.

Parameters **team** (*str*) – The showdown-format team to convert.

Returns The formatted team.

Return type list of TeambuilderPokemon

yield_team() → str

Returns a packed-format team.

Constant teambuilder

This module defines the ConstantTeambuilder class, which is a subclass of ShowdownTeamBuilder that yields a constant team.

class poke_env.teambuilder.constant_teambuilder.**ConstantTeambuilder**(*team: str*)

Bases: *poke_env.teambuilder.teambuilder.Teambuilder*

yield_team() → str
Returns a packed-format team.

Teambuilder pokemon

This module defines the TeambuilderPokemon class, which is used as an intermediate format to specify pokemon builds in teambuilders custom classes.

class poke_env.teambuilder.teambuilder_pokemon.**TeambuilderPokemon**(*nickname=None*, *species=None*, *item=None*, *ability=None*, *moves=None*, *nature=None*, *evs=None*, *gender=None*, *ivs=None*, *shiny=None*, *level=None*, *happiness=None*, *hiddenpowertype=None*, *gmax=None*)

Bases: object

HP_TO_IVS = {'bug': [31, 31, 31, 30, 31, 30], 'dark': [31, 31, 31, 31, 31, 31], 'dra

STATS_TO_IDX = {'atk': 1, 'def': 2, 'hp': 0, 'satk': 3, 'sdef': 4, 'spa': 3, 'sp

formatted

formatted_endstring

formatted_evs

formatted_ivs

formatted_moves

1.3.7 Top level modules

Utils

This module contains utility functions and objects.

poke_env.utils.**compute_raw_stats**(*species: str, evs: List[int], ivs: List[int], level: int, nature: str*) → List[int]

Converts to raw stats :param species: pokemon species :param evs: list of pokemon's EVs (size 6) :param ivs: list of pokemon's IVs (size 6) :param level: pokemon level :param nature: pokemon nature :return: the raw stats in order [hp, atk, def, spa, spd, spe]

Player configuration

This module contains objects related to player configuration.

class poke_env.player_configuration.**PlayerConfiguration**(*username*, *password*)

Bases: tuple

Player configuration object. Represented with a tuple with two entries: username and password.

password

Alias for field number 1

username

Alias for field number 0

Server configuration

This module contains objects related to server configuration.

poke_env.server_configuration.**LocalhostServerConfiguration = ServerConfiguration(server_url**

Server configuration with localhost and smogon's authentication endpoint.

class poke_env.server_configuration.**ServerConfiguration**(*server_url*, *authentication_url*)

Bases: tuple

Server configuration object. Represented with a tuple with two entries: server url and authentication endpoint url.

authentication_url

Alias for field number 1

server_url

Alias for field number 0

poke_env.server_configuration.**ShowdownServerConfiguration = ServerConfiguration(server_url=**

Server configuration with smogon's server and authentication endpoint.

CHAPTER 2

Other

2.1 Acknowledgements

This project is a follow-up of a group project from an artifical intelligence class at Ecole Polytechnique.

You can find the original repository here. It is partially inspired by the showdown-battle-bot project. Of course, none of these would have been possible without Pokemon Showdown.

2.2 Data

Data files are adapted version of the `js` data files of Pokemon Showdown.

Team data comes from Smogon forums' RMT section.

2.3 License

Python Module Index

p

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z