

Assignment 2

DUED : Friday Week 8, 11:55pm.

Introduction

Aims

- To adequately analyse the complexity of various operations / algorithms.
- To be able to use readily available Stack, Queue, Set, and SortedList ADTs to solve problems efficiently.
- To continue practising the implementation of correct, high quality and well-documented code.

Important

<https://edstem.org/au/courses/8772/discussion/988663>
Write own testcase for Task 1 and 2 in test_pokemon.py

- Much of this assessment assumes some knowledge of Object Oriented Programming basics, as would have been covered in FIT1045. If you are taking FIT2085 or are generally behind on this, slides are available in the Week 5/6 sections on moodle.
- Use the names provided for internal and external methods, as we will use them in our testing harnesses.
- Provide appropriate documentation for each file, class, function, and method you define or modify. More on this can be found in the general constraints.
- For the entirety of the assessment, **you may NOT use python inbuilt lists or the ArrayR class explicitly**. This is of course unless a particular task says otherwise. The purpose of this assessment is to grade your ability to use the ADTs discussed in the unit, and so we don't want you leaning too hard on arrays/lists.
- For all constants described in this sheet, **design your code in such a way that it is very easy to change these constants** and still have the **project work**. For example, how hard would it be to change the maximum team size from 6 to 12? You should make it so that this is a one line change.
- Tasks are really not separated by length or difficulty. Please read all tasks before deciding how your team should split the work.
- Any mention of "Randomness" should use the random_gen.py file given to you. Choosing a random number is done with `RandomGen.randint`, and checking something happens with a 30% chance is done with `RandomGen.random_chance(0.3)`
- **All methods mentioned in the spec, unless otherwise specified, require you to write 3 documented unit tests in the relevant file in the `tests` directory**. These should have docstrings, be reasonable tests to have, and also reasonably different.
- Methods which are marked as "**user facing**" should have checks to ensure that incorrect or invalid input for any **arguments is handled with preconditions**. For example, a function `price_of_apples(num_apples)` should check that `num_apples` is a number, and a non-negative one at that.

- Can use ENUM
- Dictionaries NOT ALLOWED, use hash tables instead
- Ghastly always evolves to Haunter after it's turn (Base lvl 1) and it'll be a lvl 2 Haunter
- Inheritance is required to be used (Rubric)
- Can add additional functions/methods & instance variables, if necessary, inside classes

Submission, Format, and Expectations

This assignment is a group assignment. Your group should complete the Group Work Agreement before commencing work on this assignment. Your group will need to divide the work and submit the assignment together. The assignment will require you to submit all Python files that are detailed in each task. The naming convention for each file is given in the tasks.

Once you have all the files completed, please zip them together and name the zip file in the following format:

`<groupname>_assignment2.zip`

For Example - A2_Group_100_assignment2.zip

And upload to both Gradescope and Moodle. Please do **not** include the .git or MACOSX folders in your submission, just make a new folder, and copy across your task's python files.

General Constraints and Assumptions

- The assignment will provide you with liberties on how to code your solution. However, please, make sure to go through the assessment rubric to see how you will be graded and certain concepts that need to be covered in your solution.
- When a pre-condition is violated, the function should raise an Exception (in our case, a ValueError unless specified otherwise). Every argument that has a pre-condition should be validated to ensure it meets the pre-condition.
- Preconditions must be checked in abstract methods if possible.
- The docstring for each method should contain both the best- and worst-case complexity of the method. You are allowed to include a catchall in your class header (IE all functions, unless stated otherwise, have best/worst case complexity of $O(1)$)
- Functions that require user input should be manually tested with some appropriate cases. This doesn't need to be put anywhere as part of your submission but helps ensure correctness when it is marked.
- Please DO NOT modify any of the given methods in any of the ADT files. You may also not add any additional methods to the ADT unless a particular task states otherwise.

Background

*We've heard that you want to be the very best
Like no one ever was
To get a HD is your real quest
To study is your cause*

*Will you read through the entire document
Making lots of notes
Teach yourself ADT concepts
And the power that they hold?*

pokemon name = name of base type pokemon
pokemon types = element (eg. water, fire, grass)

Calling all Pokemon trainers! The adventure of your lifetime begins here. We challenge you to an adventure across our sunny island to catch Pikachu and friends. Well, not really, but how cool would that be! We are tasked with developing a battle simulator for everyone to fight their favourite Pokemon and we need your help to code it all in Python! So, stay with us, read through the document carefully and help us design the best command line Pokemon battle simulator ever!

This game is for two players. It involves the creation of a team of different kinds of Pokemon in order to battle the team against another player until a victor is produced. There are various types of battle modes, and each player can only select up to 6 Pokemon in their team. However, for your sanity, we have limited the types of Pokemon. These are discussed in detail below.

Pokemon Types and Stats

Before handling the pokemon stats, one thing that should be discussed is a Pokemon's "effective attack". In the Pokemon games, Pokemon have types, which determine how effective some moves are against others. In our case, the types of each pokemon (attacking and defending) determine the multiplier applied to the attacking stat. This is called the effective attack.

The multipliers for attacks is as follows:

AT Type \ DEF Type	Fire	Grass	Water	Ghost	Normal
Fire	1	2	0.5	1	1
Grass	0.5	1	2	1	1
Water	2	0.5	1	1	1
Ghost	1.25	1.25	1.25	2	0
Normal	1.25	1.25	1.25	0	1

So if a Venusaur (Grass type) attacks a Squirtle (Water type), the effective attack would be the Venusaur's Attack stat **multiplied by 2**.

Note: A multiplier of 0 *does not* mean the attack does not succeed. It just does 0 damage. This will become important when dealing with status effects.

We will be dealing with the following base types of Pokemon:

Note:

Stat	Explanation
Name	Name of the Pokemon

Stat	Explanation
Type	Type of the Pokemon. Subject to type effectiveness table mentioned later
Base Level	All pokemon start at some particular level (For base pokemon, this is level 1, for others, it is the level at which they evolve).
HP	The Hit Points of the Pokemon, essentially their health. If this goes down to or below 0, the pokemon is 'fainted'
Attack	The attack points of the Pokemon.
Speed	The speed points of the Pokemon.
Defence	The defence points of the Pokemon
Defence Calculation	The calculation of how the Pokemon reacts when being attacked with effective attack points

BASE POKEMON

1.

Stat	Value
Name	Charmander
Type	Fire
Base Level	1
HP	$8 + 1 * \text{Level}$
Attack	$6 + 1 * \text{Level}$
Speed	$7 + 1 * \text{Level}$
Defence	4
Defence Calculation	When the effective attack is greater than Charmander's defence, Charmander loses HP equal to the effective attack. Otherwise, it loses only half of the effective attack (// 2).

if effective atk > defense
then HP - effective atk
else, HP - (effective atk // 2)

2.

Stat	Value
Name	Squirtle
Type	Water

Stat	Value
Base Level	1
HP	$9 + 2 * \text{Level}$
Attack	$4 + (\text{Level} // 2)$
Speed	7
Defence	$6 + \text{Level}$
Defence Calculation	<p>When the effective attack is greater than twice of Squirtle's defence, Squirtle loses HP equal to the effective attack. Otherwise, it loses only half as much of the HP as the effective attack (// 2)</p> <p>if effective atk > 2*defense then HP - effective atk else, HP - (effective atk // 2)</p>

3.

Stat	Value
Name	Bulbasaur
Type	Grass
Base Level	1
HP	$12 + 1 * \text{Level}$
Attack	5
Speed	$7 + (\text{Level} // 2)$
Defence	5
Defence Calculation	<p>When the effective attack is greater than (Bulbasaur's defence + 5), Bulbasaur loses HP equal to the effective attack. Otherwise, it loses only half as much of the HP as the effective attack (// 2)</p>

4. if effective atk > (defence + 5) then HP - effective atk ; else, HP - (effective atk // 2)

Stat	Value
Name	Gastly
Type	Ghost
Base Level	1
HP	$6 + (\text{Level} // 2)$
Attack	4
Speed	2

Stat	Value
Defence	8
Defence Calculation	Loses HP equal to the effective attack HP - effective atk

5.

Stat	Value
Name	Eevee
Type	Normal
Base Level	1
HP	10
Attack	6 + Level
Speed	7 + Level
Defence	4 + Level
Defence Calculation	When the effective attack is greater or equal to Eevee's defence, lose HP equal to the effective attack. Otherwise lose no HP.

If effective atk >= defense, HP - effective atk
else, HP - 0 (remain unchanged)

EVOLVED POKEMON

6.

Stat	Value
Name	Charizard
Type	Fire
Evolves From	Charmander
Base Level	3
HP	12 + 1 * Level
Attack	10 + 2 * Level
Speed	9 + 1 * Level
Defence	4
Defence	When the effective attack is greater than Charizard's defence,

Stat	Value
Calculation	Charizard loses HP equal to twice the effective attack. Otherwise, it loses HP equal to the effective attack

7. if effective atk > defense, HP - 2(effective atk) ; else, HP - effective atk

Stat	Value
Name	Blastoise
Type	Water
Evolves From	Squirtle
Base Level	3
HP	15 + 2 * Level
Attack	8 + (Level // 2)
Speed	10
Defence	8 + 1 * Level
Defence Calculation	When the effective attack is greater than double Blastoise's defence, Blastoise loses HP equal to the effective attack. Otherwise, it loses HP equal to half the effective attack (//2)

8. if effective atk > 2*defense, HP - effective atk ; else, HP - (effective atk // 2)

Stat	Value
Name	Venusaur
Type	Grass
Evolves From	Bulbasaur
Base Level	2
HP	20 + (Level // 2)
Attack	5
Speed	3 + (Level // 2)
Defence	10
Defence Calculation	When the effective attack is greater than (Venusaur's defence + 5), Venusaur loses HP equal to the effective attack. Otherwise, it loses HP equal to half the effective attack (//2)

9. if effective atk > (defense+5), HP - effective atk ; else, HP - (effective atk // 2)

Stat	Value
Name	Haunter

Stat	Value
Type	Ghost
Evolves From	Gastly
Base Level	1
HP	$9 + (\text{Level} // 2)$
Attack	8
Speed	6
Defence	6
Defence Calculation	Loses HP equal to the effective attack HP - effective atk

10.

Stat	Value
Name	Gengar
Type	Ghost
Evolves From	Haunter
Base Level	3
HP	$12 + (\text{Level} // 2)$
Attack	18
Speed	12
Defence	3
Defence Calculation	Loses HP equal to the effective attack HP - effective atk

Status Effects

Pokemon each have unique status effects they can inflict when battling. Pokemon can **only have 1 status effect at a time**, and **status effects are cleared when a pokemon is taken off the field** (even if they are immediately sent back on).

The pokemon's type affects what status it can inflict:

Type	Effect
Fire	Burn: After successfully attacking another pokemon, lose 1 hp. Halves effective attack.

HP -1
effective atk / 2 (opponent's)

Type	Effect
Grass	Poison: After successfully attacking another pokemon, lose 3 hp.
Water	Paralysis: Halves speed. Speed // 2
Ghost	Sleep: Always fails to attack. Attack fail = no atk = opponent HP - 0
Normal	Confusion: When attacking, you have a 50% chance to attack yourself. 50% chance (0.5) to inflict damage to self = atk others also atk yourself

HP - 3

atk urself
= minus
own HP

under random_gen.py
RandomGen.random_chance(0.5)

Whenever you successfully attack another Pokemon, you have a 20% chance to inflict your respective status effect onto that Pokemon (even if they already have a different status effect, this overwrites it). RandomGen.random_chance(0.2)

Battling

How PokeTeams are formed will be run through in more detail in task 3, so let's get right on into battling.

A battle occurs between two PokeTeams, and each PokeTeam has the ability to return or retrieve pokemon for battling.

A battle starts with each PokeTeam retrieving a pokemon to send to the field, and face off.

Then, each team has a choice of 4 options:


- **Attack:** The current pokemon on the field attacks the opposing team
- **Swap:** The current pokemon is returned to the field and another pokemon is retrieved from the team (could be the same pokemon)
- **Heal:** 3 times per battle, each team can fully heal the pokemon and clear any status effects.
- **Special:** Return your pokemon, do a special action on your team (depends on battle mode) and then retrieve a new pokemon for the field.

The battle then does the following, in order:

- Handle any swap actions
- Handle any special actions
- Handle any heal actions (If a heal is requested but that team has already healed thrice, then they automatically lose the battle)
- Handle attacks. In the case of both pokemon attacking, the following is done:
 - The faster of the two pokemon attacks first. If the defending pokemon still has not fainted, then the defending pokemon attacks second.
 - In the event where both pokemon have the same speed stat, they both attack each other regardless, but team 1's attack should be processed first (in the event of any status effects)
- If both pokemon are still alive, then they both lose 1 HP
- If one pokemon has fainted and the other has not, the remaining pokemon level up
- If pokemon have not fainted and can evolve, they evolve
- Fainted pokemon are returned and a new pokemon is retrieved from the team.
 - If no pokemon can be retrieved (the team is empty), then the opposing player wins. If both teams are empty, the result is a draw.

Note: If an evolved or levelled-up pokemon has a different HP stat to what it had before, you should change its current HP so that the difference between the maximum and current HP is kept the same.

For example, if a level 1 Gastly with 4 HP (maximum of 6) levels up and evolves into a level 2 Haunter (maximum of 10), then the new pokemon has $10 - (6 - 4) = 8$ HP.

 get the lost HP then evolve one new HP minus the lost HP

Attacking

One pokemon attacking another is handled in the following manner:

1. If a pokemon is asleep, stop the attack
2. If a pokemon is confused, see if the defending pokemon should change to be equal to the attacking pokemon (use RandomGen) `RandomGen.randint`
3. Calculate the effective attack based on the attacking pokemons current attack stat, and the type effectiveness of the pokemon matchup.
4. Have the defending pokemon do their defence calculation for effective attack and lose HP accordingly.
5. Lose HP as a result of any relevant status effects
6. See if you should inflict status effects on the defending pokemon (use RandomGen).

Task 1 - pokemon_base.py

TIPS : Abstract classes

Before we can start implementing our Pokemon battles (or even our Pokemon!) we need to set some of the groundwork.

<https://edstem.org/au/courses/8772/discussion/990539?answer=2217086>

In the file `pokemon_base.py`, implement the `PokemonBase` class. This class should implement all features required for battles to work, including but not limited to:

- Fainting
- Health
- Levels
- Stats
- Defending / Attacking
- String representation
- Evolution
- Initialisation (The `__init__` method is *user facing*)

Initialisation should require an initialisation of the max hit points of the pokemon as well as the type of the Pokemon. The pokemon type can be whichever class / inbuilt type you prefer.

See the scaffold and test cases given to get an idea of what functionality / function names are expected.

Task 2 - pokemon.py

Now that we have an interface for our Pokemon to implement, we must implement the Pokemon ourselves.

In `pokemon.py`, implement 10 classes, one for each pokemon. Each class should be a child of the `PokemonBase` class.

Note: The testing requirements for Task 1 and Task 2 only require you to write 3 tests for each method shared across all pokemon classes. So you can have **3 tests for speed**, which checks the speed of 3 different pokemon.

Task 3 - poke_team.py

Now we can get to the meat and potatoes of the tasks!

Each PokeTeam has a **limit of 6 pokemon**, and also has **a battle mode**.

All PokeTeams are **initially** built up of **Charmanders, Bulbasaur, Squirtles, Gastlys and Eevees**. All PokeTeams (except for those in Battle Modes 2 and 3) are initially sorted in the **Pokedex Order**. This order is as follows:

**Charmander, Charizard, Bulbasaur, Venusaur, Squirtle,
Blastoise, Gastly, Haunter, Gengar, Eevee**

The **battle mode** decides the ordering of the party when printing, and how pokemon are **retrieved / returned**. Each battle mode also has a **special action**. There are **4 battle modes** (although **only 3 are required for 1008/2085 students**), which work as follows:



A template to show the ordering of teams

Battle Mode 0

Battle Mode 0



- Battle Mode 0 retrieves the first pokemon in the team, having everyone shuffle up.
- Battle Mode 0 returns the pokemon to the front of the team, having everyone shuffle down.
- Battle Mode 0 special swaps the first and last pokemon on the team.

Battle Mode 1

Battle Mode 1



- Battle Mode 1 retrieves the first pokemon in the team, having everyone shuffle up.
- Battle Mode 1 returns the pokemon to the end of the team, having everyone stay where they are.
- Battle Mode 1 special swaps the first and second halves of the team (the second half includes the middle pokemon for odd team numbers) and reverses the order of the previously front half of the team.

Battle Mode 2

Battle Mode 2



- Battle Mode 2 orders pokemon by a particular criterion, specified on team creation, decreasing. (Ties are broken by the pokemon order. Further ties are broken by their initial ordering of the team).
- Battle Mode 2 retrieves pokemon from the front of the team, having everyone shuffle up.
- Battle Mode 2 returns the pokemon to their correct position within the ordering of the team.
- Battle Mode 2 special reverse the sorting order of the team (so HP increasing becomes decreasing, and if special is used again reverts back to increasing)

Original sorting order = HP low to high

1st trigger's sorting order = HP high to low

2nd trigger's sorting order = HP low to high

3rd trigger's sorting order = HP high to low

etc etc

NOT US :)

Battle Mode 3 (1054)

Battle Mode 3



- Battle Mode 3 orders pokemon the same way as Battle Mode 2
- Battle Mode 3 retrieves the pokemon with the select statistic closest to “Criterion Value” in case of ties, preferring the pokemon closest to the front of the team.
- Battle Mode 3 returns the pokemon to their correct position within the ordering of the team.
- Battle Mode 3 special does nothing.

AI and Team generator

In addition to the battle modes, you also want to give PokeTeams the ability to decide which actions to take in battles (attack, swap, heal, special).

PokeTeams are created with a team name, team numbers, a battle mode, an AI type (Covered in more detail in the requirements section), and any extra criterion information. The

team numbers specify how many charmanders, bulbasurs, squirtles, gastlys and eevees the team has respectively. (Obviously, with the sum of these numbers not exceeding 6) Also, to spice things up, you want to create random teams in the following manner:

1. If no team size is specified, generate a team size of between half of the pokemon limit and the pokemon limit.
2. Create a sorted list, add the values 0 and team size, and generate 4 random numbers from 0 to team size and insert them into the sorted list.
3. For each adjacent value in the list, their difference specifies how many Charmanders/Bulbasurs/Squirtles/Gastlys/Eevees should be added to the team.
4. If no AI mode is specified, the PokeTeam should pick options at random (for all options available, so heal should be removed from the options after 3 heals)

Team Generation

Generate Team Size: 4

Generate 4 Random numbers

from 0 to 4 add to sorted list [0, 4]:

[0, 1, 2, 2, 2, 4]

- Charmanders = 1 - 0 = 1
- Bulbasurs = 2 - 1 = 1
- Squirtles = 2 - 2 = 0
- Gastlys = 2 - 2 = 0
- Eevees = 4 - 2 = 2



Requirements

To class PokeTeam, add the following features:

- AI Enum, which can be used to select which AI to follow.
- `__init__(self, team_name: str, team_numbers: list[int], battle_mode: int, ai_type: PokeTeam.AI, criterion=None, criterion_value=None) -> None`: Creates the PokeTeam. team_numbers is an allowed exception for the use of inbuilt lists. This method is user facing

- **Class method** `random_team(cls, team_name: str, battle_mode: int, team_size=None, ai_mode=None, **kwargs)`: Creates a random team given the rules above
- **Instance method** `return_pokemon(self, poke: PokemonBase) -> None`: return a **pokemon** to the team
- **Instance method** `retrieve_pokemon(self) -> PokemonBase | None`: retrieve a **pokemon** from the team. If the team is empty, return None.
- **Instance method** `special(self) -> None`: Complete the special operation on the team.
- **Instance method** `regenerate_team(self) -> None`: Regenerate the team from the same battle numbers. Used to make a team ready for another battle.
- **Instance method** `is_empty(self) -> bool`: Whether the team is currently empty.
- **Instance method** `choose_battle_option(self, my_pokemon: PokemonBase, their_pokemon: PokemonBase) -> Action`: The AI of the pokemon. Decides on an action depending on the pokemon currently in the field.
- **Logic for four AI modes**:
 - **ALWAYS_ATTACK**: Always choose the attack option
 - **SWAP_ON_SUPER_EFFECTIVE**: Always select swap if the opposing pokemon's attacks are super-effective (will have effective damage larger or equal to 1.5 times their attack stat). Will otherwise attack.
 - **RANDOM**: Will randomly select a valid action.
 - **USER_INPUT**: Will prompt the user for some input.
- `str(my_team)` should return a valid string representation of the team. See testing files for description of format required.

Note: You **are** allowed to edit/add methods to existing ADT classes to achieve some functionality of this task.

Additionally, you **are** allowed to use `ArrayR` for the random team generation algorithm.

Task 4 - battle.py

In `battle.py` you need to implement the battle logic described in the Background section of the assessment. This includes two methods:

- **`__init__(self, verbosity=0) -> None`**: This initialises the class. The verbosity argument is simply an optional switch to possibly enable more printing / logging while battling (Implementing this functionality is optional but recommended, it makes testing way more fun :D).
- **`battle(self, team1: PokeTeam, team2: PokeTeam) -> int`**: Performs the battle between team1 and team2. battle should return 0 1 or 2. 0 representing a draw, and 1 or 2 representing player 1 or player 2 winning respectively.
- Optionally, you can hook up some of the helper functions in `print_screen.py` (namely `print_game_screen`) to make playing the Battles extra immersive. They look like this:

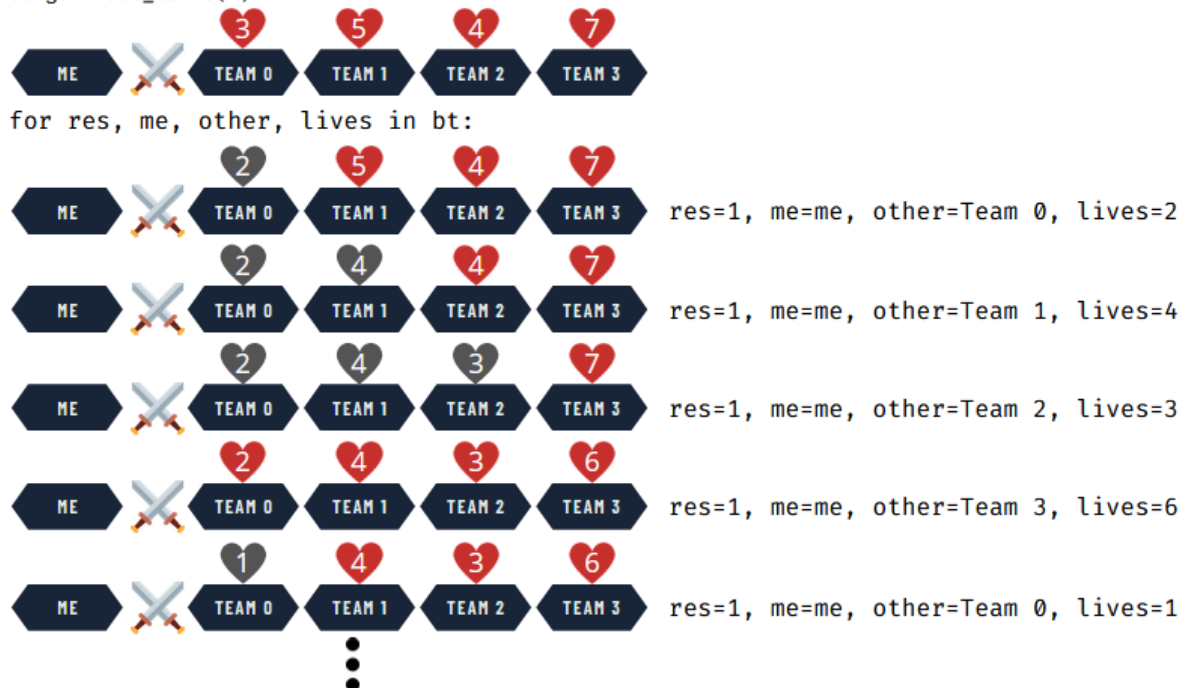
- The tower team after the battle
- The remaining lives of the tower team

Complexity Requirement: $O(B)$, where B is the complexity of battle, in other words, actually retrieving the next battle from the tower should be constant. A tuple may be used to store the final output, but should not be used for anything else.

- `avoid_duplicates(self) -> None`: Removes all currently alive trainers from the battle tower which have multiple pokemon with the same types.

Complexity Requirement: $O(N * P)$, where N is the number of trainers remaining in the battle tower and P is the limit on the number of pokemon per team.

```
RandomGen.set_seed(2)
bt = BattleTower(Battle(verbosity=0))
bt.set_my_team(PokeTeam.random_team("Me", 2, team_size=6, criterion=Criterion.HP))
bt.generate_teams(4)
```



Please see the function `test_duplicates` in `tests/test_tower.py` for more information on `avoid_duplicates`.

1054 - Task 5 Extra

- `sort_by_lives(self) -> None`: Sorts the remaining battle tower trainers by:
 1. The number of lives left, and in cases of equality:
 2. The name of the trainer, lexicographically
 In increasing order. Your approach should not introduce any extra ADTs to achieve this, and should act purely on whatever you are using to store the battle tower trainers.

Complexity Requirement: $O(N^2 * \text{Comp})$, Where N is the number of remaining trainers and Comp is the complexity of comparing two tower teams.

Please see the function `test_sort_lives` in `tests/test_tower.py` for more information on `sort_lives`.

Task 6 - tournament.py

Tournaments can be held in a number of ways. One very popular solution is the single elimination style tournament, where any loss means you exit the competition. One famous example of this format being the Tennis Grand Slams (and more locally, the Australian Open).

Normally, these tournaments are cleanly split into even halves, and the winner of each half plays against one another. However, this need not be the case! In general, we can represent a tournament using the binary operator `+`, which means 'the remaining player from the upper bracket will play off against the remaining player from the lower bracket'. For example, you might represent the [quarter final draw](#) of the latest Australian Open as:

```
((Monfils + Berrettini) + (Shapovalov + Nadal)) + ((Sinner + Tsitsipas) + (Auger-Aliassime + Medvedev))
```

This string is given in infix notation. We can get rid of the brackets if the tournament is stated in postfix notation:

```
Monfils Berrettini + Shapovalov Nadal + + Sinner Tsitsipas +  
Auger-Aliassime Medvedev + + +
```

We wish to do the same thing with pokemon tournaments, where randomly generated PokeTeams face off against one another in a tournament based on a string describing the draw.

Implement the Tournament class which implements the following methods:

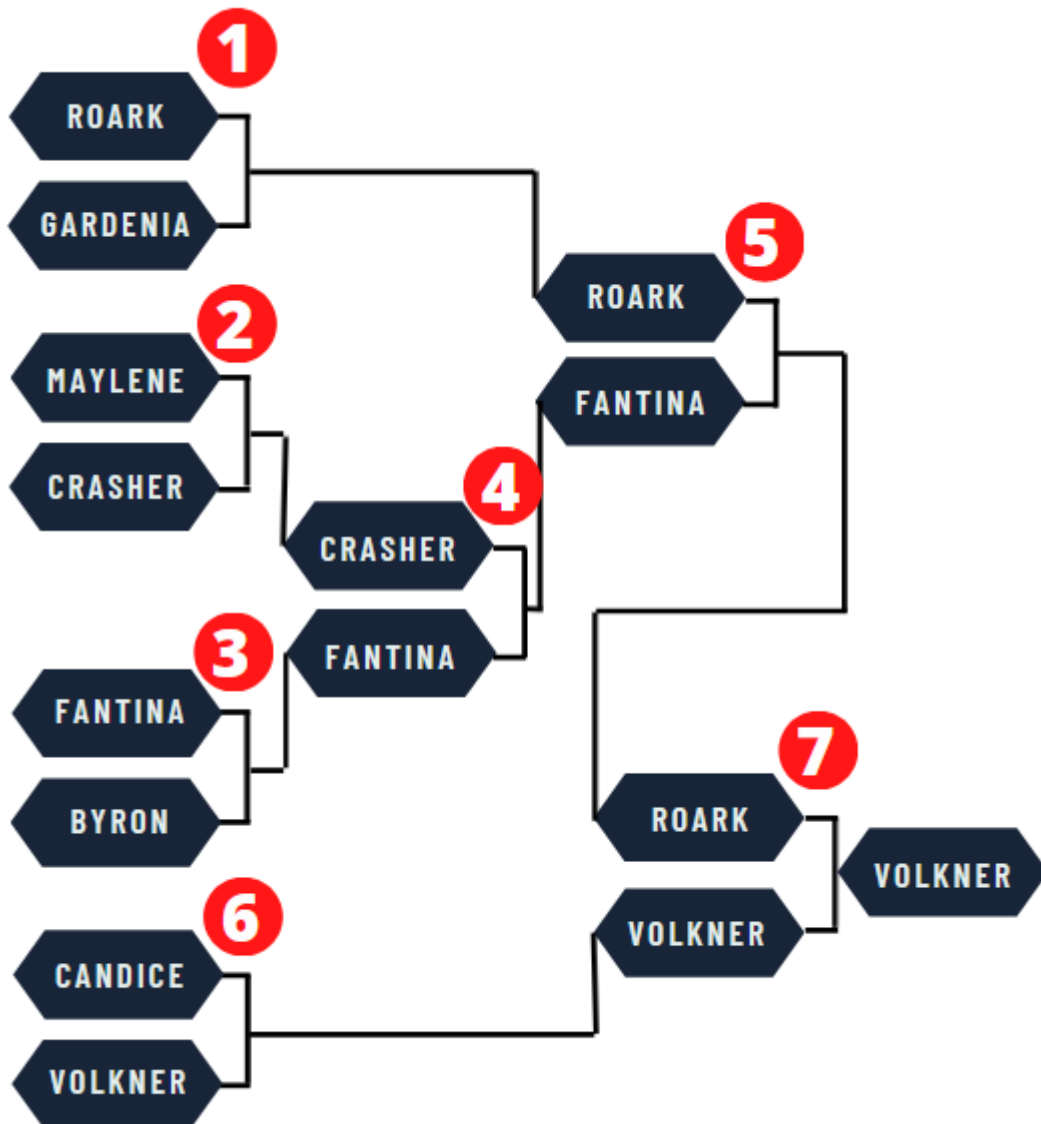
- `__init__(self, battle: Battle | None=None) -> None`: Initialises the tournament class. A battle instance can be passed in, which all tournament games will be run with. If None is provided, a new Battle instance should be created.
- `set_battle_mode(self, battle_mode: int) -> None`: Set the battle mode for all randomly generated teams. This should be called before start_tournament.
- `is_valid_tournament(self, tournament_str: str) -> bool`: Returns true if the tournament_str passed represents a valid tournament.
- `start_tournament(self, tournament_str: str) -> None`: Generates teams based on the tournament_str (it should set the names of each PokeTeam, and use `PokeTeam.random_team()`), but does not play any games. Can do any additional setup required. You **may** use `str.split` for this task, as well as the generated list. This method is *user facing*.
- `advance_tournament(self) -> tuple[PokeTeam, PokeTeam, int] | None`: Simulates one battle of the tournament, following the order of the previously given tournament string (Each + represents a game, and each game is simulated from left to right). If no games are remaining, None should be returned.
Complexity Requirement: $O(B + P)$, where B is the time complexity of running a battle, and P is the number of pokemon in the party. In short, there should not be any more than constant time operations in retrieving the next battle in the tournament.

```

t = Tournament(Battle(verbosity=3))
t.set_battle_mode(1)
t.start_tournament("Roark Gardenia + Maylene Crasher_Wake + Fantina Byron + + Candice Volkner + +")
team1, team2, res = t.advance_tournament() # Roark's Team, Gardenia's Team, 1
team1, team2, res = t.advance_tournament() # Maylene's Team, Crasher_Wake's Team, 2
team1, team2, res = t.advance_tournament() # Fantina's Team, Byron's Team, 1
team1, team2, res = t.advance_tournament() # Crasher_Wake's Team, Fantina's Team, 2
team1, team2, res = t.advance_tournament() # Roark's Team, Fantina's Team, 1
team1, team2, res = t.advance_tournament() # Candice's Team, Volkner's Team, 2
team1, team2, res = t.advance_tournament() # Roark's Team, Volkner's Team, 2
res = t.advance_tournament() # None

```

This represents the following Tournament:



Additionally, players are always interested in what pokemon types are in the meta. To analyse this, one useful thing to note is, at any draw of the tournament, what pokemon types are not present in the current match's teams, but are present in some of the teams defeated by these players.

This is something we'd like you to compute.

- `linked_list_with metas(self) -> LinkedList[tuple[PokeTeam, PokeTeam, list[str]]]`: Returns a linked list, containing a tuple for each match of the tournament, in the same order as `linked_list_of_games` does. Additionally in the tuple is a list of strings representing PokeTypes which, for the particular battle listed, are not present in either PokeTeam, but were represented by some other PokeTeam, which if they hadn't lost any of their matches, would be playing this match.
Complexity Requirement: $O(M * P)$, where M is the total number of matches played and P is the limit on the number of pokemon per team. You can use a tuple/list for the output, but for intermediate calculations please use another ADT.

Please see the function `test_metas` in `tests/test_tournament.py` for more information.

1054 - Task 6 Extra

Implement two methods:

- `is_balanced_tournament(self, tournament_str: str) -> bool`: Returns whether a `tournament_str` is balanced. A tournament is balanced if all matches are between teams that have played the same number of matches previously. The example given above is not balanced, since Roark has only 1 match played when versing Fantina, who had played 2 matches previously.
- `flip_tournament(self, tournament_list: LinkedList[tuple[PokeTeam, PokeTeam]], team1: PokeTeam, team2: PokeTeam) -> None`: The output of `linked_list_of_games` returns the reverse ordering of matches played, so that the final match is shown first, then all matches in the lower bracket, then all matches in the upper bracket, in a recursive fashion. You wonder what the output would look like if, for any match between team1 and team2, you could swap the lower and upper brackets which lead to this match. For example, in the image above, if we called `flip_tournament` with `team1 = Roark` and `team2 = Fantina`, `flip_tournament` should modify the passed in `LinkedList` so that it represents what the output of `linked_list_of_games` would be if the tournament where changed in the following way:

