MONASH University

FACULTY OF INFORMATION TECHNOLOGY

FIT1008/FIT2085 - Introduction to Computer Science

ASSESSMENT 2

Purpose	*To enable you to use and manipulate readily available implementations of Abstract Data Types, implement your own and see how they are applied in practical problem solving. To enable you to analyse the best- and worst-case complexity of an algorithm and apply this knowledge in practice.*
Your task	*In this assessment, you will be given several tasks. In each of them, based on the problem description provided, you will need to implement a Python program that solves the problem. You will also need to test and document your code and analyse its complexity. The problems will need you to deal with object-oriented implementations of Stack, List, Queue ADTs covered in the lessons as well as basic non-recursive sorting algorithms. Some parts of the code functionality may be provided to you, in which case your task will be to complete it.*
Value	**40**% of your total marks for the unit
Word Limit	N/A
Due Date	**1 May 2022 11:55 pm Melbourne Time (AEST)**
Submission	● Via Moodle Assignment Submission. ● Turnitin will be used for similarity checking of all submissions.
Assessment Criteria	Your solution will be assessed using the following criteria: 1. Implementation (40%) a. Correctness b. Quality c. Documentation d. Testing 2. Peer Evaluation (20%) 3. Interview (40%)
Late Penalties	● 10% deduction per calendar day or part thereof for up to one week ● Submissions more than 7 calendar days after the due date will receive a mark of zero (0) and no assessment feedback will be provided.
Support Resources	See Moodle Assessment page
Feedback	Feedback will be provided on student work via: general cohort performance specific student feedback ten working days post submission

FACULTY OF
INFORMATION
TECHNOLOGY

INSTRUCTIONS

Introduction

Aims

- To be able to design and implement in Python basic classes that require simple inheritance.
- To be able to use readily available Stack, Queue, and SortedList ADTs.
- To be able to use and modify the unittest code.
- To continue practising the implementation of correct, high quality and well-documented code.

Important

- 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 any method where you are asked to extend the Unit Testing methods in the test harness, you must include at least 1 extra unittest style assertion test. More info here. More on this can be found in the general constraints.

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 - A01G01_assignment2.zip

The minimum list of files (21) required for this assignment is as follows:

- pokemon_base.py
- pokemon.py
- poke_team.py
- battle.py
  (The base for the rest of these files are provided in the scaffold on Moodle)
- referential_array.py
- stack_adt.py
- queue_adt.py
- array_sorted_list.py

- sorted_list.py
  And all the associated test files for each task as well as the ones given to you.

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.
- 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.
- You must add at least 1 unit test for each method. This means 1 unittest style assertions per method that you write. You should create these tests in a separate unit testing file for each python file that you create. For example -
  `test_pokemon_base.py`
  You should not add any tests to the given test files. Although, if you do, you will not be penalised. However, please make sure that you include the extra test files.
- Please DO NOT modify any of the given methods in any of the ADT files. However, you may add method(s) if you deem it necessary.

Background

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. The teams can be arranged in different formations, and each player can only select up to 6 Pokemon in their team. However, there is a weird constraint where there are only 3 possible Pokemon options to select. These are discussed in detail below:

Pokemon Types and Stats

The following are the stats for each Pokemon:

Name	PokeType	HP	Attack	Defence	Speed	Level	Damage after being attacked

Charmander	Fire	7	6 + level	4	7 + level	1	if damage > defence:
							  HP = HP - damage
							else:
							  HP = HP - damage // 2

Bulbasaur	Grass	9	5	5	7 + level // 2	1	if damage > defence+5:
							  HP = HP - damage
							else:
							  HP = HP - damage // 2

Squirtle	Water	8	4 + level // 2	6 + level	7	1	if damage > defence*2:
							HP = HP - damage
							else:
							HP = HP - damage // 2

Where the column:

- Pokemon Name indicates the name of the Pokemon.
- PokeType indicates the class of the Pokemon. This will be used for Type Effectiveness
- Level indicates the starting level of the Pokemon. This will increase as the Pokemon faints others.
- Attack indicates the attack stat of the Pokemon. This will be used as the 'damage' that a Pokemon deals to another while battling, which may increase on level up.
- Defence indicates the defence of a Pokemon against enemy attacks, which may increase on level up
- Speed indicates the speed of a Pokemon, which may increase on level up
- Damage after being attacked indicates the damage that is being dealt to the Pokemon depending on its defence and the attack damage stat of the enemy Pokemon.

You may be wondering what ‘Type Effectiveness’ refers to. Well, we know that certain elements work better against others. Hence, when a Pokemon of a certain ‘Type’ attacks the other, we apply a damage multiplier to it according to what Pokemon is standing in battle against it. Use the following table as a guide to find out what multiplier to attach:

PokeType	Type Effectiveness (Damage Multiplier)

Fire	Fire: 1 Water: 0.5 Grass: 2
Water	Fire: 2 Water: 1 Grass: 0.5
Grass	Fire: 0.5 Water: 2 Grass: 1

For example - If a fire pokemon attacks a water pokemon, and it has a damage rating of 20, the attack gets cut in half:

*Effective damage = damage * type effectiveness*

Game Structure

Selecting a Pokemon

In this game, we are simplifying to only 3 types of Pokemon so you can have a combination of different numbers of Bulbasaurs, Squirtles and Charmanders.

The game begins with both players constructing their team of Pokemon. A player can choose up to 6 Pokemon and can have multiples of each type of Pokemon.

Positioning/Ordering

After assembling the team, the order of each of the teams looks like in the following example:

1. Charmanders
2. Bulbasaurs
3. Squirtles

The order in this example means that Charmanders will battle first, then Bulbasaurs, and finally the Squirtles. **Note that the actual order will depend on the *battle mode* as defined in tasks 3-5.**

Battle Instructions

After being positioned, the teams battle. Battling involves attacking and defending, after which HP is lost according to the attack stat and the damage after being attacked stat. Remember to use the damage multiplier here!

The battle begins with the first Pokemon of each team (positioned in the right order) getting into battle. The battle between two units (say P1 and P2) proceeds as follows:

- If the speed of unit P1 is greater than that of P2, P1 attacks and P2 defends.
- If the speed of unit P2 is greater than that of P1, P2 attacks and P1 defends.
- If the speeds of P1 and P2 are identical, then both attack and defend simultaneously, regardless of whether one, or both, would faint in combat.

After this initial attack, if the defending Pokemon has not fainted (that is, it still has HP > 0), then they will retort with their own attack to the first Pokemon. Once this has happened, there can only be three scenarios:

1. One of the two Pokemon faints. In this case, the other Pokemon gains 1 level. The other Pokemon then goes back to the team.
2. Both Pokemon faint. In this case, we just leave their carcasses on the battlefield and move on.
3. Both Pokemon are still alive after they have attacked each other. In this case, they both lose 1 HP. If they are still alive after losing this 1 HP, they are sent back to their own teams. If a Pokemon faints here due to losing this 1 HP and the other Pokemon is alive, such a Pokemon still gains 1 level.

Ending the Battle

The game ends when at least one of the teams is empty (i.e., it has no usable Pokemon).

Task Descriptions

Task 1 - ABC & Its Children

Using the information contained in Background, create a file called `pokemon_base.py` that contains an abstract class called `PokemonBase` with three variables called `hp` and `level`, and `poke_type`.

The values for `hp` and `poke_type` will be provided while creating the object, whereas the initial value for `level` is always 1 while creating an object.
You will now need to create a few methods to provide functionality to the base class. You will be using this class as a base for other Pokemon's classes.

NOTE: Please note that some of these methods that you create may be abstract, so please understand the concept of abstract methods and try to create the Abstract Base Class properly using that knowledge.

The following functionality is required to be created:

1. The class should allow the creation of a new Pokemon with the base stats as provided in the information table. The init method must receive the `hp` and `poke_type` in that order.
2. The class should allow the user to get and set the HP and level attributes.
3. Additionally, the class should allow the user to get the name, speed, attack damage and `poke_type` of the Pokemon.
4. The class should allow the user to calculate the damage it takes when being attacked by another Pokemon
5. When printing the object of the class, you should return a formatted string in the following format:
   "<Pokemon_Name>'s HP = <hp> and level = <level>"
   For example - "Charmander's HP = 7 and level = 1"

Once you have created the base class, you should now create another file called `pokemon.py`, which contains three classes, one for each of the Pokemon classes defined in the Background. These classes should be child classes of the Abstract Base class just created and perform all the functions as described above.
HINT - You may need to override some of the methods that you defined as abstract in the first part of this task.

Task 2 - Assembling the Team

What does every Pokemon battle need? A team! Yes, now you will write a class called `PokeTeam` to assemble a team. There are a few constraints that need to be considered while creating a team. We will go through these now.

1. The `team` (needs to be a variable that holds the pokemon objects) can never have more than 6 Pokemon. This is a `limit` that exists for all teams
2. The team should have a `battle_mode` which can only take values from 0,1 or 2. This is explained further in the upcoming tasks.
3. The class should include a method called `choose_team(battle_mode: int, criterion: str = None) -> None` which takes input from the user in the following format:

```
Howdy Trainer! Choose your team as C B S
where C is the number of Charmanders
      B is the number of Bulbasaurs
```

```
      S is the number of Squirtles
>
```

For example:

```
Howdy Trainer! Choose your team as C B S
where C is the number of Charmanders
      B is the number of Bulbasaurs
      S is the number of Squirtles
> 2 3 1
```

This will make a team that contains 2 Charmanders, 3 Bulbasaurs and 1 Squirtle.

4. This class should also include a method called `assign_team(charm: int, bulb: int, squir: int) -> None` which populates the team based on the ADT that is chosen for the `battle_mode` entered by the user.

NOTE: This method should keep asking the user for a valid team until a valid team has been entered. The `criterion` mentioned in this method is a placeholder for a variable that will be used later. You can set the default value to None and just ignore it for now.

5. The class should also set the `battle_mode`, according to whatever the user sets it to. The `battle_mode` is described in the tasks that follow. The default value of `battle_mode` is set to 0. Furthermore, the `battle_mode` cannot be anything other than either 0,1 or 2. All other values are not accepted.
6. If the user wants to print the team, it should print out the Pokemon in the order they would come out.
   For example - If a team contains 1 Charmander 1 Bulbasaur and 1 Squirtle and their stats are the same as when they were created, the following should be the result:

```
print(team)
"Charmander's HP = 7 and level = 1, Bulbasaur's HP = 9 and level = 1,
Squirtle's HP = 8 and level = 1"
```

Task 3 - Set Mode Battle

Now we work with set mode for the battle. In this mode, one Pokemon keeps fighting until it faints. The battle ends when at least one of the teams is empty.
You have been given certain Abstract Data Types (ADTs), one of which can be used for this task. Think about what you need to accomplish and then pick a suitable ADT.

Once you have chosen an appropriate ADT, you need to edit your PokeTeam class to add a method called assign_team, which creates the team in your chosen ADT if the battle_mode is set to 0.

You will now need to create a new Python file called `battle.py`, containing the class `Battle`, which will have the following methods:

1. The constructor:

```
__init__(self, trainer_one_name: str, trainer_two_name: str):
```

This constructor for the class also creates empty teams `team1` and `team2` for `trainer_one_name` and `trainer_two_name` respectively. Note that the method should also create an instance variable `battle_mode` which is set to `None` – its value will be determined at a later stage when an actual battle mode is executed.

2. `set_mode_battle(self) -> str` which asks the user for input and sets up the players' teams in such an order where a pokemon fights until it faints. It is here where you set the team's battle mode to 0. This method should not take any arguments. This method returns the name of the player that wins the battle, `Draw` otherwise

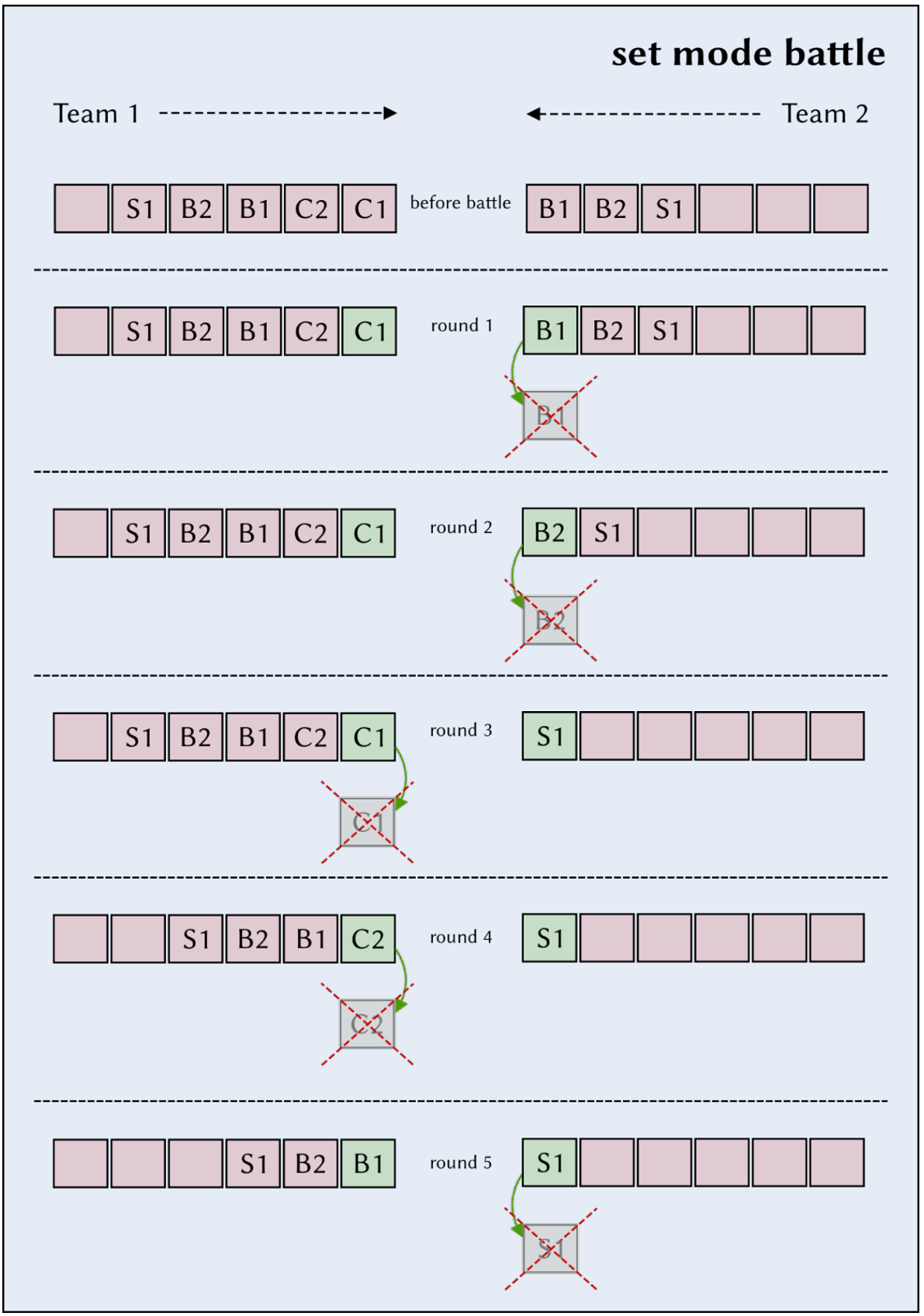
NOTE: The order of the Pokemon in this battle mode MUST be Charmanders -> Bulbasaurs -> Squirtles

An example of a set mode battle would be the following:

Team 1: C1 C2 B1 B2 S1
Team 2: B1 B2 S1

Round 1: Team 1's C1 faints Team 2's B1
Round 2: Team 1's C1 faints Team 2's B2
Round 3: Team 1's C1 is fainted by Team 2's S1
Round 4: Team 1's C2 is fainted by Team 2's S1
Round 5: Team 1's B1 faints Team 2's S1

So, we can see that Team 1's first Charmander keeps fighting until it finally faints in Round 3.


set mode battle
Team 1
Team 2
S1 B2 B1 C2 C1
before battle
B1 B2 S1
round 1
round 2
round 3
round 4
round 5

Task 4 - Rotating Mode Battle

We will now focus on the rotating mode for battle. In this mode, a Pokemon fights a round, and then is sent to the back of the team, making the next pokemon in the party fight the next round. The battle ends when at least one of the teams is empty. You should again choose one of the ADTs provided to you to make this mode work the way it is intended.

Once you have chosen a suitable ADT, you must now edit the `assign_team` method that you made in the previous task to assign the team in this newly selected ADT, if the battle mode is set to 1.

You will also need to edit battle.py to accommodate the rotating mode of battle. You should add a new method called `rotating_mode_battle(self) -> str` which asks the user for input and sets up the players' teams in such an order where a Pokemon fights one round and then gets sent back to its team. It is here where you set the team's battle mode to 1. This method should not take any arguments. This method returns the name of the player that wins the battle, `Draw` otherwise.

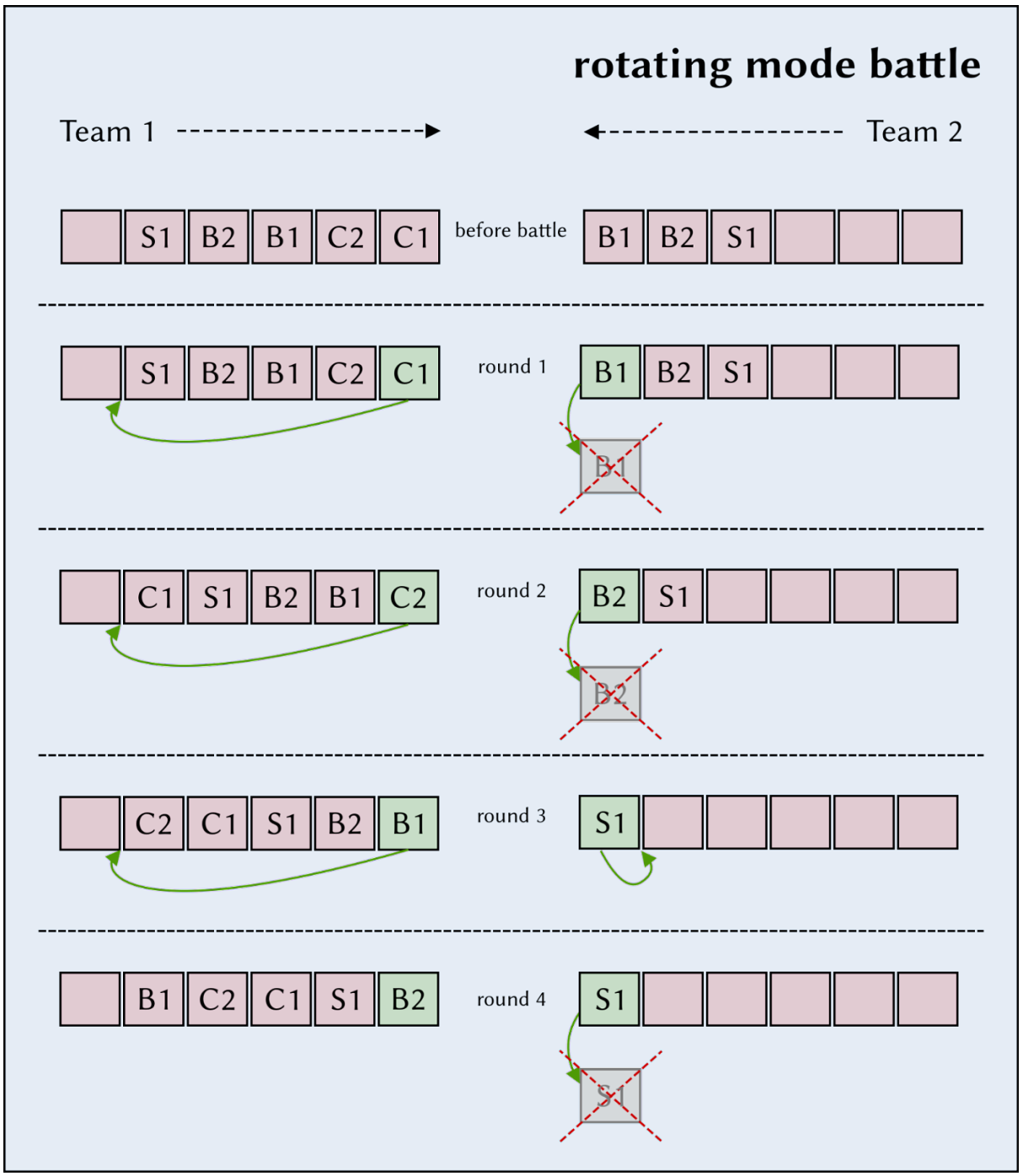
NOTE: The order of the Pokemon in this battle mode MUST be Charmanders -> Bulbasaurs -> Squirtles

An example of a rotating mode battle would be the following:

Team 1: C1 C2 B1 B2 S1
Team 2: B1 B2 S1

Round 1: Team 1's C1 faints Team 2's B1
Round 2: Team 1's C2 faints Team 2's B2
Round 3: Team 1's B1 fights Team 2's S1 and both live
Round 4: Team 1's B2 faints Team 2's S1

So we can see that each round has a different Pokemon fighting, until one team has only one Pokemon left, where that Pokemon keeps fighting until the battle ends.


rotating mode battle
Team 1
Team 2
S1 B2 B1 C2 C1
before battle
B1 B2 S1
round 1
B1 B2 S1
B1
round 2
C1 S1 B2 B1 C2
B2 S1
B2
round 3
C2 C1 S1 B2 B1
S1
round 4
B1 C2 C1 S1 B2
S1
S1

Task 5 - Optimised Mode Battle

Now, things get a little interesting in the optimised mode! In this mode, the user gets to choose an attribute to order their team. They can choose between Level, HP, Attack, Defence and Speed. This order will be maintained throughout the battle, even when the stats change after each round. The battle mode for this mode is 2. And, once again, you need to choose from the given ADTs and once you have chosen a suitable ADT, you need to you need to edit the `assign_team` method to assign the order of the team based on the chosen attribute.

You will also need to edit the `battle.py` file to accommodate the optimised battle mode. You should add a new method called
`optimised_mode_battle(self, criterion_team1: str, criterion_team2: str) -> str`
which asks the user for input and sets up the players' teams in such an order where the Pokemon come in the battle in *non-increasing order* of the chosen attribute by the user. It is here where you set the team's battle mode to 2. This method returns the name of the player that wins the battle, `Draw` otherwise.

NOTE: The order of the Pokemon in this battle mode can vary according to the chosen attribute but it must always be in a non-increasing order. In the case of a tie, it should be in the order of the previous battle modes, that is,
Charmanders -> Bulbasaurs -> Squirtles

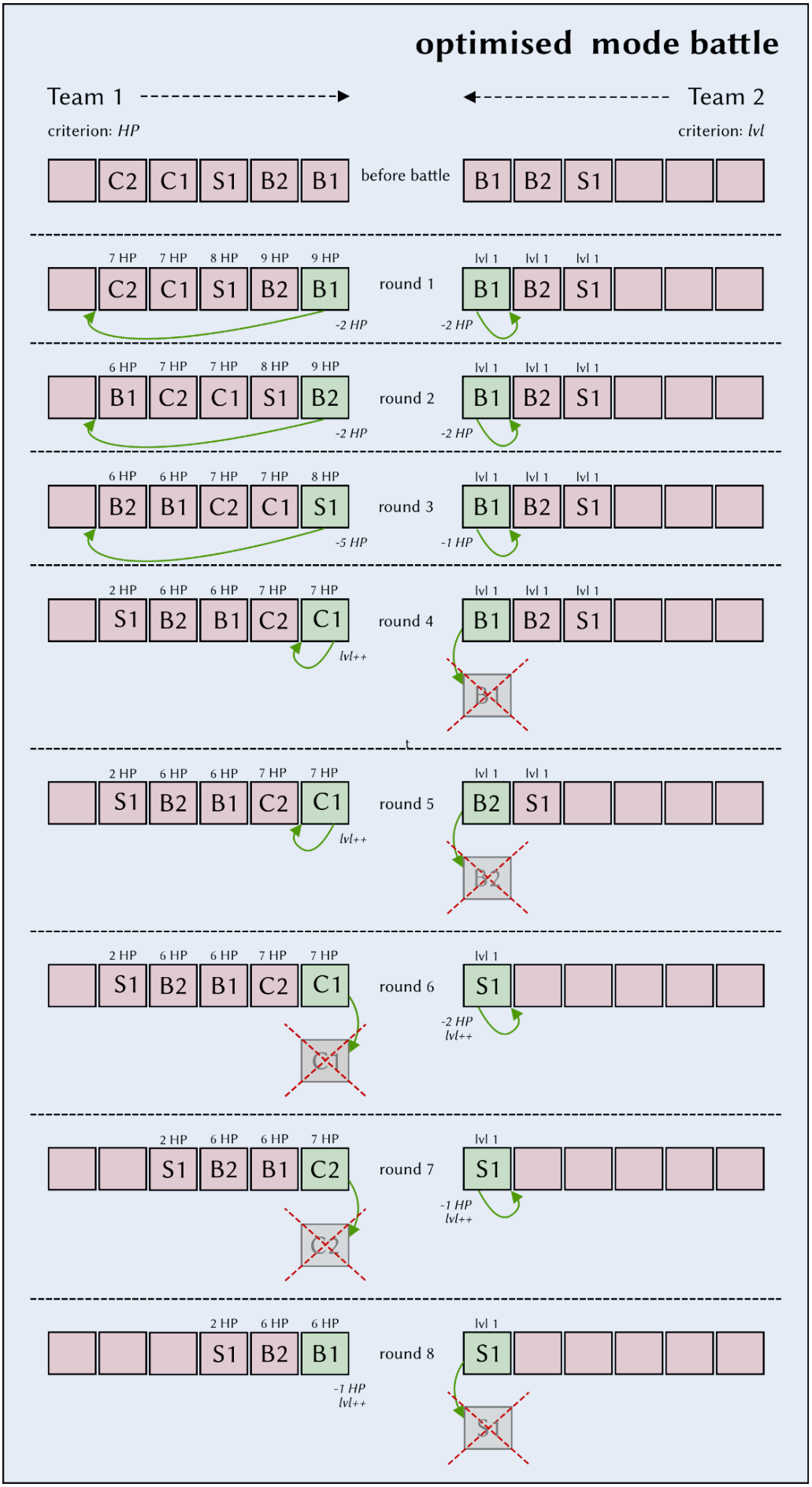
An example of a rotating mode battle would be the following:

Team 1: Chosen Attribute: HP
Team 1: B1 B2 S1 C1 C2

Team 2: Chosen Attribute: Level
Team 2: B1 B2 S1

Round 1: Team 1's B1 attacks Team 2's B1 and they both lose 2 HP
Round 2: Team 1's B2 attacks Team 2's B1 and they both lose 2 HP
Round 3: Team 1's S1 attacks Team 2's B1, loses 5 HP while Team 2's B2 loses 1 HP
Round 4: Team 1's C1 faints Team 2's B1
Round 5: Team 1's C1 faints Team 2's B2
Round 6: Team 2's S1 faints Team 1's C1
Round 7: Team 2's S1 faints Team 1's C2
Round 8: Team 1's B1 faints Team 2's S1

As we can see here and also in the diagram below, the Pokemon in team 1 change their order as their health decreases after each round.


optimised mode battle
Team 1
Team 2
criterion: HP
criterion: lvl
C2 C1 S1 B2 B1
before battle
B1 B2 S1
round 1
7 HP 7 HP 8 HP 9 HP 9 HP
-2 HP
lvl 1 lvl 1 lvl 1
round 2
6 HP 7 HP 7 HP 8 HP 9 HP
round 3
6 HP 6 HP 7 HP 7 HP 8 HP
-5 HP
-1 HP
round 4
2 HP 6 HP 6 HP 7 HP 7 HP
lvl++
round 5
round 6
round 7
round 8

Task 6 - MissingNo Appears!

Wait! Is this a glitch? What's going on?! A mysterious new Pokemon appears. This seems to be completely different to the previous Pokemon and it does not have a Type! All damage is neutral (Type effectiveness multiplier: 1)

You need to define a new class called `GlitchMon` that contains two extra methods in addition to the base class that you created for the previous 3 Pokemon:

- One method to increase the HP
- One method called superpower, which has a random chance to choose one of three effects:
    - Gain 1 level
    - Gain 1 HP
    - Gain 1 HP and 1 level

  This method is called at a 25% chance every time the Pokemon has to defend from an attack.

You should then create another class called `MissingNo` which should be a child of this `GlitchMon` class that you just created. This Pokemon should have stats that are an average of the three Pokemon created previously, but they all scale by 1 after every `level`. This means that each stat increases by 1 every time this monster levels up!
What an overpowered beast! So, we need to include a constraint to the `PokeTeam` class that every team can have a maximum of ONE `MissingNo`. `MissingNo` also comes out for battle *after all the other Pokemon have already battled*.