Harry potter battle

Spell Class

This class is responsible for storing a single spell. It stores its name and power.

```
class Spell:
    def __init__(self, name, power):
        self._name = name
        self._power = power

    def get_name(self):
        return self._name

    def set_name(self, name):
        self._name = name

    def get_power(self):
        return self._power

    def set_power(self, power):
        self._power = power

    def __str__(self):
        return self._name + " " + str(self._power)
```

Wizard Class

This class is responsible for storing wizard information such as his name, health, energy, number of shields and his spells.

```
class Wizard:
    def __init__(self, name, heath, energy, shield):
        self._name = name
        self._health = heath
        self._energy = energy
        self._shield = shield
        self._spells = []

    def get_name(self) -> str:
        return self._name

    def get_health(self) -> int:
        return self._health

    def decrease_health(self, amount: int):
```

```
        self._health -= amount

    def get_energy(self) -> int:
        return self._energy

    def decrease_energy(self, amount: int):
        self._energy -= amount

    def get_shield(self):
        return self._shield

    def has_shields(self) -> bool:
        return self._shield > 0

    def use_shield(self):
        if self._shield > 0:
            self._shield -= 1

    def get_spells(self):
        return self._spells

    def find_spell(self, name: str) -> Spell:
        for spell in self._spells:
            if spell.get_name().lower() == name.lower():
                return spell

    def add_spell(self, spell: Spell):
        self._spells.append(spell)

    def is_dead(self) -> bool:
        return self._health <= 0

    def has_no_energy(self) -> bool:
        return self._energy <= 0

    def has_enough_energy(self, spell_power: int) -> bool:
        return self._energy >= spell_power
```

WizardsBattle Class

This is the main class that is responsible for the game. It initializes the required data such as each wizard. It reads the spells from the file and print the output to xml file.

```
def __init__(self):
    self._harry = Wizard("Harry", 100, 500, 3)
    self._voldmort = Wizard("Voldmort", 100, 500, 3)
    self._load_spells()
    self._xml_root = eT.Element("Game")
```

This is how it reads spells from the file and adds the spells to the appropriate player

```
# Reads the spells from the file and adds them to the player spells
def _load_spells(self):
    with open("spells.txt", "r") as f:
        for line in f:
            buffer = line.split(" ")
            spell_type = buffer[0]
            spell_name = buffer[1]
            spell_power = int(buffer[2])

            if spell_type == "A":
                self._harry.add_spell(Spell(spell_name, spell_power))
                self._voldmort.add_spell(Spell(spell_name, spell_power))
            elif spell_type == "H":
                self._harry.add_spell(Spell(spell_name, spell_power))
            else:
                self._voldmort.add_spell(Spell(spell_name, spell_power))
```

Firstly, We have a method responsible for determining if there is a winner, We have a winner if any of the players is dead or any of them used all his energy. It both players are dead at the same time or used all there energy at the same time we have a draw.

```
def _is_winner(self):
    if self._harry.is_dead() and self._voldmort.is_dead():
        return "\t\tDraw"
    elif self._harry.has_no_energy() and self._voldmort.has_no_energy():
        return "\t\tDraw"
    elif self._harry.is_dead() or self._harry.has_no_energy():
        return "\t\tVoldmort is the winner ..."
    elif self._voldmort.is_dead() or self._voldmort.has_no_energy():
        return "\t\tHarry is the winner ..."
    else:
        return None
```

Secondly, We have a method called start_game() were all the main logic of the game belongs. We start by having an infinite loop and a variable to keep track of the rounds.

```
game_round = 1
game_on = True
while game_on:
# the game logic
```

Then we take input from the user and keep asking him until he enters a valid input

```
# keeps asking the user for input until it is correct
spells = []
wrong_input = True
while wrong_input:
    spells = input("Enter the two spells (harry then voldmort):\n").split(" ")
```

```
    # if the user enters two strings then it is correct
    if len(spells) == 2:
        wrong_input = False
    if wrong_input:
        print("Invalid spell")
```

Then we check if the player owns the spell he enters by searching for them in his spells array if he enters a spell he doesn't own he will be prompted for another input.

```
# gets the spell from the user list of spells
harry_spell = self._harry.find_spell(spells[0])
voldmort_spell = self._voldmort.find_spell(spells[1])

# ensures that the user enters his own spell
if not harry_spell or not voldmort_spell:
    print("Invalid spell")
    continue
```

Then we check that the player has enough energy to use this spell

```
# ensures that the player has enough energy to cast the spell
if not self._harry.has_enough_energy(harry_spell.get_power()):
    print("Harry doesn't have enough energy")
    continue

if not self._voldmort.has_enough_energy(voldmort_spell.get_power()):
    print("Voldmort doesn't have enough energy")
    continue
```

Finally, It the inputs passes all the checks we get the power difference between the two spells

```
# Gets the difference between the two spells
power_difference = abs(harry_spell.get_power() - voldmort_spell.get_power())
```

If a player uses shield we check if he has shields left then we reassign the power difference to 0, so he won't be affected by any damage

```
# when the player uses a shield
if self._harry.has_shields() and (
        harry_spell.get_name() == "sheild" or harry_spell.get_name() == "shield"):
    # Decreases the number of shields
    self._harry.use_shield()
    # Sets the difference to zero so the health won't be affected
    power_difference = 0

if self._voldmort.has_shields() and (
        voldmort_spell.get_name() == "sheild" or voldmort_spell.get_name() == "shield"
```

```
        self._voldmort.use_shield()
        power_difference = 0
```

Then we decrease the power difference from the player that used weaker spell

```
    # Decreases the health of the player that used a weaker spell
    if harry_spell.get_power() > voldmort_spell.get_power():
        self._voldmort.decrease_health(power_difference)
    elif voldmort_spell.get_power() > harry_spell.get_power():
        self._harry.decrease_health(power_difference)
```

Then we decrease the energy of each player according to the used spell

```
    # Decreases the energy of the casted spell
    self._harry.decrease_energy(harry_spell.get_power())
    self._voldmort.decrease_energy(voldmort_spell.get_power())
```

In the end, we add the round results to xml data, by adding a tag for round number than inside it a tag for each player which contains the used spell at this round and his health and energy after this round ends

```
    self._log_results(harry_spell, voldmort_spell, game_round)


    def _log_results(self, harry_spell, voldmort_spell, game_round):
        element1 = eT.SubElement(self._xml_root, "Round_" + str(game_round))

        player1 = eT.SubElement(element1, self._harry.get_name().capitalize())
        spell1 = eT.SubElement(player1, "Casted_Spell")
        spell1.set("name", harry_spell.get_name())
        spell1.set("power", str(harry_spell.get_power()))

        player2 = eT.SubElement(element1, self._voldmort.get_name().capitalize())
        spell2 = eT.SubElement(player2, "Casted_Spell")
        spell2.set("name", voldmort_spell.get_name())
        spell2.set("power", str(voldmort_spell.get_power()))

        eT.SubElement(player1, "Heath").text = str(self._harry.get_health())
        eT.SubElement(player1, "Energy").text = str(self._harry.get_energy())

        eT.SubElement(player2, "Heath").text = str(self._voldmort.get_health())
        eT.SubElement(player2, "Energy").text = str(self._voldmort.get_energy())
```

Then, we print the round results to the console with is each players health and energy

```
    def _print_result(self):
        # Ensures that the number always take 3 spaces on the screen
```

```
        harry_health = str(max(0, self._harry.get_health())).ljust(3)
        harry_energy = str(max(0, self._harry.get_energy())).ljust(3)
        voldmort_health = str(max(0, self._voldmort.get_health())).ljust(3)
        voldmort_energy = str(max(0, self._voldmort.get_energy())).ljust(3)

        print("\t\tHarry\t\tVoldmort")
        print(
            f"Health : {harry_health}\t\t{voldmort_health}")
        print(
            f"Energy : {harry_energy}\t\t{voldmort_energy}")
```

Finally, We check if a user wins and print him to the console then stop the game. Also, it adds the winner to xml data and write xml data to output.xml file. If there is not a winner we start a new round

```
    if self._is_winner():
        game_on = False
        # Announce the winner
        print(self._is_winner())
        # Add Result to xml file
        eT.SubElement(self._xml_root, "Game_Result").text = self._is_winner().strip()
        # Write output to xml file
        with open("output.xml", "wb") as f:
            f.write(eT.tostring(self._xml_root))
```