Lab05A: Building an Arena Battle Simulation in Python

Goal:

Learn to design and implement a simple simulation using classes and objects in Python.

By the end of this lab, you will understand how to define classes, create objects, and make them interact in a basic battle game.

1. Introduction

In this lab, you will design a small text-based battle simulation in which fighters from different historical eras compete in an arena. You will learn to:

- Define Python classes (`Character`, `Arena`).
- Create objects that represent game entities.
- Use methods to model actions (attack, defend).
- Use loops and conditionals to control the flow of a simulation.

2. Step 1 – Create the Character Class

Objective: Learn to define a simple Python class with attributes.

Instructions:

1. Open a new Python file named `game.py`.(or use Jupyter)
2. Copy and run the following code:

```
class Character:
    def __init__(self, name, era, hp, atk):
        self.name = name
        self.era = era
        self.hp = hp
        self.atk = atk
```

3. Add a short test to create one character:

```
if __name__ == "__main__":
    hero = Character("Ambatus", "Lusitanian, II BC", 100, 15)
    print(hero.name, hero.era, hero.hp, hero.atk)
```

Check: You should see Ambatus's details printed correctly.

3. Step 2 – Add the Attack Method

Objective: Allow one character to attack another.

Instructions:

1. Inside the `Character` class, add this method:

```
    def attack(self, other):
        print(f"{self.name} attacks {other.name} for {self.atk} damage!")
        other.hp -= self.atk
```

2. Test it:

```
if __name__ == "__main__":
    ambatus = Character("Ambatus", "Lusitanian, II BC", 100, 15)
    caius = Character("Caius", "Roman, I AD", 120, 12)

    ambatus.attack(caius)
    print(f"{caius.name} now has {caius.hp} HP")
```

Check: You should see that Caius's HP decreases after the attack.

Question: What happens if you attack the same character several times?

4. Step 3 – Add a Method to Check if Alive

4. **Objective:** Let characters know if they still have HP left.

Instructions:

1. Add this method to your `Character` class:

```
    def is_alive(self):
        return self.hp > 0
```

2. Test it:

```
if __name__ == "__main__":
    ambatus = Character("Ambatus", "Lusitanian, II BC", 20, 15)
    caius = Character("Caius", "Roman, I AD", 10, 12)

    ambatus.attack(caius)
    print(f"Is {caius.name} alive? {caius.is_alive()}")
```

Check: When a character's HP drops below 0, `is_alive()` should return `False`.

5. Step 4 – Create a Simple Arena (Two Fighters)

Objective: Make two characters fight automatically.

Instructions:

1. Create a new file called `arena.py`.
2. Add this code:

```
from character import Character
import random

class Arena:
    def __init__(self, fighter1, fighter2):
        self.fighter1 = fighter1
        self.fighter2 = fighter2

    def fight(self):
        print("Battle begins!\n")
        while self.fighter1.is_alive() and self.fighter2.is_alive():
            attacker, defender = random.choice([
                (self.fighter1, self.fighter2),
                (self.fighter2, self.fighter1)
            ])
            attacker.attack(defender)
            print(f"{defender.name} now has {max(0, defender.hp)} HP.\n")

        winner = self.fighter1 if self.fighter1.is_alive() else self.fighter2
        print(f"{winner.name} from {winner.era} wins the duel!")
```

3. Test the arena:

```
if __name__ == "__main__":
    ambatus = Character("Ambatus", "Lusitanian, II BC", 100, 15)
    caius = Character("Caius", "Roman, I AD", 120, 12)
    arena = Arena(ambatus, caius)
    arena.fight()
```

Check: You should see the battle unfold with random attacks until one fighter wins.

6. Step 5 – Expand to Multiple Fighters

Objective: Allow more than two fighters to join the arena.

Instructions:

1. Modify your `Arena` class as follows:

```
class Arena:
    def __init__(self):
        self.fighters = []

    def add(self, character):
        self.fighters.append(character)

    def fight(self):
        print("\nThe Grand Battle Begins!\n")
        alive = [c for c in self.fighters if c.is_alive()]
        while len(alive) > 1:
            for c in alive:
                targets = [e for e in alive if e != c]
                if not targets:
                    break
                target = random.choice(targets)
                c.attack(target)
                print(f"{target.name} now has {max(0, target.hp)} HP
remaining.\n")
            alive = [c for c in self.fighters if c.is_alive()]
        winner = alive[0]
        print(f"\n {winner.name} from {winner.era} wins the battle!\n")
```

2. Create and add multiple fighters:

```
if __name__ == "__main__":
    from character import Character

    fighters = [
        Character("Ambatus", "Lusitanian, II BC", 100, 15),
        Character("Caius", "Roman, I AD", 120, 12),
        Character("Fredrick", "Goth, V AD", 110, 13),
        Character("Ali", "Moorish, IX AD", 100, 14),
        Character("Sancho", "Templar, XII AD", 130, 11),
        Character("Gil", "Sailor, XV AD", 105, 16),
    ]

    arena = Arena()
    for f in fighters:
        arena.add(f)

    arena.fight()
```

Check: The program now simulates a large-scale arena fight, where all characters battle until only one remains.

7. Step 6 – Optional Challenge

Objective: Add new features to make the game more interesting.

Suggestions:

- Add a `defend()` method that reduces damage by half.
- Show a health bar (like `#####-----`) to visualize HP.
- Give random bonuses or abilities to different eras.

8. Reflection

Questions:

- How does each method (`attack`, `is_alive`) change an object's state?
- Why do we use loops inside the `fight()` method?
- How could you make the game results more strategic and less random?