Learning Python Through A Fun Battle Game

IIT Ropar - Minor in AI

1st Feb, 2025

Have you ever talked about something with a friend on WhatsApp – maybe how much you love a particular brand of shoes – and then, *bam*, those same shoes start popping up in ads all over the internet? It's like your phone is reading your mind! You might wonder, is WhatsApp sharing your private chats? Is Google listening in? The truth is, probably not directly. There's something a bit more clever going on called *Federated Learning*. It's a way for computers to learn from data without actually seeing your data.

While we won't be building something as complex as that just yet, we are about to start building our own game. It is so cool that you can learn a lot of concepts of programming and Artificial Intelligence just by building a simple game like that. And today, you will. Yes, you will build your own battle game. And the best part? We're going to do it using just three basic things you probably already know in Python: `if` statements, `for` or `while` loops and `print` statements.

That's right. If you know how to print something on the screen, use `if` conditions and use loops, then you already have everything you need to create your own battle game, analyze winning strategies and even make a simple AI opponent to play against. Sounds amazing, right? Let's jump in!

Figure 1: Let's get started!

Building Our Text-Based Battle Game

Our game is a simple battle between two players. Here's the breakdown:

1. **Two Players:** We need two participants for a battle, which we'll simply call Player 1 and Player 2.
2. **Health:** Each player starts with 100 health points (HP).
3. **Attacks:** Players will take turns attacking each other, causing damage and reducing the opponent's health.
4. **Turn-Based:** The game proceeds turn by turn. Player 1 attacks, then Player 2, and so on.
5. **Text-Based:** This is a text-based game. You will type commands, and the game's results will be shown to you in the terminal or interpreter (Google Colab in our case).
6. **The Goal:** Whoever reaches zero health first loses the game.

Let's start coding. We will be using Google Colab, a free online platform where you can write and run Python code.

Figure 2: The learning begins!

Setting Up the Game

First, let's create the initial variables that hold the health information for our players and set the game up. Let's open up a notebook in Colab and put this code inside a cell:

```
Player1_HP = 100
Player2_HP = 100
```

Here, we've just created two variables: `Player1_HP` and `Player2_HP`, each starting with a value of 100. These represent the health points of each player.

The Game Loop

Next, we need our game to keep running until one player runs out of health. For this, we will use a `while` loop which will continue until one of the player's health becomes zero. Let's add the following code:

```python
turn = 1
while Player1_HP > 0 and Player2_HP > 0:
    print("\n------Turn Start-------")
    print('Player 1 HP:', Player1_HP)
    print('Player 2 HP:', Player2_HP)
```

Let's break this down:

- `turn = 1`: We create a variable called `turn` and set it to `1` initially. This will keep track of whose turn it is.
- `while Player1_HP > 0 and Player2_HP > 0:`: This is our `while` loop. It will keep running as long as both players' health is greater than zero.
- Inside the loop, we print "Turn Start", followed by each player's current health. The \n in `"n------Turn Start-------"` moves the output to a new line so it is more readable.

Determining Who's Turn it Is

Now we need to implement our `if` statement to determine whose turn it is. This part is crucial because it ensures that players take turns. Look at the following code and we will see how it works:

```python
    if turn == 1:
        print("Player 1's Turn")
        input("Press ENTER to attack:")
        damage = random.randint(10, 20)
        print("Generating attack.....")
        time.sleep(2)
        Player2_HP = Player2_HP - damage
        print("Player 1 has attacked Player 2 for", damage, "damage!")
        turn = 2
    else:
        print("Player 2's Turn")
        input("Press ENTER to attack:")
        damage = random.randint(10, 20)
        print("Generating attack.....")
        time.sleep(2)
        Player1_HP = Player1_HP - damage
        print("Player 2 has attacked Player 1 for", damage, "damage!")
        turn = 1
```

Let's understand it, step by step:

- `if turn == 1:`: This is an `if` statement which checks if the `turn` variable is equal to 1. If it is, it means it's Player 1's turn.

- Inside the `if` block:
  - We print `Player 1's Turn`.
  - We get an input from the player, telling them to press ENTER to attack.
  - We generate a random damage amount between 10 and 20 using `random.randint(10, 20)`. To do this, we first need to add the following line of code at the top of the notebook:

    ```
    import random
    import time

    ```

    This imports the necessary `random` library that helps us generate random integers and `time` library to make the delay.
  - We print "Generating Attack", and add a delay of 2 seconds using `time.sleep(2)` – this is simply to make the game feel a bit more realistic.
  - We then reduce Player 2's health by the generated damage: `Player2_HP = Player2_HP - damage`.
  - We print the damage dealt by Player 1 to Player 2 using another print statement and an emoji at the end.
  - Finally, we change `turn` to 2 (`turn = 2`). This ensures that next time, it will be player 2's turn.
- `else:`: If the `turn` is *not* 1, then the code inside the `else` block runs (meaning it is player 2's turn) and Player 2 attacks. The steps are almost the same as for player 1. The health of Player 1 is decreased, and the turn is changed to 1.

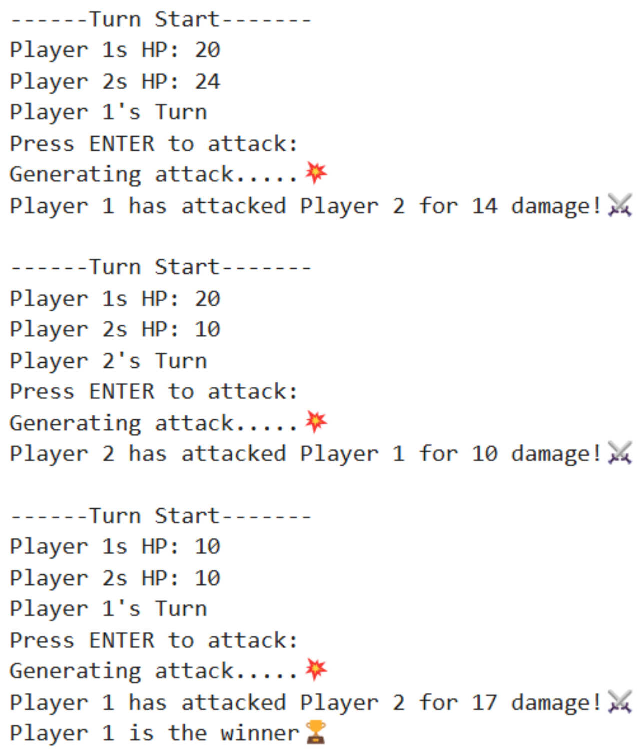
Determining the Winner

After the `while` loop ends, it means that one player has reached zero health. We need to determine the winner:

```
    if Player1_HP > 0:
        print('Player 1 is the winner')
    else:
        print('Player 2 is the winner')
```

If `Player1_HP` is greater than 0, it means that Player 1 is the winner. Otherwise, Player 2 is the winner.

And that's our basic game! You can now copy and paste all of the above blocks of code into a Google Colab notebook, and run the game. You'll see each player attacking turn-by-turn and you will see who wins.



```
------Turn Start-------
Player 1s HP: 20
Player 2s HP: 24
Player 1's Turn
Press ENTER to attack:
Generating attack.....💥
Player 1 has attacked Player 2 for 14 damage!⚔️

------Turn Start-------
Player 1s HP: 20
Player 2s HP: 10
Player 2's Turn
Press ENTER to attack:
Generating attack.....💥
Player 2 has attacked Player 1 for 10 damage!⚔️

------Turn Start-------
Player 1s HP: 10
Player 2s HP: 10
Player 1's Turn
Press ENTER to attack:
Generating attack.....💥
Player 1 has attacked Player 2 for 17 damage!⚔️
Player 1 is the winner🏆
```

Figure 3: A screen shot of a sample output from the game, showing turns, health, and damage.

Making it Fair: Analyzing the Game

Our game is fun, but is it fair? Does one player have an advantage over the other? Let's put our game to the test by simulating multiple game matches. Let's not be human and play this game ourselves, rather let's have the machine play this game over and over again and tell us if this is a fair game.

Let's wrap the game code in a function so that it's easy to repeat multiple times:

```python
def game():
    Player1_HP = 100
    Player2_HP = 100
    turn = 1

    while Player1_HP > 0 and Player2_HP > 0:
        if turn == 1:
            damage = random.randint(10, 20)
            Player2_HP = Player2_HP - damage
            turn = 2
        else:
            damage = random.randint(10, 20)
            Player1_HP = Player1_HP - damage
            turn = 1

    if Player1_HP > 0:
        return 'Player 1'
    else:
        return 'Player 2'
```

Now let's write some code to simulate 10000 games:

```
num_games = 10000
Player1_wins = 0
Player2_wins = 0

for i in range(num_games):
    winner = game()
    if winner == 'Player 1':
        Player1_wins = Player1_wins + 1
    else:
        Player2_wins = Player2_wins + 1
```

Here, we're running the `game()` function 10,000 times. We are keeping count of how many times player 1 has won and how many times player 2 has won. At the end, we'll have a count of wins for each player.

To visualize this, we will create a bar graph using `matplotlib`, a Python visualization library. Let's add this code at the end:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(5, 5))
x_axis = ['Player 1', 'Player 2']
y_axis = [Player1_wins, Player2_wins]

plt.bar(x_axis, y_axis)
plt.ylabel('Number of wins')
plt.title('Game Simulation Results')
plt.show()
```

This code will generate a bar graph showing the number of wins for each player. If you run this code, you will see that the game is biased towards player 1, as the game begins with his turn.

Making an AI Opponent: Strategic Choices

Now, let's make our game more interesting by adding an AI opponent. Instead of us choosing to attack, the computer will make a decision to attack or defend for us, making it the AI opponent.

To do this, we need to add another action: the option to *defend*. Defending will reduce the damage taken by 50 percent. We will add two new variables at the beginning of our `game()` function:

```
Player1_defending = False
Player2_defending = False
```

These variables will keep track of whether a player is currently defending. We'll then update our code to ask for input whether to attack or defend:

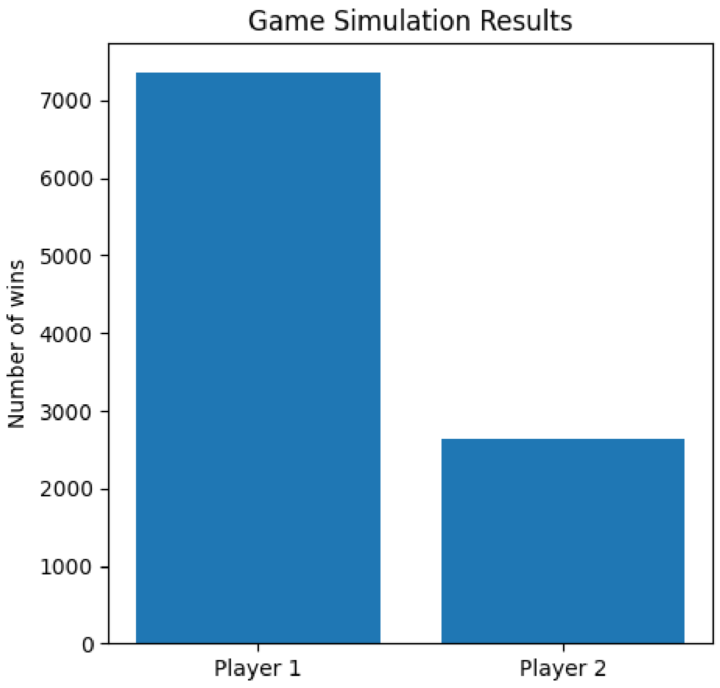


Figure 4: A sample bar chart showing the number of wins for each player, with player 1's bar much bigger than that of player 2.

```
    if turn == 1:
        print("Player 1's Turn")
        action = input("Choose attack or defend: ")
        if action == 'attack':
            damage = random.randint(10, 20)
            print("Generating attack.....")
            time.sleep(2)
            if Player2_defending:
                damage = damage // 2
                Player2_defending = False
            Player2_HP = Player2_HP - damage
            print("Player 1 has attacked Player 2 for", damage, "
    damage!")
        else:
            Player1_defending = True
            print("Player 1 is defending")
        turn = 2
```

If the player chooses to attack, the code does what it did previously – deal a random damage between 10 and 20 to the other player. The only difference is now that if the other player was defending, we halve the damage being dealt. If the player chooses to defend, then that player's `Playerx_defending` variable is set to `True`, to indicate that that player is defending. And when it's the next player's turn, that defending variable gets set to `False`, no matter what the other player chooses to do, whether to attack or defend. The damage is only halved if one of the players was defending *during* the other player's attack.

We need to also update this same code for Player 2. And finally, to create our AI opponent, we will write a function for the computer's choice:

```
def computer_choice(Player1_HP, Player2_HP, Player1_defending,
    Player2_defending):
    if Player1_defending:
        return 'attack'
    elif Player2_defending:
        return 'attack'
    elif Player1_HP < 30:
        return 'attack'
    elif Player2_HP < 20:
        return 'defend'
    elif random.random() < 0.7:
        return 'attack'  # 70% chance to attack
    else:
        return 'defend'  # 30% chance to defend
```

Here, the computer makes a decision to either attack or defend. Let's understand the code:

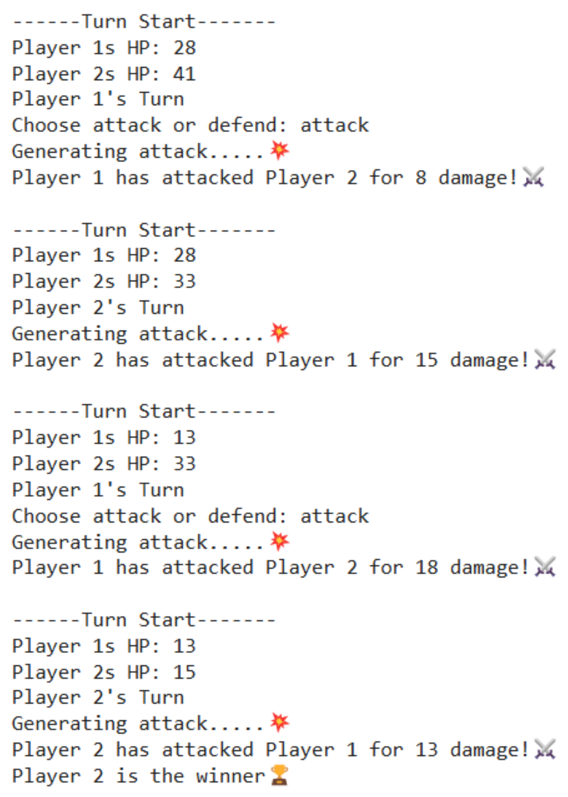
- First, if either of the players is defending, it chooses to attack.
- If Player 1 has less than 30 HP, it chooses to attack.
- If Player 2 has less than 20 HP, the computer chooses to defend.
- Otherwise, it randomly chooses to attack with a 70% chance, and defend with a 30% chance.

Now let's modify the player 2 input part to include this computer choice:

```
    else:
        # Player 2's turn
        print("Player 2's Turn")
        action = computer_choice(Player1_HP, Player2_HP,
    Player1_defending, Player2_defending)
        if action == 'attack':
            damage = random.randint(10, 20)
            print("Generating attack.....")
            time.sleep(2)
            if Player1_defending:
                damage = damage // 2  # Integer division
                Player1_defending = False
            Player1_HP = Player1_HP - damage
            print("Player 2 has attacked Player 1 for", damage, "
   damage!")
        else:
            Player2_defending = True
            print("Player 2 is defending")
        turn = 1
```

In the modified code above, we're calling our `computer_choice` function which takes the relevant parameters and makes a choice for player two. We no longer require user input here, since the computer chooses the action.

Run this code. You will be able to play the game against your new AI opponent.



```
------Turn Start-------
Player 1s HP: 28
Player 2s HP: 41
Player 1's Turn
Choose attack or defend: attack
Generating attack.....💥
Player 1 has attacked Player 2 for 8 damage!⚔️

------Turn Start-------
Player 1s HP: 28
Player 2s HP: 33
Player 2's Turn
Generating attack.....💥
Player 2 has attacked Player 1 for 15 damage!⚔️

------Turn Start-------
Player 1s HP: 13
Player 2s HP: 33
Player 1's Turn
Choose attack or defend: attack
Generating attack.....💥
Player 1 has attacked Player 2 for 18 damage!⚔️

------Turn Start-------
Player 1s HP: 13
Player 2s HP: 15
Player 2's Turn
Generating attack.....💥
Player 2 has attacked Player 1 for 13 damage!⚔️
Player 2 is the winner🏆
```

Figure 5: Sample output screenshot for the improvised version of the game!

And there you have it! You've built a text-based battle game, analyzed its fairness, and created a simple AI opponent, all using the basic building blocks of Python: `if` statements, `while` loops, and `print` statements.

You've come a long way! This is just the beginning of your journey. You're well on your way to learning the fundamentals of Python and Artificial Intelligence through engaging and fun methods. Remember, keep experimenting and keep asking questions, and you can do a lot of cool things just using this basic knowledge. Keep practicing!