UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

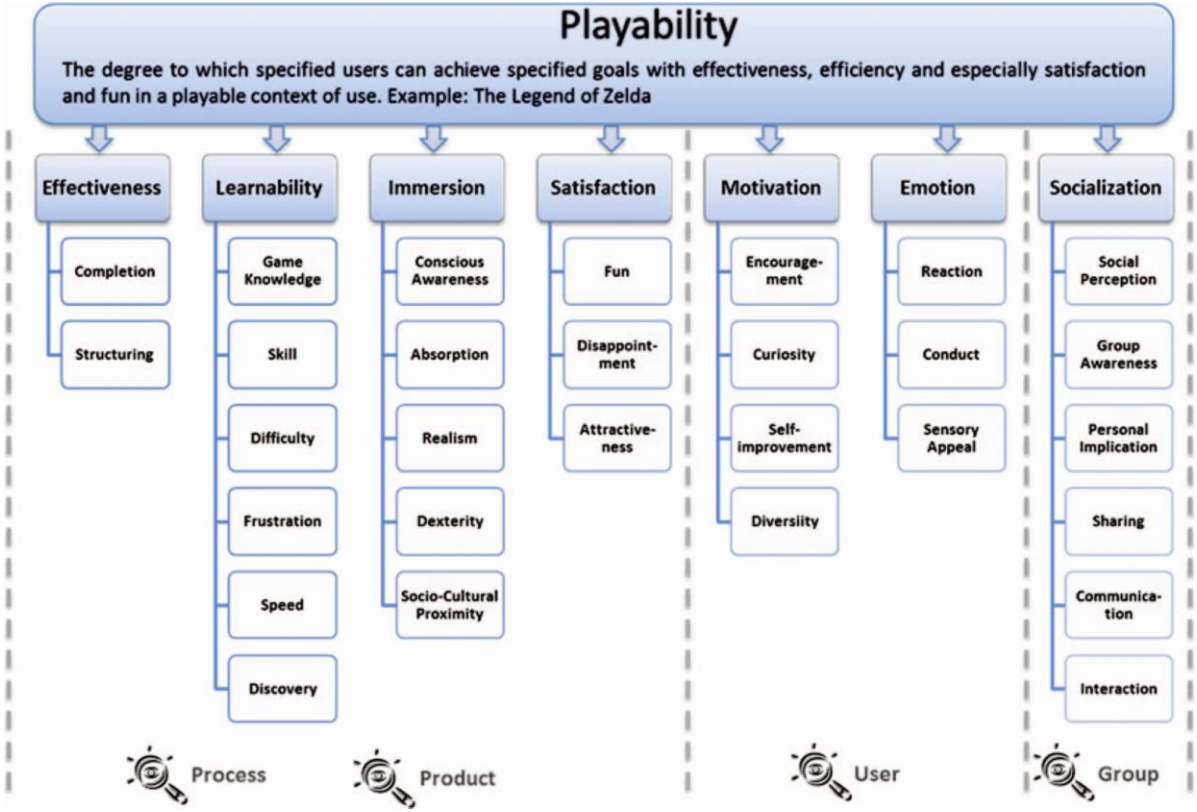
Python Programming for Data Science and Engineering

Ph.D. School in Information Engineering
A.Y. 2024/2025

Stefano Tortora
stefano.tortora@unipd.it
Intelligent Autonomous Systems Laboratory (IAS-Lab)
Dept. of Information Engineering (DEI)
University of Padua, Italy

Playability

Playability is the ease by which the game can be played or the quantity or duration that a game can be played and is a common measure of the quality of gameplay.
Playability is defined as a set of properties that describe the Player eXperience (PX) using a specific game system.



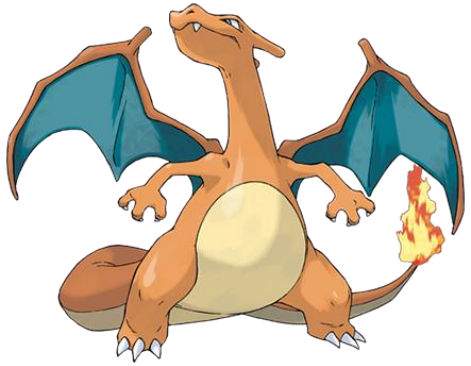
[1]José Luis González Sánchez , Francisco Luis Gutiérrez Vela, Francisco Montero Simarro & Natalia Padilla-Zea (2012) Playability: analysing user experience in video games, Behaviour & Information Technology, 31:10, 1033-1054, DOI:10.1080/0144929X.2012.710648

Game difficulty

The degree of difficulty in a video game is a key factor in playability, because it encourages players to go ahead in the game or not.
Difficulty may be higher or lower depending on the level of the player.
A high Difficulty level can provoke disappointment and less satisfaction as well as a low Difficulty level.

We have a simple game: no realistic 3D GUIs, no immersive storytelling.
We can stay simple and think about just 2 levels:

Novice user: 80% of possibilities to win a battle by using random attacks



Skilled user: 60% of possibilities to win a battle by using random attacks

Assignment 3

The aim of **Assignment 3** is to identify the wild Pokemon opponents during the exploration in order to reach the appropriate *Difficulty level*. Thus, building on top of what you have done in the previous assignments, to complete this assignment you must:

1. Load *Pokemons*, *Moves* and *Type effectiveness* from file
2. Handle the *Type effectiveness* in the useMove() method
3. Develop a *Random Battle Mode* to collect data
4. Analyse and plot the results

Explanations on how to perform these steps are provided in the following slides.

Attention! For this and the following assignments, organize your Python project with its own virtual environment and upload the assignment with the virtual environment included in the .zip file.

Load data from file

At the program start, your program must load the information used by the game from files.

In the *Additional Materials & Links* section of the Moodle course, you can find three files in the *.json* format containing, respectively:

- The list of Pokemons with their stats (**pokemons.json**)
- The list of Moves with their stats (**moves.json**)
- The list of Type effectiveness for each pair of types (**type_effectiveness.json**)

When loading the data, make the following assumptions:

- ❑ When loading Pokemons, assume each pokemon to have "Level" equal to 1
- ❑ When loading Moves, exclude all the moves with "power" equal to `null` and do not consider the "effect", "effects" or "changes" fields
- ❑ Add to each Pokemon at least two moves randomly selected with the following constraints:
  - A pokemon can have a move only if pokemon "types" contain the move "type"
  - Every pokemon can have "normal" type moves

Manage JSON files

JSON (JavaScript Object Notation) is a popular data format used for representing structured data. It's common to transmit and receive data between a server and web application in JSON format.

In Python, JSON exists as a string. For example:

```
p = '{"name": "Bob", "languages": ["Python", "Java"]}'
```

To work with JSON (string, or file containing JSON object), you can use Python's `json` module. You need to import the module before you can use it, open the file in 'reading' option and parse each line of the file containing a JSON string.

```
import json

pokemons = dict()
with open('pokemons.json', 'r') as file:  # open the file containing the data
    for line in file:
        p = json.loads(line)              # convert each json line into a dictionary
        pokemons[p['name']] = p           # save data into a unique container
                                          # e.g., a dictonary with 'name' as keys
```

Type effectiveness

The type effectiveness must be considered when calculating the damage in the useMove() method and it is included as one of the terms of the modifier.

$$modifier = stability * effect * critical * luck$$

To consider the type effectiveness in the damage calculation, you should check the move type (*attack*) and the defending pokemon type (*defend*), and look to the "effectiveness" field extracted from the **type_effectiveness.json** file at game start.

```
# Examples of effectiveness
{"attack": "normal", "defend": "fire", "effectiveness": 1}      # Standard effect
{"attack": "fire", "defend": "water", "effectiveness": 0.5}     # Not very effective
{"attack": "fire", "defend": "grass", "effectiveness": 2}       # Super effective!
{"attack": "electric", "defend": "ground", "effectiveness": 0}  # No effect…
```

Attention! Beware that, since the defending Pokemon can have more the one type, the type effectiveness must 'stack up', e.g., If the attacker uses a move of type "fire" against a pokemon of types ["water", "grass"]:

```
effect = effectiveness["fire"]["water"] * effectiveness["fire"]["grass"]
       = 0.5 * 2 = 1
```

Random Battle Mode

In order to set the difficulty level, we will face the problem through a data-driven approach.
To do so, modify your previous program as follow:

- **Automatic exploration-only story**
  Allow only the "Explore" action in the Story with 100% probability to find a random wild pokemon. Automatically "Exit" the game after having performed *Nbattles* battles.
- **Automatic random battles**
  Develop a demo battle mode in which the player fights with the opponent using only random moves (no items, no pokemon change, no runaway) until one of the two is defeated.
  After each battle, go to the Pokemon Center before going on with the Story.
- **Save the statistics**
  Run the game several times (e.g., *Ngames* = 500 or more) for each starter pokemon (you can also include Pikachu if wanted) and with a high-enough number of battles (e.g., *Nbattles* = 150 or more). Keep track of:
  - the list of encountered wild pokemons,
  - the list of victories and losses,
  - the total number of turns in each battle,
  - the percentage of residual player's pokemon HPs at the end of each battle.

Note: in the Random Battle Mode DO NOT consider the PP value of the moves!

Manage data savings

For general saving of data on your disk, Python provides a useful library for saving any Python object by serialization, that is called **Pickle**.

Pickling is the serializing and de-serializing of python objects to a byte stream. Unpickling is the opposite.

```
import pickle

## Saving a generic Python object (e.g., a dictonary) in a pickle file
example_dict = {1:"6",2:"2",3:"f"}
pickle_out = open("dict.pickle","wb")    # open a file in "write byte" mode
pickle.dump(example_dict, pickle_out)    # save the object inside the file
pickle_out.close()                       # remember to close the file when finished

## Loading data from pickle file
pickle_in = open("dict.pickle","rb")   # open a file in "read byte" mode
example_dict = pickle.load(pickle_in)  # retrieve the object from the file
pickle_out.close()
```

Analyse & Plot the results

Display a set of graphs showing the results of the tests on random battles.

- Simple plot
  1. For each starter pokemon, display in the same graph the cumulative number of victories over the *Nbattles* battles averaged across the *Ngames* games.
- Box Plot
  1. Display the distribution of the number of turns in each battle.
  2. Display the distribution of the residual player's pokemon HPs at the end of each battle.
  3. Compute and show the distributions mean, median, 25th quartile and 75th quartile.
- Bar charts (a graph for each starter pokemon)
  1. For each unique enemy pokemon encountered show the percentage of player's victories.
  2. For each unique enemy pokemon encountered show the mean and standard deviation of the residual player's pokemon HPs at the end of each battle in the same graph.

  In the same figure, highlight the enemy pokemons that would be more appropriate for each user according to the following division:
  - **Novice user:** 80±10 % victories AND mean player's pokemon HPs at the end of each battle greater than 70%
  - **Skilled user:** 60±10 % victories AND mean number of turns in a battle greater than the median of turns distribution