Part 1: create the Maze class

- Create a class `Maze` in a file `maze.py`.
- The constructor of this class receives a filename in argument.
  - the file is opened, and all lines are read
  - the lines represent the structure of a maze
  - a space ( ) is an "empty space"
  - a `X` is a wall (cannot go through)
  - set up a structure to store the information about the maze
- Create a method `check` that receives two arguments: a line number, and a column number.
  - It returns a boolean:
    - `True` if the maze has an "empty space" in the position (line_number, column_number).
    - `False` otherwise
- Create a method `display` that prints the structure of the maze on the screen, using `print`.
- Create a method `find_random_spot` that returns a tuple (line_number, column_number) which is an "empty space" in the maze structure.
- What is the most computationally / memory efficient way of doing this? Discuss in the forums or on Discord!

Part 2: Improve the maze game

A. Add random items to the maze

- Randomly select 4 empty spots in the maze, and put objects instead.
- Note: each object is different.
- Rename the `check` method, and call it `can_move_to`:
  - if the location requested is a wall, return `False`
  - otherwise, return `True`
- Add a new method `is_item`:
  - if the location requested is a random item, return `True`

B. Setup the player

- In the maze file, choose a character to represent the starting point of the player (for example: `P`).
- Create a `Player` class in a `player.py` file.
  - it has an attribute: `backpack`, which will contain the random items picked up along the way.
- Adjust the maze so that an instance of the `Player` class is created, and tracks the location of the player in the maze.

C. Make sure the game ends

- In the maze file, choose a character to represent the exit of the maze (for example: `E`).

- Add a new method `is_exit` to the `Maze` class. It returns `True` if the location requested is the exit point.