Dungeon Version 4: Final version

Let's add some final touches.

In this version, you will implement:

- Displaying map with nice Unicode characters
- Checking the win condition (whether the player reached the finish)
- Showing help

What to do

In addition to the functions you already have:

1. Update the `display_map` function so that it displays the map using Unicode characters:

- Start (S): 🏠
- Finish (F): 🏺
- Dash (-): 🧱
- Path (*): 🟢
- Player (@): 🧝

Example of the original map:

```
**S--
*----
@**--
--**F
```

Should be displayed as:

🟢🟢🏠🧱🧱
🟢🧱🧱🧱🧱
🧝🟢🟢🧱🧱
🧱🧱🟢🟢🏺

2. Implement the `check_finish` function that would check whether the player reached out the finish and return either `True` or `False`.

3. Implement the `display_help` function that would show the contents of help.txt.

4. Update the `main` function:

   a) At the end of each step, the program should check whether the player reached the finish. If so, it should print **Congratulations! You have reached the exit!**
   b) For the **help** command, the program should display help (contents of the help.txt file).

5. **(optional)** Refactor your `main` function by moving the game logic (the suite of the game loop) into a separate `make_step` function with the following signature:

```
def make_step(grid: list[list[str]], player_position: list[int, int]) -> bool:
    """
    Makes a single step of the game.
    """
```

Use the following template. All functions defined in the template **must be present and implemented** in your code (you may **not** omit functions or change the given function definitions.). You **may** add extra functions if needed.

```
MAP_FILE = 'cave_map.txt'
HELP_FILE = 'help.txt'

def load_map(map_file: str) -> list[list[str]]:
    """
    Loads a map from a file as a grid (list of lists)
    """
    # Implemented in version 1

def find_start(grid: list[list[str]]) -> list[int, int]:
    """
    Finds the starting position of the player on the map.
    """
    # Implemented in version 1

def get_command() -> str:
    """
    Gets a command from the user.
    """
    # Implemented in version 1

def display_map(grid: list[list[str]], player_position: list[int, int]) -> None:
    """
    Displays the map.
    """
```

```
    # TODO: update this function


def get_grid_size(grid: list[list[str]]) -> list[int, int]:
    """
    Returns the size of the grid.
    """
    # Implemented in version 2

def is_inside_grid(grid: list[list[str]], position: list[int, int]) -> bool:
    """
    Checks if a given position is valid (inside the grid).
    """
    # Implemented in version 2

def look_around(grid: list[list[str]], player_position: list[int, int]) -> list:
    """
    Returns the allowed directions.
    """
    # Implemented in version 2

def move(direction: str, player_position: list[int, int], grid: list[list[str]]) ->
bool:
    """
    Moves the player in the given direction.
    """
    # Implemented in version 3

def check_finish(grid: list[list[str]], player_position: list[int, int]) -> bool:
    """
    Checks if the player has reached the exit.
    """
    # TODO: implement this function

def display_help() -> None:
    """
    Displays a list of commands.
    """
    # TODO: implement this function

def main():
    """
    Main entry point for the game.
    """
    # TODO: update the main() function

if __name__ == '__main__':
```

```
    main()
```

Hints

- `display_map` refactoring: try using a nested data structure to save the emojis in your code. Like so:

  `emojis = [(" ", "🧱"), ("S", "🏠"), ("F", "🏺"), ("*", "🟢")]`
- `check_finish`: remember that the player finishes when they reach the letter **'F'** in the map
- For the Optional function `make_step` function consider where your conditional statements should be, and where the loop should be in the program.

Program name

Save your program as `dungeon4.py`.

Demo

In this demo, `cave_map.txt` is used.

https://asciinema.org/a/e7m9NcOCEkUG0H1XYny6rKBPe

Testing

To make sure your program works correctly, you should test it.

Good news: we wrote the unit tests for you: **test_dungeon4.py**

To test your functions, simply run the unit tests:

```
$ python -m pytest test_dungeon4.py
```

All tests should pass.

Submitting

Submit `dungeon4.py` via eClass.