Programming on Purpose (CSC 120)
Winter 2024

Project 2 – Igel Ärgern (New and Improved!)

Due: 2/8/2024

In this project, you will refactor and expand on your implementation of "Igel Ärgern" from Project 1.

1 Objectives

- Practice refactoring.
- Practice information hiding.
- Practice object-oriented programming.
- Practice testing.

2 Your Mission

Your mission is two-fold: improve your implementation and expand its functionality.

2.1 Refactor

- Critically review your project along all dimensions we have discussed: understandability, coherency, modularity, reusability, robustness. In particular, examine whether there are ways to improve modularity by introducing additional helper functions or additional modules. For example, since you will have to expand on the code that gets input from the user, it might be useful to have the code for getting input from the user encapsulated in helper functions.
- Make your project object oriented. That is, your program should be a collection of classes.

2.2 Expand

- We are no longer assuming that the players are cooperative and intelligent. So your implementation should check whatever input they provide to ensure that it is valid and possible given the current state of the board.
- Add traps.

PITFALL ALERT! Never try to *refactor* and *expand functionality* at the same time. Do one and *then* the other.

3 Starter Code

I am providing you with some files to help you refactor your program into an object-oriented one. Note that this may mean that you are ignoring this starter code in the beginning (see guidance below). It only becomes relevant once you have reached the point where you start to work on making your implementation object oriented.

- `igel_view.py`: This module implements the same terminal based user interface that we used for Project 1. However, this new implementation is object oriented. You should not modify this file. **All** interaction with the players should be through the methods `refresh`, `request_input`, `inform`.
- `igel_aergern.py`: Starter code for an object oriented implementation of the game's main logic.
- `board.py`: Starter code for an object oriented implementation of a model of the "Igel Ärgern" board.
- `testing.py`: Utility functions for testing.

4 Requirements

1. As before, **all** writing to the terminal or reading input from the terminal has to be done using the functions `refresh`, `request_input`, `inform` provided in the module `igel_view.py`.
2. Your implementation may have more classes, but at the very least it should have the following:
   - `Board` (defined in module `board.py`), which models an "Igel Ärgern" board.
   - `IgelAergern` (defined in module `igel_aergern.py`, which models one game of "Igel Ärgern".
   - And of course `IgelView` (defined in module `igel_view.py`), which models the "Igel Ärgern" interface and is given as starter code. Please do not modify this file.
3. The `Board` class must contain the following methods (it may have more).

```
def __init__(self, board_str=""):
    """Initialize a Board object representing an empty board (default). If a value
    is provided for board_str, the board is initialized to reflect that string
    representation of the board.
    """
```

Notice that the `__init__` method has an optional parameter. If you have not worked with optional parameters before and did not read the textbook section on it during the lab in Week 4, read textbook Section 9.18 now. During the normal running of the game, that optional `board_str` parameter will never receive a value because we start the game with an empty board. The purpose of this optional parameter is to be able to create a specific board configuration for testing purposes.

The `IgelAergern` class provides a method called `test_run_game` that you can use for testing. For example, the following snippet of code should run and should allow you to start your game with a predefined board configuration.

```
igel_game = Igel_Aergern()
init_board_str = "P|R||@|||||\\||PY||||@||\\|||||@RYP||||\\||Y|||@|||\\||@|R||R|||\\Y||P|||||@|"
igel_game.test_run_game(init_board_str, 3)
```

```
def as_str(self):
    """Return a string representation of the current state of the Board object.
       The string representation is suitable for being passed to the refresh
       method of a IgelView object.
    """
```

That means that the string representation returned by the `as_str` method for an empty board with traps should look as follows. Note the use of `@` to indicate the traps.

```
"|||@|||||\\\|||||@||\\\|||@||||\\\|||||@|||\\\||@||||||\\\|||||||@|"
```

The string representation returned for a board with some tokens should look as follows. Note the tokens in the third row that are stuck in a trap.

```
"P|R||@|||||\\\||PY||||@||\\\|||||@RYP||||\\\||Y|||@|||\\\||@|R||R|||\\\Y||P|||||@|"
```

```
def move_forward(self, row, col):
    """Move the token that is at the top of the stack in the given row and
       column one step forward.
       Parameters:
       row : int
           the row number; between 1 and 6
       col : int
           the column number; between 1 and 9
    """

def move_sideways(self, row, col, dir):
    """Move the token that is at the top of the stack in the given row and
       column one step sideways in the given direction.
       Parameters:
       row : int
           the row number; between 1 and 6
       col : int
           the column number; between 1 and 9
       dir : int
           either 1 (down) or -1 (up)
    """

def get_winner(self):
    """Check whether there is a winner with the current state of the Board.
       If not, return None. If so, return the winning player's color. I.e.
       Y', 'R', 'G', etc.
    """
```

4. You also have to create a module `board_tests.py` which runs a test suite for all public methods of your `Board` class. That is, all methods that are called from outside of the `Board` class, e.g. from the module `igel_aergern.py`.

   This test suite should be built using the utility functions provided in `testing.py`. See the example code from Friday of week 2 on Nexus.

5 Some Guidance

- **How do I start?** Start by critically reviewing your implementation from Project 1. It might be helpful to print it out for this purpose. Look for anything you could improve. But especially look for opportunities to improve information hiding by modularizing the code with additional helper functions or additional modules. For Project 2, I am prescribing that you have a separate module to represent the board. If you did not have that module in your implementation of Project 1, analyze how you would need to change the structure of your implementation to create a separate module to represent the board. Then refactor accordingly.

Once you are ready to start refactoring, make a copy of Project 1 and use the copy as the place to start Project 2. If things start going south, you can always go back to your working version of Project 1.

- **When do I start?** Right now! Here is one suggestion for how you could break this project into smaller chunks:

  1/26–1/29:
  - Review your code.
  - Refactor for better readability, coherency, modularity, reusability.

  1/30–2/2:
  - Refactor to make your implementation object-oriented.
  - Add tests as you go along.

  2/3–2/8:
  - Add traps.
  - Add input validation.

6 Getting Ready to Submit

Remember: *Just because it works doesn't mean it's good.*

- Make sure that all modules and functions are documented.
- Re-read your code and check for ways in which readability could be improved (e.g., good variable/function names, no magic numbers, consistent formatting, code is organized into logical units, each function serves one main purpose). Have a look at the official Python Style Guide: `https://peps.python.org/pep-0008/`.
- Don't forget to cite who you received help from and include the honor code affirmation at the top of each Python file you have written or edited:

```
I affirm that I have carried out my academic endeavors
with full academic honesty. [Your Name]
```

7 How to submit

Submit the project by uploading all Python files necessary to run your implementation to Gradescope (including the starter files).

At the very least, your submission should include the following files, but you are likely to have more:

- `igel_aergern.py`
- `board.py`
- `igel_view.py`
- `testing.py`
- `board_tests.py`