WORDLE crate exercises

STEP 0: Getting started

Start by cloning the wordle repository, branch `tutorial`:

```
$ git clone https://github.com/mosteo/wordle_ada --branch tutorial
```

The new folder `wordle_ada` contains the root `wordle` crate on which the steps take place.

STEP 1: Fixing initial dependencies.

File to edit: `wordle_ada/alire.toml`

At this moment, you won't be able to build or edit the project, because there are missing dependencies. Check the `[[depends-on]]` entry, you will see a dependency.

- **TODO**: Verify this from the command-line by running `alr with --solve`

You should get a report where the missing libraries are labeled as such. This dependency isn't available in any index. It is only available in a remote repository, so we will add it as a development "pin", as the first step.

- **TODO**: add this dependency as a remote pin. You can do it in a single step with the `alr with` command. Check its help, or the slides dealing with pins. The syntax to use is :
  - `alr with --use <url>`
  - The location of the dependency is: https://github.com/mosteo/wordlelib_ada

When you add this pin, you'll get a report with **two** new dependencies. Where is the second one coming from? There is a clue in the report itself.

After adding the pin, check back the contents of the file. You should see a new `[[pins]]` entry at the end. If for some reason the pinning didn't work properly, you can remove the `[[pins]]` section and try again, or fix it by hand and run `alr update`

- **TODO**: the project should build properly now. Try it with `alr build`
- **TODO**: Also you can play already. Try it with `alr run`

You will see an obvious limitation after a couple of games. We will fix this in a later step.

As you will have seen, the output format is monochrome and not very clear. We will improve this output in the next step.

- **TODO**: edit your project. To use GNAT Studio, run `alr edit`. To use VSCode, set up the environment in advance with `alr printenv`, and then launch the editor from the terminal.

Step 2: Enable the use of colors.

File to edit: `wordle_ada/src/wordle.adb`

There is an ANSI crate providing color sequences for most usual terminals. If you are on Windows 10, this should work too. Otherwise you can skip to step 4 at the bottom of this file.

More info: https://stackoverflow.com/questions/51680709/colored-text-output-in-powershell-console-using-ansi-vt100-codes

Despite the Windows consoles supporting ANSI colors, there is a current limitation in `alr` that requires extra steps. See the annex at the end of this file.

- **TODO**: add the 'ansiada' dependency. This crate is available normally in the community index, so you only need to `alr with it.
- **TODO**: after adding it, you need to close and reopen your editor; otherwise the environment won't contain the new crate and the build in GPS Studio will fail.
- **TODO**: look for the "Print_Char" procedure (around line 100). Continue with STEP 3 from there.

STEP 3: Using ANSI colors

File to edit: `wordle_ada/src/wordle.adb`

- **TODO**: Uncomment the commented out implementation of the Print_Char procedure, and comment the current implementation that isn't using Ansiada.
- **TODO**: also add/uncomment "with AnsiAda;" at the top of this file.
- **TODO**: execute the game with the new implementation and verify colors are now showing. This would be also a good time to inspect the spec in wordlelib.ads if you're curious about how track of the game is being kept.

STEP 4: Using a random word

As you'll have noticed, we are now playing using always a predefined word. To fix this issue we are going to use a wordlist, which is available in the crate with the same name. We could simply pin it, but to spice things a bit up, we'll first use the indexed version in a personal index of mine.

- **TODO**: verify "wordlist" is not yet available, running one of:
    - `alr show wordlist`
    - `alr search --crates wordlist`
- **TODO**: add a new index, found at https://github.com/mosteo/alire-personal-index . To do so, you can use the `alr index` command. Check its help, or the slides that explain how to use it. The syntax to use is:
    - `alr index --add <url> --name <name>`
    - The name is arbitrary (e.g., you can use "personal" or "mosteo").
- **TODO**: verify the index is properly added by running `alr index`
- **TODO**: check that the "wordlist" crate is available now with `alr show <crate name>`
- **TODO**: add the crate as a dependency with `alr with`. Don't forget to close and reopen the editor afterwards.

STEP 5: Random word selection

File to edit:

In this step we will apply the random word selection, and we will inspect a few details of the 'wordlist' crate.

- **TODO**: Look for the "Word_To_Guess" procedure in `wordle.adb`, near the top of the file (around line 40). You'll see that there is a few commented lines, and also the return

with the predefined word. Uncomment the block that gets a random word. Inspect the spec at wordlist.ads is you feel curious about the offered API.
- **TODO**: `alr run` to verify that now you get a random word every time.
- **TODO** : in `wordle.adb`, in the procedure "User_Input", uncomment the few commented lines that check that the user-provided word does indeed exist.

The wordlist crate makes use of the resource sharing facilities in Alire: the list is not hardcoded in Ada sources, but is loaded at elaboration time from a TOML file. By default, the "large.toml" file is loaded.

- **TODO**: locate the resource files of this crate navigating in the console or a file browser to the `./wordle_ada/alire/cache/dependencies/wordlist.../share/wordlist` folder. Take a peek at the file contents if you feel curious.

A problem with this wordlist is that it includes many obscure words. In the next step we will see how to use the same wordlist that is used in the original Wordle game.

To do so, we are going to use a development version of the 'wordlist' crate. We will pin it in the way you're already familiar:

- **TODO**: pin the 'wordlist' crate, using the branch 'tutorial':
  - `alr pin wordlist --use https://github.com/mosteo/wordlist_ada --branch tutorial`
- **TODO**: verify the pin by running `alr pin`. Observe how the tutorial branch is indicated, by contrast to the 'wordlelib' pin which tracks the default branch.
- **TODO**: reopen your editor with `alr edit` to apply the new environment...

STEP 6: debug the wordlist crate

The wordlist crate uses the code generation configuration facilities of Alire. In this step we will use these to debug the crate and, later, to switch the wordlist.

- **TODO**: open the manifest of the wordlist crate, found at:
  - `wordlist_ada/alire/cache/pins/wordlist/alire.toml`

Observe the `[configuration.variables]` section, where two variables are defined. One of those is for logging, the other is to select the wordlist to be loaded. You may also see step 7 at the end of this file, but **don't start with it yet**.

- **TODO**: enable logging of the crate. Todo so, edit the alire.toml **of the wordle_ada** folder (that is, the one we have at the top of the working tree) to override the value of the Logging variable. To do so, add:

```
[configuration.values]
wordlist.Logging = true
```

to the `alire.toml` at `wordle_ada/alire.toml`

- **TODO**: build everything with `alr build` run in the `wordle_ada` folder.
- **TODO**: locate with your editor the `wordlist_config.ads` file, and verify that the `"Logging"` variable has the expected value. Check how the enum for the available wordlists is also defined there.
- **TODO**: `alr run` and observe a new message before the game starts, such as:

```
Loading words from /path/to/wordlist.json
Loaded 370105 words
```

STEP 7: use the original Wordle wordlist.

The problem with the default wordlist is that it contains too many obscure words. On the other hand, the small wordlist available above in the configuration variables, is so limited that most usual words are not recognized and so it is very difficult to progress.

If you look into the `wordlist_ada/share/wordlist` folder, you'll see a third wordlist in the file `"wordle.toml"`. We could edit the sources to hardcode that file, but instead, we will add this third possibility via the crate configuration in the crate manifest.

- **TODO**:
  - In the manifest at `wordle_ada/alire/pins/wordlist/alire.toml`:
  - Edit the "Wordset" entry in the `[configuration.variables]` table in this file, add `"wordle"` as a third value in the `"values"` array.
- **TODO**: Edit the root manifest at `./wordle_ada/alire.toml`, to select the `"wordle"` wordset.
  - To do so, add a new table `[configuration.values]` to the manifest, with an entry "`Wordset = "wordle"`"
- **TODO**: `alr run` the game again and verify that the number of loaded words is now smaller. You can also inspect the generated configuration file found at `wordle_ada/alire/cache/pins/wordlist_ada/config/wordlist_config.ads` to double check. Finally, you can inspect the source code of `wordlist.adb` to check how this generated configuration is used during the loading of the words.

THE END

This is the end of the Wordle tutorial exercise. Congratulations!

Annex: running Wordle with colors on Windows

Running executables through `alr run` will cause the terminal to fail to recognize ANSI sequences. This can be worked around by running executables directly, e.g., `bin\wordle.exe` from the `wordle_ada` folder.

This fails to take into account the resource locator of Alire. From step 5 onwards, before running the executable directly, one must set up the environment in advance with:

`alr printenv --powershell | Invoke-Expression`

If the environment changes (due to changes in dependencies, pins, or crate configuration), this command must be run in a new console (otherwise conflicts with already set variables with cause the invocation to fail.)