EECS3311 Fall 2019
Lab Exercise 4
Extending the Chess-Solitaire Game with Undo/Redo Functions using ETF (the Eiffel Testing Framework)

Chen-Wei Wang

Common Due Date for Sections A & E: 2pm, Monday, November 25

– **Check the Amendments section of this document regularly for changes, fixes, and clarifications.**

– **Ask questions on the course forum on the moodle site for Sections A and E.**

1 Policies

– **Your (submitted or un-submitted) solution to this lab exercise (which is not revealed to the public) remains the property of the EECS department. Do not distribute or share your code in any public media (e.g., a non-private Github repository) in any way, shape, or form. The department reserves the right to take necessary actions upon found violations of this policy.**

- You are required to **work on your own** for this lab. **No** group partners are allowed.
- When you submit your lab, you claim that it is **solely** your work. Therefore, it is considered as **an violation of academic integrity** if you copy or share **any** parts of your Eiffel code during **any** stages of your development.
- When assessing your submission, the instructor and TA may examine your code, and suspicious submissions will be reported to the department/faculty if necessary. **We do not tolerate academic dishonesty**, so please obey this policy strictly.

– You are entirely responsible for making your submission in time.

- You may submit **multiple times** prior to the deadline: only the last submission before the deadline will be graded.
- Practice submitting your project early **even before it is in its final form**.
- No excuses will be accepted for failing to submit shortly before the deadline.
- Back up your work **periodically**, so as to minimize the damage should any sort of computer failures occur. Follow this tutorial series on setting up a **private** Github repository for your Eiffel projects.
- The deadline is **strict** with no excuses: you receive **0** for not making your electronic submission in time.
- Emailing your solutions to the instruction or TAs will not be acceptable.

– You are free to work on this lab on your own machine(s), but you are responsible for testing your code at a Prims lab machine before the submission.

Contents

2 Learning Outcomes of this Lab Exercise

1. Implement the Undo/Redo Pattern.
2. Work with a separation of abstract user interface and business model logic.
3. Design your own classes.
4. Write unit tests to verify the correctness of your software.
5. Draw professional architectural diagram using the draw.io tool.

3 Required Tutorials

1. **Tutorial Videos on ETF**
2. Link to a Written Tutorial: **Tutorial on ETF: a Bank Application**

4 Working from Home

- To generate the ETF project, you must use the `etf` command that is available on your Prism account.
- For a generated ETF project to compile on your machine, you need to first download a library called `MATHMODELS`, and then set a environment variable `MATHMODELS` which points to the location of its download. See your Lab0 instructions for details.

5 Grading Criterion of this Lab

– When grading your submission (separate from the BON diagrams), your ETF project will be compiled and built from scratch, and then executed on a number of acceptance tests (similar to `at01.txt`, `at02.txt`, . . . given to you).

– Each acceptance test is considered as **passing only if** the output generated by your program is character-by-character identical to that generated by the *oracle*. No partial marks will be given to a test case even if the output difference is as small as a single character.

– It is therefore critical for you to always switch to the command-line mode of your ETF project and use either `diff` or `meld` to compare its output and that of the *oracle*.

6 Problems

This lab exercise builds upon your work of the previous lab (i.e., the chess-solitaire game implemented using the ETF tool). All requirements of the game remain the same as stated in Lab 3 instructions, and in **basic and grading tests used to asses your Lab 3**.

6.1 Abstract User Interface

For this lab, we assumed the following extended abstract user interface:

```
system chess_solitaire_undo

type CHESS = {
  K, -- King
  Q, -- Queen
  N, -- Knight
  B, -- Bishop
  R, -- Rook
  P  -- Pawn
}

setup_chess(c: CHESS; row: INTEGER; col: INTEGER)
        -- Set up by adding a chess at (‘row‘, ‘col‘).
start_game
        -- Start the game after setting up chess pieces.
reset_game
        -- Start to setup a new game.

move_and_capture(r1: INTEGER; c1: INTEGER; r2: INTEGER; c2: INTEGER)
        -- Move the chess at (‘r1‘, ‘c1‘), in a way that is valid,
        -- and capture the chess at (‘r2‘, ‘c2‘).

moves(row: INTEGER; col: INTEGER)
        -- Show all possible moves of the chess located at (‘row‘, ‘col‘),
        -- including those that are not valid.

-- These are new events for this lab.
-- undo, redo features
undo
redo
```

That is, there are two events `undo` and `redo` added to the abstract user interface, so you will need to regenerate a fresh ETF starter project for this lab exercise.

To understand the rationale behind these two new events, you are required to study **Chapter 21 from OOSC2** on a design pattern supporting the undo/redo features. This pattern exploits the object-oriented notions of polymorphism and dynamic binding. **Although Chapter 21 from OOSC2 contains obsolete syntax of Eiffel, you can easily find the syntactic alternatives from the lecture notes.** You can access an electronic copy of OOSC2 from the course moodle page.

6.2 Outputting the Abstract State

The abstract state to be displayed between user-initiated events is similar to that in Lab 3. Study the three example acceptance tests `at04-undo.txt`, `at09`, and `at10.txt` given to you. All error messages required by Lab 3 are still required by this lab exercise (the same set of acceptance tests `at01.txt` to `at08.txt`, given to you for Lab3, are again given to you). There are two additional error messages:

Message	Triggering Events
`Error: Nothing to undo`	`undo`
`Error: Nothing to redo`	`redo`

Table 1: Extended Messages: String Values and Triggering Events

6.3 Requirements

While you have freedom on designing classes under the `model` cluster, your implemented classes **must** satisfy the following requirements:

- You must support the undo/redo features by following the pattern discussed in Chapter 21 from OOSC2. $\boxed{\text{You receive a \textbf{zero} for this lab if you do not follow the undo/redo pattern covered in OOSC2.}}$
- In the `model` cluster, you must implement an inheritance hierarchy of commands (for the chess-solitaire game) as discussed on $\boxed{\text{page 700 in Chapter 21 of OOSC2}}$.

  Remark: Do not get confused with these "model" commands (which are created in order to implement the undo/redo pattern) and the "user" ETF commands (which are generated to handle user events in, e.g., the batch mode). You should still only put error-message-related code in these ETF command classes.

6.4 Hints

- Once the game is restarted, the user cannot roll back, via `undo`, to before the game was restarted. See `at04-undo.txt`.
- Recall from Lab 3 that all ETF command classes have the singleton access (declared in the `ETF_COMMAND` class) to the same "model object". You may also want all "model" command classes (created for the undo/redo pattern) to have the same singleton access.

7 Getting Started

First of all, make sure you have already acquired the basic knowledge about the Eiffel Testing Framework (ETF) as detailed in Section 3.

Download the **`chess_solitaire_undo.zip`** file from the course moodle page and unzip it (e.g., into a directory `Lab_4`). The text file `extended-chess-solitaire-events.txt` is for you to generate the ETF project for your extended Chess-Solitaire game application. The eleven input files **example** use cases for you to test your software (where `at01.txt` to `at08.txt` are identical to those given to you in Lab 3, and `at04-undo.txt`, `at09.txt`, and `at10.txt` are examples of the undo/redo features). The eleven expected output files (e.g., `at01.expected.txt`, `at02.expected.txt`, ..., `at10.expected.txt`) contain outputs that your software must produce to match. You are advised to, before start coding, study the given expected output files carefully, in order to obtain certain reasonable sense of how your extended Chess-Solitaire game app is supposed to behaviour.

All your development will go into this downloaded `chess_solitaire_undo` directory, and when you make the submission, you must submit this directory. To begin your development, follow these steps:

1. Open a new command-line terminal. Change the current directory into this downloaded `chess_solitaire_undo` directory, type the following command to generate the ETF project:

   ```
   etf -new extended-chess-solitaire-events.txt .
   ```

   Notice that there is a dot (`.`) at the end to denote the current directory.

2. There are two `chess_solitaire_undo` directories: one is the top-level, downloaded directory that contains all generated ETF code and your development; the other is the sub-directory that contains the `model` cluster. **When you submit, make sure that you submit the top-level `chess_solitaire_undo` directory.**

3. Open the generated project in Eiffel Studio by typing:

   ```
   estudio19.05 chess_solitaire_undo.ecf &
   ```

4. Once the generated project compiles successfully in Eiffel Studio, go to the `ROOT` class in the `root` cluster. Change the implementation of the `switch` feature as:

```
switch: INTEGER
    -- Running mode of ETF application
  do
-- Result := etf_gui_show_history -- GUI mode
   Result := etf_cl_show_history
-- Result := unit_test -- Unit Testing mode
  end
```

   This overrides the default GUI mode of the generated ETF. To make it take effect, re-compile the project in Eiffel Studio.

5. Switch back to the terminal and type the following command:

   ```
   EIFGENs/chess_solitaire_undo/W_code/chess_solitaire_undo
   ```

   Then you should see this output (rather than launching the default GUI of ETF):

   ```
   Error: a mode is not specified
   Run 'EIFGENs/chess_solitaire_undo/W_code/chess_solitaire_undo -help' to see more details
   ```

6. As you develop your ETF project for the extended Chess-Solitaire game, launch the batch mode of the executable. For example:

   ```
   EIFGENs/chess_solitaire_undo/W_code/chess_solitaire_undo -b at01.txt
   ```

   This prints the output to the terminal. To redirect the output to a file, type:

   ```
   EIFGENs/chess_solitaire_undo/W_code/chess_solitaire_undo -b at01.txt at01.actual.txt
   ```

   The `at01.actual.txt` file stores the *actual* output from your current software, and your goal is to make sure that `at01.actual.txt` is identical to `at01.expected.txt` by either typing:

   ```
   diff at01.expected.txt at01.actual.txt
   ```

or typing:

```
meld at01.expected.txt at01.actual.txt
```

Of course, the actual output file produced by the default project is far from being identical to the expected output file.

7. You should first aim to have your software produce outputs that are identical to those of the eleven expected output files (i.e., `at01.expected.txt`, `at02.expected.txt`, . . . , `at10.expected.txt`).

8. Then, as you develop further for your ETF project, create as many acceptance test files of your own as possible. Examine the outputs and make sure that they are consistent with the requirements as stated in this document.

9. A few days before the lab is due, you will be given an *oracle* program for you to test if your software and the oracle produce identical outputs on all of your acceptance test files. **You certainly want to finish all your development before the oracle program is made available to you, so that if you find any inconsistencies of outputs, you still have sufficient time to debug and fix.**

8 Modification of the Cluster Structure

You must not change signatures of any of the classes or features that are generated by the ETF tool. The only exception is the `model` cluster: you may only rename `ETF_MODEL` and `ETF_MODEL_ACCESS`, or add your own clusters or classes to the `model` cluster as you consider necessary.

When you add a new cluster, simply create a subfolder inside `model` (from the file system) and re-compile. **You are responsible for testing if your ETF project compiles in a Prism machine.**

9 Two BON Diagrams to Submit

See Section 10 for details.

10 Submission

10.1 Checklist before Submission

1. Make sure the `ROOT` class in the `root` cluster has its `switch` feature defined as:

```
switch: INTEGER
    -- Running mode of ETF application
  do
-- Result := etf_gui_show_history -- GUI mode
   Result := etf_cl_show_history
-- Result := unit_test -- Unit Testing mode
  end
```

2. A first BON diagram that shows your design (classes inside the **model** cluster), and how it is related to the command classes (e.g., `ETF_MOVES`, `ETF_MOVE_AND_CAPTURE`, `ETF_SETUP_CHESS`, `ETF_START_GAME`, `ETF_RESET_GAME`, `ETF_UNDO`, `ETF_REDO`, *etc.*) in the `user_commands` cluster (including the singleton pattern).

   Here you do not need show details of your model classes (i.e., oval shapes with their names, indicating if they are deferred or effective, suffice).

3. A second BON diagram that shows details of your model classes only (i.e., contracts and any inheritance or client-supplier relation between them).

4. You must include the draw.io XML source files of your two BON diagrams and their two exported PDF files in the `docs` directory. **If the TA cannot find, in the `docs` directory, the two draw.io XML sources and PDF files of your BON diagrams, you will immediately lose 50% of your marks for that part of the lab.**

10.2 Submitting Your Work

- There are two `chess_solitaire_undo` directories: one is the top-level directory that contains all generated ETF code and your development; the other is the sub-directory that contains the `model` cluster. **When you submit, make sure you submit the top-level `chess_solitaire_undo` directory.**
- Go to the directory containing the top-level `chess_solitaire_undo` project directory:
- Run the following command to remove the `EIFGENs` directory:

```
eclean chess_solitaire_undo
```

- Run the following command to make your submission:

```
submit 3311 Lab4 chess_solitaire_undo
```

  A check program will be run on your submission to make sure that you pass the basic checks (e.g., the code compiles, passes the given tests, *etc*). After the check is completed, feedback will be printed on the terminal, or you can type the following command to see your feedback:

```
feedback 3311 Lab4
```

  In case the check feedback tells you that your submitted project has errors, you must fix them and re-submit. Therefore, you may submit for as many times as you want before the submission deadline, to at least make sure that you pass all basic checks.

  Note. You will receive zero for submitting a project that cannot be compiled.

11 Questions

There might be unclarity, typos, or even errors in this document. It is **your responsibility** to bring them up, early enough, for discussion. Questions related to the lab requirements are expected to be posted on the on-line course forum. It is also **your responsibility** to frequently check the forum for any clarifications/changes on the lab requirements.