|epcc|

Software Development
Coursework 3: Improvement and Reflection

Exam no. B076963

Contents

List of Figures

List of Tables

1 Introduction

Prior to the commencement of this project, we have created a proposal for a plan of the process of restructuring a poorly designed code for a simple card game, including potential risks and estimation of time and effort required for this code alongside other commitments. Outlined herein are thoughts on the development process of restructuring said code and how the actual process corresponded with the initial plan. We also present changes made to this code and how the implementation of these changes affected the development process, as well as potential future enhancements to the code that are at present outside the scope of the development and what risks would be carried by implementing these enhancements.

2 Repository access and build instructions

The Git repository for the project is public but it is hosted under a privately hosted instance of GitLab, which is available at `http://git.138.io/atar/public.git`
Run the following command to clone the repository.

```
git clone git@git.138.io:atar/public.git
```

The root directory of this project is accessible via the following path.

```
cd public/hpc/sd-cardgame
```

There are two main subdirectories here; one for the code and another for the documentation of the code, including a list of changes, some ideas of future implementations and some thoughts on the process.

In order to test the file, run the following script in the `code` directory

```
./play_dbc.sh
```

3 Improvement and Reflection

3.1 Reflection on Process

As stated in the development plan, the highest risk was predicted to be the refactoring of the code, which was confirmed in the process of development. Quite a significant amount of effort was put into trying to understand how to implement the suggested ideas and create the most suitable object-oriented design. This extended development time from what was hoped to be a rather small amount of time to almost half of the required time for the entire project.

One such issue that hindered the development process was the treatment of the function that would load in the JSON data. In the end, the best idea was to create a class called Data that contains the JSON loader function. While there may be a significantly more streamlined solution, for the time being it is sufficient for our purposes.

Another major issue that persisted in the coding process was creating coherent tests for the code. While in a lot of cases it seemed sufficient to put the asserts equal to some expected value in the output, this is not general enough for a non-specific use case for the function that the test was written for. This does not affect the way the program itself is executed.

3.2 Summary of Changes

For a concise list of changes, see `docs/changelog.md` The major changes are outlined below.

Change	Description
OO Deisgn of code	OO is ideal for a game project like this, since that makes it easier to identify what code would run each component of the game, such as a card being an object, a pile of cards being an object, the players, the game board, the game itself, and so on. The code base has been changed from a single file to a package that contains the game code.
Input and text	Card input and text strings are now handled as input from a file. This reduces the amount of hard-coded elements in the code and makes it easier to make changes or additions to game assets, i.e., all you have to do is add a line to the JSON document
Text output	The text output is made clearer by clearing the game screen for each move. While not a perfect solution, it does significantly clean up the interface and make obvious what moves are performed by each player
Surrender option	Before there was no option for the player to leave the game during the game other than killing the process. Now it is easy for the player to leave the game before it finishes if they wish to.
Shell script executable	`play_dbc.sh` is now included to simply execute the main script in the game directory. This is a simple solution to having an executable to run the code. The scrips also cleans the directory after the game finishes from Python bytecode that is created upon execution of the package.

Table 1: Table of changes and their impact on the code.

3.3 Future Enhancement

For a concise list of changes, see `docs/future.md`

Change	Description	Risk
GUI	We have started on a mockup of a graphical user interface in Qt4. For a project like this it would not be difficult to extend the already existing code and write some more methods to implement a graphical user interface. Some risks are involved in this, of course, where some functions would need to be rewritten so that they are usable with Qt4	High
Multiplayer/PvP	It is possible to have an option at the beginning of the game to choose between playing with a bot and playing with a human opponent. What that would entail is changing the declaration of the player in the code and prompting the players to pass the client running the program to each other between each turn. The risk associated with this would require restructuring of the bot functions and the Bot class to be encapsulated, in order to make the code easier to use for different game cases.	Medium
Bot	Following on from the previous suggestion, the bot code was, admittedly, written in a hurry. While the code works as is, it would be a better idea to have the bot operate standalone, and while a skeleton code exists for a Bot class, it is not populated.	High

Table 2: Table of possible future enhancements.

4 Conclusion