CS5031 p1 report

Student id:230009784

1 OVERVIEW OF HANGMAN IMPLEMENTATION

1.1Hangman implementation

The main logic for implementing this game is to set two word lists. One is “correctLetters”, the other is “wrongLetters”. Every time the user inputs a letter, the program judges if the letter is right. The right letter will be put in the Set correctLetters, the wrong letter will be put in the Set wrongLetters. The program will show the current word state according to if the Set correctLetters contains the letter of the target word, draw the Hangman according to the guessesRemaining.

1.2How to play

In the beginning, the current word state is five underscores “_ _ _ _ _”, and the program will hint the user to enter your guess, so the user needs to input a single letter.

When the user guesses correctly, all the letters of the target word will appear to replace underscores. When the user guesses wrong, the image of Hangman will draw one part of Hangman in the order of head, torso, left arm, right arm, left leg, and right leg.

When the user guesses all the letters of the target word within the max numbers that the user can guess wrong, the user wins. When the user uses all the chances can guess wrong, the user loses, and the game is over.

The program will give the user a score if the user wins according to the remaining number the user can guess wrong.

2 TEST-DRIVEN APPROACH

Firstly, in this program, I divide four main requirements according to the game Hangman. They're used to generate random words, handle the user input, control the game state, and display the game state respectively.

Secondly, I design tests and test uses according to those requirements. More specifically, I designed InputHandlerTest to test if the user input is invalid and return the letter user inputs. I design GameStateTest and write the test cases to test if the letter the user guesses is correct, to get the remaining numbers of user can guess, to get a max number that the user can guess wrong, to get wrong letter lists, to get current word state, and to judge if the user wins. I designed DisplayTest to test the display of the current word state, guessed letters list, Hangman image, win message, and lose message.

Thirdly, I design corresponding classes and methods to try to meet those requirements and test cases using the minimum amount of code.

The above steps helped me figure out the project requirements at the very beginning and meet the requirements in a very short time.[1]

3 REFACTORING APPROACH THROUGHOUT DEVELOPMENT

3.1Modify function name, variable name

Improved the name of every method and variable in the project.

3.2Refactored the method of some classes

In the beginning, I try to implement the method as soon as possible in order to pass the tests. However, during this process, some methods may lack standardization. So after I passed the tests, I needed to refine the method and make the method most efficient.

For the class RandomWords, I refactor the method getWord() making it more simple and readable. This operation didn't affect the outcome of the method.[2]

For the class Hangman, I added methods startGame(), procrssGuess(), and concludeGame(). They can stand for each stage and control each procedure in the whole game. This operation also makes each method do one thing and only does one thing. Then the program can call those methods in method startGame(). Refactoring the method can make the logic more clearly.[3]

4 OTHER SOFTWARE DEVELOPMENT PRACTICES USED

4.1 Git

I used Git to version-control the code.

I commit 6 times in total. I can use `git log` command to view the commit history for each commit and view modification in detail by 'git log -p` command. In the meantime, I push the local repository to my Github repository by SSH, which can help me check the files more conveniently.

4.2 Comments

I wrote comments for each class and method. I didn't write comments for the specific code, because the code is clear enough to understand.

5 OTHER RELEVANT DISCUSSIONS

For the implementation of this game, I used 5 classes to control. And divide the method the methods as small as possible. I try to make this program understandable and readable.

REFERENCES

[1] Anwer, F., Aftab, S., Waheed, U. and Muhammad, S.S., 2017. Agile software development models tdd, fdd, dsdm, and crystal methods: A survey. International journal of multidisciplinary sciences and engineering, 8(2), pp.1-10.

[2] Martin, R.C., 2009. Clean code: a handbook of agile software craftsmanship. Pearson Education.

[3] Fowler, M. and Beck, K., 1997, June. Refactoring: Improving the design of existing code. In 11th European Conference. Jyväskylä, Finland.