Assignment 1: Kalah program report

Zhiqian Zhao
Student
University of Auckland
Auckland, New Zealand
zzha198@aucklanduni.ac.nz

***Abstract*—This electronic document is about the maintainable software design report in the Kalah game program.**

Keywords—Kalah, maintainable, comprehensibility, alterability, testability, java.

I. INTRODUCTION

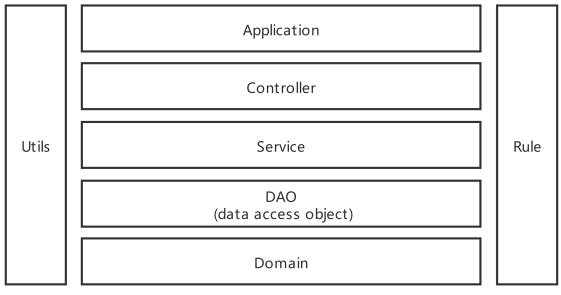
Kalah is a chess game played by two people. It is called "Mancala" in some places.1 This program realizes the Halah game function and carries out the battle between two players through the player's keyboard operation. In this report, we focus on maintainable software design including comprehensibility, alterability, and testability.

II. MAINTAINABLE SOLUTION

A. Comprehensibility

Comprehensibility is an important part to help developers understand Kalah's game rules and the program mentality of designing. An easy comprehension program will help the coder easily and efficiently do maintenance projects.

In the Kahal program, the significant consideration is a hierarchical structure. This would be convenient for developers to understand and analyze the program code2. We use the idea of layering to split the complex system into domains, to modularize the system, reduce the learning cost of developers, and pay more attention to the specific functions of segmentation. Like Figure 1, the Kahal program split five-layer which are the domain layer, the DAO (data access object) layer, the service layer, the controller layer, and the application layer.



(Figure 1: a hierarchical structure)

In the domain layer, there are two objects which are player information (PlayerInfo) and board information (BoardInfo). The PlayerInfo object is used to record the information of single-player include player name, houses information, and store. The BoardInfo is to show the game board. So, it contains each player's information and hidden game control information.

The DAO field provides data operation ability. This field only has a BoardInfo object operation function. The BoardInfoDao object can update or get the specific BoardInfo property, such as getting/setting the specific player' house, get/set the specific player' store, setting the specific player' seeds in the appointed house.

The service layer is the logical layer that mainly processed business logic. The service layer is the logical layer that mainly processed business logic. The named GameSerice object will get game rules, board information, and interface presentation function together. Consequently, this object would not be named as BoardInfoService. In the GameSerice, there are four important moving seeds methods, which apply to different situations, such as the method of movingOwnHouseByAppointedHousesNumber (moving own house seeds By appointed houses number), the method of movingOwnStore (moving seeds to own store), the method of movingOpponentHouse (moving seeds to the opponent house), and the method of movingOwnHouseByFirstHouse (moving seeds to own house from the first house).

The major functions of the controller layer are to controller game step, include game initialization, game key action, and end of the game judgment. The controller layer mainly integrating the functions of the scheduling service layer and the game rule to realize the rule-compliant game control ability. In terms of action methods, we know that in this game, players can only enter the houses number and “q” to exit. To improve comprehensibility, the business logic that should be hit in each operation is distinguished by controlling the input of keys. Therefore, the specified case judgment is carried out through the switch method, and the input that does not meet the rules will exit directly.

The main method, the application layer, is just to easily control the game output. To facilitate understanding the game logically, after the game is initialized, only the 'Q' command is detected before exiting, otherwise, the entered houses number will be monitored all the time. Before each action, simply add the judgment of whether the game is over or not.

B. Alterability

The variability of software mainly considers that the rules or functions of the object can be changed effectively and efficiently without introducing defects or reducing the quality of existing products. An important idea in java programming is the inversion of control (IOC)3.

In the Kalah game program, the DAO layer and the service layer adopt the programming idea of IOC. The boardInfoDao interface is designed to define how to operate the functions of the BoardInfo object. Similarly, there is the same design in the service layer. Similarly, there is the same design in the service layer. In the GameService interface, the main functional interfaces of the game are designed, including the init method, the isHouseEmpty method, and the move method.

Furthermore, set various parameters during game initialization through the global config configuration file. For example, the number of seeds initialized. In this way, when the initialization rules of the game change, we can easily change the initialization data rules without modifying the code layer.

C. Testability

Software testability mainly focuses on how to perform testing effectively and efficiently to ensure that the program function meets the standard. Through interface-oriented programming, the functions of each interface can be easily tested in the test environment. The functions are granulated in a hierarchical manner so that each function can be tested arbitrarily.

III. Conclusion

To sum up, the Kalah game is a good case to explain the design idea of maintainable software. The understanding of comprehensibility, variability, and testability will keep a software product useful and easy maintenance activities.

References

[1] Wiki Group (n.d.), (2016, Feb), "Halah", https://en.wikipedia.org/wiki/Kalah

[2] G.Ziemoński, (2019, Feb 13), "Layered Architecture Is Good", https://dzone.com/articles/layered-architecture-is-good

[3] Crusoveanu, L, (2021, July 22), "Intro to Inversion of Control and Dependency Injection with Spring ", https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring