SOFTENG701 Assignment 3

Kalah implementation

Edwin Roesli
eroe303 155610518
Electrical & Computer Engineering
University of Auckland
New Zealand

I. Project Structure

Kalah, also known as Mancala, is a board type game which is traditionally played with 2 players, a board embedded with 6 house pits and a store pit for each player. The aim of the game is to be the player with the highest score at the end of the game and the game ends when a player contains zero seeds in the players' houses.

The aim of this assignment is to design an object-oriented implementation of the game Kalah while making the implementation as changeable as possible.

II. Object Oriented Design

A. *Encapsulation*

Encapsulation refers to the overall hiding of data, implementation and overall inner workings of classes. This provides security and prevents data corruption through outside classes. This can be shown in my implementation through the abundant use of private fields in classes, with the exception of the Pit class with a protected field. However, this still protects that field to all other classes not inheriting the Pit class. This can also be shown through the GameManager class, the implementation of the game does not use or alter any other classes' information, it only uses the information.

B. *Abstraction*

There are many cases of this in my implementation, the important cases in which abstraction is being used is shown as follows. The game manager class is where the main game logic is being handled (at a high level). The game manager class stores an object reference to the Board class and constantly calls the boards class' methods, for example, pickTurn, checkMovePossible and moveSeedToStore. These methods all either alter the game board object instance or analyses the board to return the state that the board is in. This hides any implementation details from the GameManager class and in turn, makes identifying logic in the program easy to locate.

C. *Inheritence*

The "seeds" are stored in a Pit, there are two types of Pit: House and Store. The two types of pits are essentially the same, the main difference being that during a player's turn the player cannot set the stores seeds count to zero and pass the seeds along with the board. Thus, we have a Pit class which contains the commonalities between the store and house class. Both the House and Store class extend the Pit class but only the House class has the additional method: "resetSeedCount" which sets the seed count to 0. This method is used in conjunction with iterating the number of seeds that the chosen house has throughout the board.

D. *Single responsibility principle*

The classes in my implementation of Kalah all coincide with the single responsibility principle. The main classes such as the Board class or the GameManager class all serve one job which helps with decreases coupling between classes. All the printing of the game board/ text or score is all done in a separate class rather than in the GameManager class or Board class.

III. Changeability

Changeability refers to the number of classes that need changes to implement or amend new functionality without considering understanding the code base.

In the case of my implementation of Kalah we observe that in the main class, Kalah.java, contains the bare essentials of what the game instance needs. Creating a new game with a set number of players, initial seed count in each house and the number of houses a player should have, starting the game and ending the game. If the game, at a high level, ever needs to be changed, for example, multiple instances of a game to be running at once only one class needs to be edited. This also conforms to object-oriented programming conventions as we observe low coupling and the ease of instantiating a new GameManager object for each game instance that is needed.

The separation of logic is used as much as possible to ensure that if any changes are needed to conform to new design features then only a low percentage of the total number of classes need to be changed. This can be seen through the separation of the core game logic and the inner game logic. If any changes to the core game logic should change, the developers should only have to look at the GameManager class.

The Board class only contains logic which directly edits the board, be it through the player choosing a house to spread its seeds or different instances such as capturing. The board class also deals returning the current state the board is in at that instance. So, if any logic that deals with the movement of the seeds or the decision to decide what state the game is in should change then the developer would only be required to edit the functions of this class. The board also contains ArrayLists of Houses and Stores which is mapped to different players, since these Houses and Stores are superclasses of Pits we can easily edit the way in which the game initialises the board by simply editing the initBoard() method that the Board class contains.

The module which deals with printing the board or text onto the terminal is all handled by the IOPrinterManager class. This allows any modifications to the output seen in the terminal to be edited in this one class rather than edit the modifications in the Board class or the GameManager class.