Implementing Kalah with a Changeable Object-Oriented Design

Matthew Eden
Department of Electrical, Computer and Software Engineering
University of Auckland
Auckland, New Zealand
mede607@aucklanduni.ac.nz

***Abstract*—There were several decisions made in designing Kalah (a.k.a Mancala) in Java 7 such that it supported changeability. Three of those decisions are discussed here: the creation of a configuration class, the abstraction of game elements and the separation of IO from game logic.**

***Index Terms*—kalah, mancala, object-oriented design, changeability, java**

I. Design Decisions

Associate Professor Ewan Tempero defined changeability as *How much it costs to make the necessary changes to existing code, once those changes have been identified* [1]. The design decisions outlined here were made with this specific definition in mind.

A. Use of a Config class

The design I have implemented groups commonly used values, such as `NUM_HOUSES` and `STARTING_SEEDS`, inside a globally accessible configuration class `GameConfig`. These values can be modified within the class to then alter the behaviour of the game, such as increasing the number of houses for each player. This decision supports changeability because the cost of changing these values is very low (given they are merely constants) and although they are referenced in many places, they are only defined once. As such, if an individual wishes to change the value of some parameter, like the number of houses per player, they need only change this value at its definition in the configuration class instead of at every point in the code where that value is used.

B. Abstraction of Game Elements

My design abstracts different game elements into dedicated classes, such as `Player`, `Board` and `Pit` (a combination of house and store). Each of these classes has been designed in such a way to reduce coupling and increase cohesion. For example, there is a single method in `Board`, `makeMove`, which is concerned with the logic of a player's move. This decision supports changeability as an individual looking to change these aspects of the game can do so by only changing the implementations of these classes or methods contained within these classes. As such, the cost of a change to these aspects of the game is minimised.

C. Separation of IO and Game Logic

My design makes use of a class `GameIO`, which is concerned with handling player input and console output. It is a dedicated class; that is to say outside of it there are no other invocations of IO methods. This decision supports changeability as it means future alterations to IO are constrained to a single class. It also means that an individual does not need to be concerned with breaking the functionality of the game by altering IO, as they have been separated in this way, thus lowering the cost of the change.

References

[1] E. Tempero "SOFTENG 701 - Lecture 01b - Changeability", Slide 12 of 17, SOFTENG 701 2020, canvas.auckland.ac.nz, retrieved on May 13th 2020.