Implementing the *Change Board Orientation* change case

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

***Abstract*—To evaluate the changeability of an implementation of Kalah (a.k.a Mancala), a change case was proposed. This involved changing the *horizontal* orientation of the board to a *vertical* orientation. A change plan was created in anticipation of implementing the changes in code. During the execution of this plan, there were some issues encountered, but not a significant amount. The time taken to implement the change case was roughly 16 minutes. The overall impact of the change case was low and it was straightforward to preserve the functionality for the horizontal orientation if this change need be reverted in future. As such, there is a good amount of changeability in this design.**

***Index Terms*—kalah, mancala, changeability, java**

I. Planning the Change Case

Before any code was actually written, a plan was devised describing the minimal amount of changes that would be required to achieve the desired change case. This plan is as follows.

1) Find the `GameIO` class
2) Locate the `printBoard` method
3) Add an additional argument: `boolean isVertical`
4) With the exception of the first non-blank line concerning `List<Players>`, wrap the method body in an `if-else` statement.
5) Make this `if` condition controlled by the new method argument, such that `if (isVertical) {...} else {/* ORIGINAL METHOD BODY */}`
6) Within the `if` branch, add 4 `io.println` statements for printing out the outline of the board and the player stores, ensuring that they follow the specified format.
7) In the middle of those statements (i.e. between the printing of player stores), add a call to a new method `printHousesVertically`, passing in `GameConfig.NUM_HOUSES` as the argument.
8) Implement this method to print out the houses of both players such that it follows the specifed format.
9) Update invocations of `printBoard` to pass in `true` as a third argument.

II. Implementing the Change Case

The total amount of time taken to implement the change case was timed as being 15 minutes and 58 seconds. During the execution of the aforementioned plan, there were some issues encountered. Overall, the plan was a success and it was not difficult to implement the change case. A breakdown of the steps carried out follows.

1) No issues in finding the `GameIO` class.
2) No issues in locating the `printBoard` method.
3) No issues in adding an additional argument to `printBoard`.
4) No issues in creating an `if-else` statement to wrap the method body.
5) No issues in implementing the condition for the `if-else` statement.
6) No issues in adding the required `io.println` statements. Although the argument to be provided to those statements was not specified (i.e. what exactly should be printed), it was not difficult to figure it out based on the format specified by the test cases and the pre-existing implementation of the `printBoard` method.
7) No issues in adding a call to `printHousesVertically` and providing `GameConfig.NUM_HOUSES` as the argument.
8) There was some issue in implementing `printHousesVertically`. The number of method arguments specified in the plan was not sufficient, as the final version of the method created required *three* arguments, as opposed to the *one* that was originally planned. The two additional arguments were `IO io` and `List<Pit> pits`. This then meant that the invocation of `printHousesVertically` in `printBoard` required changing and required additional values to be passed in. There was already a pre-existing `IO io` argument in `printBoard`, so there was no issue in passing that as an argument to `printHousesVertically`. However, there was no pre-existing `List<Pit> pits` or equivalent argument. Within the `printBoard` method there was a pre-existing `Board board` argument, with that object containing a `List<Pit> pits` field that would be sufficient. However, there did not exist a getter method to access that field, so this had to be implemented. The plan did also not go into detail about *how* to implement the `printHousesVertically` method, however the pre-existing method `formatHousesAsString` was sufficient as a guide for figuring out the implementation.

9) No issues in updating existing invocations of `printBoard` to pass in `true` as a third argument.

III. Impact of the Change Case

On a scale from *0.0* to *1.0*, the impact of this change case is *0.1*. For all intents and purposes, the game of *Kalah* can be played in the same manner as before the change case despite the visual change. In saying that, these visual changes mean the impact is strictly non-zero.

IV. Changeability Assessment

My original assessment of the changeability of this design claimed that the design supported changeability in part because of its separation of IO and game logic [1]. The successful implemention of the *Change Board Orientation* change case further supports those claims. Although there were some issues faced when following the change plan, these issues could be argued to arise from poor planning rather than a lack of changeability in the design. The only issue directly related to *changeability* that arose was the lack of a `getPits` method in the `Board` class. In spite of this, the implementation of the change case did not take a significant amount of time and resulted in minimal impact on the functionality of the implementation. As such, the original changeability assessment of the design was correct.

References

[1] M. Eden "SOFTENG 701 - Assignment 3 - Submission", *Implementing Kalah with a Changeable Object-Oriented Design*