SE701: Assignment 5 | Robot Player

Shane du Plessis
The University of Auckland
sdup751@aucklanduni.ac.nz
25 May 2020

1. IMPACT

The change case will have an impact of 0.05. The change case would only affect how the game takes input from a given 'player'. Otherwise, the games functionality is still the same, therefor the impact falls in the lower end of the scale.

2. PLANNED CHANGES

As the change case behaviour is drastically different to the existing player class, it is sensible to create a new class. The new Robot class will implement the player interface and make use of the abstract method declared. This design ensures minimal existing code is modified and minimal new code is added.

Within Kalah Class

1. Modify 'player' ArrayList to be of type PlayerInterface and not Player.
2. Create new variable field addRobot = 1 (to store number of robots to add)
3. Modify how players are instantiated, write loop to read numberOfPlayers and addRobot variables (i.e. can add 0, 1 or 2 robots to make up numberOfPlayers)
4. Refactor method name promptHumanPlayer to promtPlayer.

Within KalahBoard Class

5. Create public get method to return storage element objects (to be called from Robot class), declare method in Board interface.
6. Redeclare getArrayIndex method to be public (to be called from Robot class), declare method in Board interface.

Create Robot Class

7. Create new class Robot to implement PlayerInterface. Create private fields io, board, playerID, sizeOfBoard, hollowPicked. (Robot constructor to take appropriate variables to instantiate these fields.)
8. Create private enum MoveResult to represent possible move decisions (AdditionalMove, CaptureMove, LegalMove).
9. Implement abstract method promtPlayer(). This method prints decision made using a switch case statement on MoveResult enum (set in selectHollow helper). PromptPlayer returns the hollow number picked in the form of a string as expected in Kalah class.
10. Create private function 'selectHollow()'. Function sets local variable 'move' to appropriate enum when corresponding case is found. The function then returns the int 'i' (hollow index) as a string value for PromptPlayer to return as expected by the Kalah class input.

3. EXCECUTION || START TIME 22:00

Changes to Existing Code – 4 mins: Changes to existing code involved carrying out steps 1, 3, 4 and 6. The steps provided enough detail to carry out the plan and prepare the program to accept the new Robot module. This support my initial thought of the change case being low cost and low impact.

New Code Additions – 30 mins: Majority of the implementation time was taken up by implementing the Robot class functionality. The steps outlined provided clear enough instructions to carry out the change case successfully. I did not have to deviate from the plan outlined. Using the PlayerInterface to implement this class meant very little existing code changes had to be made. The only addition to the plan was creating helper function to the selectHollow method in the Robot class. This reduced duplicate code and reduced the methods code smell. Tests were carried out (not included in timing) and all passed on the first attempt as the plan was detailed enough to capture all the necessary functionality.

Completion time – 22:34 (total time – 34 minutes)

4. DESIGN CHANGEABILITY

The starting codebase used for this assignment was my original assignment 3 submission. This is because the design showed promise and was not discredited by implementing the assignment 4 change case. The assignment 5 implementation has been set up to be easily changed to pass the assignment 3 tests. All that is needed is to set addRobot = 0 in the Kalah class.

This change case was anticipated in my assignment 3 architecture. The change case could make use of a PlayerInterface class to implement the PromptPlayer abstract method. Changes to existing code took up only a small fraction of the implementation time. This approach meant less changes were needed to be made to the existing code. This is a trait of a changeable design.

A construct to gauge the changeability of the design is to assess '# classes changed' over '# total classes'. In total only 2 of the now 8 classes had to be modified, yielding a low ratio. The new functionality can be plugged in as a separate module. This points to the design exhibiting changeable qualities. Overall adding the changes to the existing code was inexpensive and low impact. Implementing this change case highlighted the changeable traits within the design.