

Modularity Benefits of Strategy Pattern in Kalah

SOFTENG 701

Advance Software Engineering

Development methods

Wong Chong (wcho400)

wcho400@aucklanduni.ac.nz

I. STRATEGY IMPLEMENTATION

The *Strategy* pattern was implemented by observing the possible difference which could occurred between the two players. In the game *Kalah*, the only difference between both players is how they calculate the movements of seeds. As such, I have implemented an interface for movement called *MovementStrategy*, which is inherited by two classes, *Player1Strategy*, and *Player2Strategy*.

MovementStrategy contains two methods. The two methods are to check the legibility of the move and to start moving the pieces. For the methods to work, each strategy is assigned a *Team* element, which is used to set up both methods. This element is to provide the strategy the original stall associated with the team. Overall, implementing the strategy pattern was done by refactoring the code used to determine which Team was being used.

The first part of the difference is how to calculate if the move was allowed. As ensuring a valid number is the same for both teams, that remained unchanged. However, determining if the store was a possible selection is different in both classes. I have now defined a method called *checkLegibility* in the strategy which returns if the move is allowed for the strategy. The other part of the difference is how to begin move the seeds around the board. Previously, I had defined a method *doAtMid* to move the seeds around. However, this method exists dependent on the *Team* object, which meant the movement of each team would be the same. Now, I have defined a method called *move* in the strategy which allows the starting points to be clearly separated.

Other changes made for this strategy is the renaming from *Board* to *Logic*, as this better encapsulated what the code did. I then refactored out the team collection which contained each teams' stalls and houses into a new class called *Board*.

II. STRATEGY AND MODULARITY

In my experience of implementation *Kalah* with this pattern, implementing the *Strategy* pattern does improve modularity, but with a non-positive impact. The modularity has improved as this allows different strategies to be used without creating a dependency to each different strategy, provided the interface is used. However, in this implementation of *Kalah*, the pattern gives no benefit as the code in each strategy is the same. As the only difference between the game is who is playing the game, the possible strategy difference would be how to calculate the results of their actions. However, the structure of the implementation was designed such that each team is aware of its own stalls and can communicate with the next team. As such, the code to calculate the movement would be the same. With such little benefit, this implementation, this has increased the class count and code duplication, which can cause greater issues in future development as well as understandability of the code.

From this experience, I do believe implementing the *Strategy* pattern in general can improve the modularity of a design. By implementing this pattern, this can reduce the amount of dependencies by only associating with the interface whenever a different strategy is being used. Yet from this experience in implementing this pattern in *Kalah*, it is clearly not suitable for all occasions. As I have mentioned before, this pattern in *Kalah* simply increased code duplication and class count for some possible improvement in modularity. This pattern should only be used in designs which have genuine variance, rather than a perceived difference in actions.

III. DISCUSSION

Integrating the *Strategy* pattern into the design of a project should be done with caution. As designing software is always a balancing act of different decisions, I would be wary of integrating a strategy pattern if the only purpose of it is to increase the modularity of the design.

I would also like to point out that my opinion on the effects of modularity on this pattern is also influenced by my previous design decisions. I believe that due to my previous design decision to clearly separate and group each team with its associated stalls and house, the strategy for both instances are the same. Without properly grouping of these elements, I believe the strategy would be able to provide more value to the design than simply increasing the modularity, yet come at a cost of poorer encapsulation