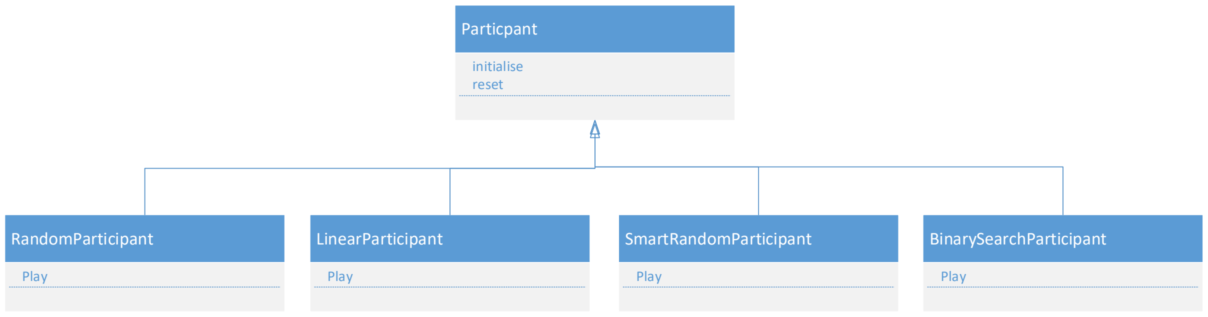
Template Method Pattern and Strategy Pattern – Day 1

Initially I found this quite difficult. This was probably a lot to do with the fact that I was rusty with code and there was a reasonable bit of detail here from my perspective. The code initially looked quite messy and I needed to step through it a bit to get the "gist" of what was going on.

Reasonably quickly I could see the repeated code in the Participant.rb file and the main.rb file also. There were slight differences in each implementation but lots with lots of common code also. From a conceptual level I could see that each "mini-game" looked like a candidate for the strategy pattern.

As the exercise called for attention in the participant class I focused my initial attention there and discussed with my partner. We quickly identified the approach to apply the strategy pattern to the participant class in the following manner



After completing this I struggled a little trying to think about how to use the Template Method Pattern to reduce the amount of repeated code.

This was easier for me to do in the main.rb class. I created a method run_game and passed the strategy as a parameter. The method would then instantiate the appropriate participant class and run that objects play method to run the exercise. This action really simplified the main.rb file making the code a lot more manageable.

The work in cleaning up this main class was very similar to the work in the Participant class in that the strategy was passed as a parameter and that common code was compressed into one method that all strategies would call. This would support the requirement of changing strategy at run time.

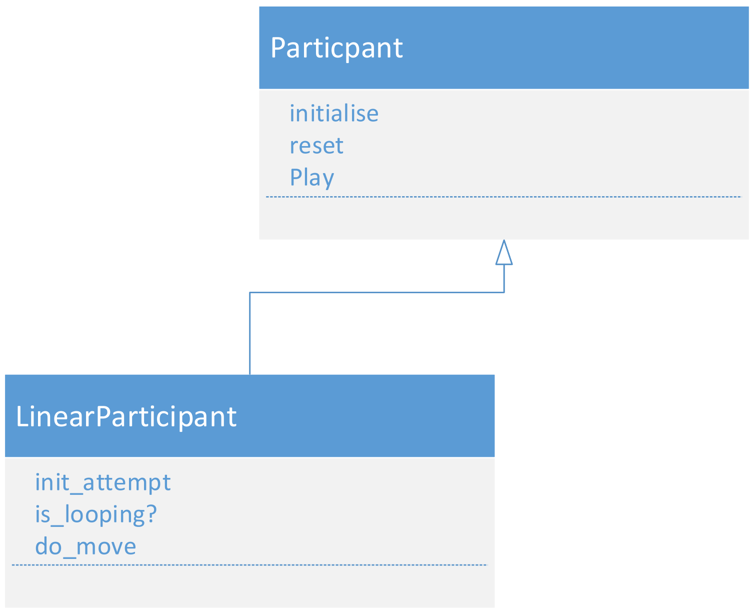
The last piece of work was to refactor the individual strategy implementations to use the Template Method patterns. I struggled a lot with this initially, as the implementations were spread over multiple files making it hard to me to spot stability and instability between them. I identified that the base structure was common

1. Initialise the starting state
2. Determine when to loop and when to break
3. Make a move
4. Return a success or failure.

All strategies performed these 4 steps but the specifics of 1..3 were different.

I refactored the play method to be the template, and each subclass would evaluate parts 1..3 specific to the type of game they were modelling.

This new structure looked as follows



All three methods are abstract methods but I chose not to throw an exception in the superclass following guidance in the morning.

I added a descending game also to extend the problem to more game types. This proved pretty easy to do compared to the other work in the exercise…

Overall in this exercise I think I was able to spot the need for each of the patterns from a quick look at the 2 files (main.rb and participant.rb) but found myself making them more complex by focusing on how to make the two work together in an appropriate manner.

The end result was code that certainly looked a lot better, would read much cleaner, and was easier to debug through but one consequence was that code was split over more locations, that caused the diffing to individual algorithms more complex.

I naturally identified the Strategy pattern first and this was the pattern that I kept seeing as a good help to the problem and as such I implemented that first. With hindsight I do wonder if leading with the Template Method Pattern would have gotten me to where I ended up in a faster manner.