Peer Review: Fia med knuff

av Koworkers

Koden är lätt att förstå och sätta sig in i, eftersom namn på metoder och klasser är bra, man får förståelse på vad som görs i metoderna och det är tydligt vad klasserna representerar. Koden är även väldokumenterad, några metoder saknar fortfarande dokumentering men det ser ut som att det bara inte hunnits med än. En bra sak är att även privata variabler har kommentarer som förklarar vad de representerar, vilket även det gör koden lättare att förstå. En annan sak som är väldigt bra är att koden är vältestad.

Det används inte några interfaces eller abstrakta klasser, men det finns inte riktigt ett behov för det i nuläget. På så sätt används rätt abstraktioner, då det inte är bra att till exempel ha ett interface bara för att ha det.

Designmönster

Strukturen på koden följer Model View Viewmodel, MVVM, det vill säga att vyn ska bero av viewmodel, och viewmodel ska bero av modellen, men inte i omvänd ordning. Dock implementeras den inte riktigt rätt då det ligger mycket logik i vyklasserna som egentligen borde ligga i modellen. Till exempel borde makeTurn i standardboardFragment ligga i modellen. Vyn borde egentligen bara kalla på metoder i viewmodellen hanterar omvandlingen från modellen till vyn. Sen borde viewmodellen kalla på metoder i modellen som har hand om logiken i programmet. Modellen borde sedan notifiera viewmodellen att den uppdaterats och sedan notifierar viewmodellen vyn via till exempel observer-pattern eller en eventbus. Mycket av logiken som ligger i viewmodellen skulle också kunna ligga i modellen, allt med selected piece, currentplayer, playerNames, diceValue, movablePieces etc. skulle kunna läggas i modellen istället. Vissa av dessa verkar inte ens användas i nuläget.

I koden finns en klass GameFactory som för tankarna till factory-pattern, men som inte använder ett interface. För att GameFactory skulle fylla någon funktion borde den använda sig av ett interface som Game implementerar. Ett Game-interface skulle kunna göra att man senare kan utvidga koden med olika sorters spel, men vi ser inget behov av detta och tycker att det vore rimligt att ta bort GameFactory och istället lägga logiken i konstruktorn till Game.

Designprinciper

Principen Open-Closed Principle innebär att moduler ska vara öppna för utvidgning, men stängda för ändring. Koden har många privata variabler, vilket innebär att andra klasser inte kan ändra på dem, vilket gör de stängda för ändring. Det finns flera variabler som är final, vilket är ännu bättre ur ett OCP-perspektiv, men det finns fler variabler som skulle kunna vara final, så det finns utrymme för förbättring. För att följa OCP behöver koden också vara öppen

för utvidgning vilket den inte alltid är i nuläget. Det märks främst i StandardBoardFragment där det finns flera hårdkodade listor för t.ex positioner. Vill man göra brädet större blir därför detta svårt, då fler variabler behöver hårdkodas, istället för att t.ex bara ändra en längd på en lista.

Det finns även flera getters för listor m.m som gör att man kan komma åt och ändra i listorna. Detta skulle man kunna fixa genom att, till exempel, tillämpa defensive copying. Det finns även flera metoder som inte har några access modifiers, till exempel getMovablePieces, isMovable och removePiece i Player-klassen.

Att det ligger så mycket logik i vyn och viewmodellen bryter även mot single responsibility principle vilket gör koden svårare att förstå och som det är nu så fungerar inte modellen utan vyn/viewmodellen vilket gör det svårt att återanvända modellen/koden i andra sammanhang.

Förbättringar

Funktionen connectBoardsPositionsIds skulle kunna göras om till en lista och loopas igenom istället, liknande kan också göras i connectHomePositionsIds. Alla imageViews för de möjliga positionerna borde kunna läggas i en lista vilket skulle göra det lättare att kunna återanvända koden för olika antal spelare.

Det skulle förbättra koden att ha en viewmodel för varje vy och sedan flytta logiken som för närvarande ligger i vyerna och varken hör hemma i view eller modellen där. Vyerna borde egentligen bara säga när något blivit klickat på och sedan låta viewmodellen/modellen hantera detta. Att ha en viewmodel för varje vy är också bra för att göra koden mer återanvändbar, då viewmodellen korrekt implementerad viewmodel ger uppdelad kod och det är lätt att bara ta den delen av vyn och viewmodellen man vill använda igen.

För att förbättra användarvänligheten skulle det vara bra med något som förmedlar var man kan klicka och vems tur det är.