Peer review done by: Kevin Jeryd, Vincent Hellner, Dadi Andrason, Elias Falk
Peer review of code pulled: 14/10-2021 14:15
Commit revision number: 02752bc35a39a186f802f1fc4be12578b551125b

Designprinciper

Kodstil

Projektet använder en ganska konsekvent coding style men den kan förbättras på ett antal ställen genom en generell kod reformattering.

Kodåteranvändbarhet

Vissa klasser är väldigt stora (såsom BoardController) och den kan refactoreras in i lite mindre klasser. Om man hade refaktorerat de större delarna i många klasser in i mindre delar kan man mycket lättare återanvända koden som skrivits. Eftersom att många metoder är väldigt långa och gör många olika saker så kan funktionaliteten inte återanvändas som finns i dessa metoder.

Kodunderhållning

Det finns många stora metoder och som dessutom innehåller många if satser och har stor cyklomatisk komplexitet som exempelvis Board.java. Detta gör att de blir svårare att testa som i t.ex. Board.java (checkCol, checkRow, helpcalculatescore). Komplexiteten av metoderna leder till att det blir omöjligt att testa delar av dessa metoder och reda ut var problemet kommer ifrån. Koden blir även svårförståelig vilket gör att den i framtiden blir svårare att debugga och ändra på om någonting är fel.

Skulle vara bra att dela upp dessa stora metoder i flera stycken mindre vilket leder till större testmöjlighet men även bättre förståelse. Alternativt så kan även vissa komplicerade if statements göras om till namngedda booleaner så att man förstår vad det är if statementen faktiskt kollar.

Utöver detta bör single responsibility principle kollas över. Till exempel så gör metoden selectBag i TileBag klassen en del grejer som vi inte riktigt förstår vad det gör, men antar att det väljer en bag på något sätt och sen gör den även någon typ av shuffle i slutet som bör vara en egen funktion. Detsamma går för selectBoard i Board.java där det smidigt har lagts kommentarer för allt som metoden gör, vilket är 4 saker som alla kan delas upp i egna metoder.

Designmönster

Koden använder sig av saker som liknar vissa design mönster, men de utförs inte helt korrekt så det är svårt att kalla dessa för just design mönster. Två exempel på detta är singelton patternet som används i GameManager som instantieras utanför sin egna klass (publik konstruktor). Singeltonen som används i Dictionary är dock korrekt, så följ den. Den andra skumma är LetterObserver som är en konkret klass, bör vara ett Interface.

Utöver detta används typ MVC patternet. Controllern är dock motsatsen till tunn, viss viewkod ligger i BoardControllern, en view ligger i fel package, detta patternet bör ses över för rätt implementation.

Dokumentation

Koden är relativt dokumenterad, men vid vissa ställen stämmer inte dokumentationen. Ett exempel är selectBoard (som kanske mer passande borde heta readBoard) och som i dokumentationen säger att den skapas efter board creation, vilket den inte gör. Detta förstör hela syftet med dokumentation.
Vissa metoder är även “över-kommenterade”, de har kommentarer på varje rad vilket gör koden svår att läsa och man kan tycka att koden kanske borde “dokumentera sig själv” lite mer med hjälp av bättre valda variabelnamn, refactora block till metoder med förklarande namn osv.
Javadoc används dessutom felaktigt genom att skriva Javadoc för metoderna i klasserna som implementerar interfacet. Javadoc borde skrivas på metoderna i själva interface klassen.

Variabel, metod och klassnamn

Många namn är otydliga eller rent av fel. Exempelvis används namnet ILetterObservable till en observer medan ILetterObserver används till objektet som observeras, dess borde byta namn.

Sedan finns det metodnamn som ser ut som substantiv snarare än verb på flera ställen såsom exempelvis “actuallyWritable” istället för “getActuallyWritable” eller “calculateActuallyWritable”.

Många helper metoder börjar även med ordet help såsom “helpCalculateScore“ medan ett namn som “calculateScore” hade varit tydligare. När metoden heter “helpCalculateScore” vet man inte egentligen vad den gör, om den hjälper till att räkna ut score, vilken del av detta gör den då och på vilket sätt?

Koordinatsystemet som används är även i och j. Hade blivit tydligare att bara använda x och y istället eller “vertical” och “horizontal”.

Vissa namn är även önödigt komplexa såsom booleanen “rowAreIndeedValid”, vilken endast borde heta “rowIsValid”.

Modulär design

Generellt sett kan många metoder delas upp i mindre små. De flesta klasserna ser dock helt ok ut, förutom de som är väldigt stora.

Abstraktioner

IGame är ett stort interface som kanske inte riktigt känns nödvändigt. Detta eftersom Game inte har någonting som döljs av interfacet. Dessutom används IGame endast i GameMangerer vilken även är dependant på både Game vilket lite förstör poängen, någon form utav GameFactory bör användas. Det samma gäller Player och IPlayer, även här borde en PlayerFactory användas.

Tester

Koden ser ut att vara testad men dock så körs code coverage på test-klasserna och inte på kodbasen vilket gör att alla tester körs ju och därmed får man 100% coverage. Dock fick den inte 100% coverage utan det finns metoder i test-klasserna som aldrig körs. För övrigt såg det ut som att det mesta testades vilket är bra. (bara en lite sak också, skulle rekommendera att göra test-packaget till en test-package så att intellij fattar att det är det)

Säkerhet och prestanda

Dictionary klassen använder sig i nulägen av en ArrayList med alla ord i ordboken som används. Contains metoden används sedan för att se om ett spelat ord är giltigt. Detta är långsamt eftersom alla strängarna måste jämföras tills ordet hittats. I värsta fall kräver detta lika många jämförelser som ordboken är lång.

Detta borde kunna förbättras substantiellt till en constant time lookup genom att istället använda sig av ett HashSet.

Gällande själva gameplay:en hittades en bug där varje gång du lade ett ord, lades ett likadant ord fast spegelvänt runt diagonalen. De dubbla poängen räknades till hos spelaren. Se figuren till vänster. De spegelvända tilesen kunde dessutom försvinna om du la en tile från ditt rack på den spegelvända och sedan bytte plats på tilen du precis la till en anna plats.

MVC

Projektet har halvt en MVC struktur. Det finns en början till det då det finns ett package till Model, View och Controller. Dock bryts mönstret en del då View mappen är tom och all View-kod ligger i de andra klasserna (t.ex. BoardController). Controllern bör också tunnas ut då mycket view-kod och model-kod ligger i BoardController. Som t.ex. Labels och färger på olika FXML element. Hade varit bra att se också en refactorering på hela model paketet och lägga vissa klasser i mindre paket såsom t.ex. alla klasser som har något med Cell att göra osv. Det gör att man kan använda package-private metoder vilket inte kan göras atm då alla klasser ligger i samma model-paket.