Texas Holdem / Cards

Anton Österberg	Beatrice Beta	Caesar Gamma	Diana Delta
anos3557	bebe5678	caga9012	dide3456

24 oktober 2014

Introduktion

En kort introduction till projektet där ni också listar de verktyg ni använt. Om ert versionshanteringssystem går att komma åt ska adressen dit finnas med, annars ska det finnas en länk från vilken man kan ladda hem källkoden.

Slutlig design

En övergripande modell över systemet. Lämpligt format är ett eller flera klassdiagram, plus eventuella andra modeller som behövs för att förstå hur systemet är uppbyggt. Diagrammen ska vara läsbara. Det är dock fullständigt okej att de är detaljerade, bara det går att zooma in ordentligt på dem. Ett tips är att börja med ett översiktligt diagram som inte innehåller mer än paket och klassnamn, och att sedan lägga till mer detaljerade diagram efter det.

Vi valde att dela upp vårat projekt på ett sådant sett att det finns ett separat paket som heter Cards vilket är tänkt som ett allmänt kortspels-paket med kort, kortlekar, spelare, händer osv och sedan ett paket som är mer specifikt inriktat på kortspelet texasholdem.

Nedan är ett klassdiagram över hela vårt projekt: `https://api.genmymodel.com/projects/_XAN6sEQoEeSWhcMY2AoZUA/diagrams/_XAN6s0QoEeSWhcMY2AoZUA/svg` (UPPDATERA DETTA DIAGRAM INNAN INLÄMNING)

Testdriven utveckling – process

En översikt över hur ni tillämpat TDD med exempel från olika personer och olika faser i projektet. Om ni har använt versionshanteringssystemet ordentligt bör all information som efterfrågas här finnas i det. Tänk på att kodexemplen ska vara läsbara.

Testdriven utveckling – erfarenheter

En diskussion om vilka era erfarenheter ni dragit av att tillämpa TDD. Det finns inget rätt eller fel här. Enda sättet att bli underkända är att bara fuska över punkten och säga något pliktskyldigt.

Vi har arbetat med testdriven utveckling nästan helt och hållet genom projektet vilket har gett oss en mängd erfarenheter av hur bra/dåligt det kan fungera. Till en början kändes det väldigt överflödigt och framtvingat då man arbetade med simpla metoder, klasser och konstruktioner som kändes självklara hur man skulle skriva och som alltid fungerade direkt. Desto mer vi arbetade och kom in i svårare metoder och konstruktioner så började testdriven utveckling kännas mer och mer användbart och mot slutet i projektet, speciellt när vi arbetade med klasses TexasRules, så kändes det nästan som att det hade varit omöjligt att genomföra den delen utan testdriven utveckling och testfall. Eftersom man i klassen TexasRules behövde testa väldigt specifika fall, som t ex att rätt person vinner om båda har stege men en av personerna har högre kort, så gjorde testfall att man enkelt kunde sätta upp ett eller flera komplexa fall som endast gått igenom om koden är korrekt skriven. Detta skulle vara väldigt svårt att testa utan testfall.

Testdriven utveckling ledde också till att vi snabbt upptäckte fel som introducerats i koden. Detta eftersom man kunde se om ett testfall gått igenom tidigare antingen i commit-kommentarerna eller för att man själv kört dem tidigare och då också se om man kan ha introducerat något fel i en specifik del av koden. Lyckades man då inte se vilket fel man introducerat så gick det alltid att titta på versionen som commitats t ex med kommentaren *testfall X går nu igenom* och se hur koden såg ut i det tillfället.

En ytterliggare positiv effekt av testdriven utveckling som vi inte tänkt på i förhand är att den gör koden mycket mer lättförståelig för de man arbetar med då man kan se hur koden är tänkt att fungera även om den inte i nuvarande läge fungerar. Detta leder till att man enkelt kan ta över metoder och skriva klart dem även om det inte var man själv som började skriva på den. Även om man har dokumentation om ungefär vad metoden ska göra, så ger testfallen mycket med information om exakt hur metoden ska bete sig och vilket resultat som förväntas.

Vi har förstås också fått uppleva några utav baksidorna med testdriven utveckling. Det är tidskrävande, speciellt att skriva bra tester som ger god täckningsgrad och täcker alla ekvivalensklasser. Det krävs god disciplin för att inte slarva och börja övergå i ad-hoc-testning, speciellt under tidspress, något som vi kanske inte kan säga att vi alltid lyckats med. Det är inte heller alla gånger som vi varit helt överens om hur vi ska prioritera testbarhet kontra inkapsling. Oftast har vi lyckats finna någon kompromiss där t.ex. privata metoder har varit testbara via publika metoder där

den privata anropas.

Ekvivalensklassuppdelning – subtractChips

En kort presentation av vad ni valt ut för att tillämpa ekvivalensklassuppdelning på. Ni ska kort motivera valet, och ge tillräckligt med information för att det ska gå att bedöma er. Detta avsnitt och de tre föjande (till och med testmatrisen) ska finnas för samtliga delar ni tillämpat ekvivalensklassuppdelning på.

Ekvivalensklasser – subtractChips

Samtliga ekvivalensklasser för denna del presenterade på ett tydligt sätt.
public boolean subtractChips(int bet)

Invalida interval		**Valida interval**	
int bet	$<= 1$	int bet	$0 < bet <= chips$
int bet	$> chips$		

Testfall – subtractChips

Testfallen som ni fått fram från ekvivalensklasserna. Observera att vi inte vill ha någon kod här, utan bara en tydlig presentation av testfallen i någon lämplig tabellform. ?

Testmatris – subtractChips

En testmatris som visar sambandet mellan ekvivalensklasserna och testfallen för denna del.

Ekvivalensklassuppdelning - Game-konstruktor

En kort presentation av vad ni valt ut för att tillämpa ekvivalensklassuppdelning på. Ni ska kort motivera valet, och ge tillräckligt med information för att det ska gå att bedöma er. Detta avsnitt och de tre föjande (till och med testmatrisen) ska finnas för samtliga delar ni tillämpat ekvivalensklassuppdelning på.

Ekvivalensklasser – Game-konstruktor

Samtliga ekvivalensklasser för denna del presenterade på ett tydligt sätt.

Invalida interval		**Valida interval**	
int bigblind	$<= 0$	int bigBlind	> 0
double blindsRaisePercentage	< 0	double blindsRaisePercentage	$>= 0$
Players... players	null	Players... players	*

Testfall – Game-konstruktor

Testfallen som ni fått fram från ekvivalensklasserna. Observera att vi inte vill ha någon kod här, utan bara en tydlig presentation av testfallen i någon lämplig tabellform. ?

Testmatris – Game-konstruktor

En testmatris som visar sambandet mellan ekvivalensklasserna och testfallen för denna del.

Tillståndsbaserad testning

En kort presentation av vad ni valt ut för att tillämpa tillståndsbaserad testning på och vilket täckningskriterium ni valt att använda er av. Ni ska kort motivera valen, och ge tillräckligt med information för att det ska gå att bedöma er. Glöm inte att ta med själva modellen.

Testfall för tillståndsbaserad testning

Testfallen som ni fått fram från tillståndsmaskinen. Observera att vi inte vill ha någon kod här, utan bara en tydlig presentation av testfallen i någon lämplig tabellform. Det ska enkelt gå att mappa testfallen till tillståndsmaskinen.

Kodkritiksystem

En presentation av de problem som hittats med hjälp av verktyg för statisk analys och en diskussion av dem enligt anvisningarna. Det räcker alltså inte med att bara lista problemen, ni måste förhålla er till dem också. Tänk också på att ni ska göra detta både på koden som den såg ut före granskningen och på koden efter att ni rättat det som kommit fram under granskningen.

Statiska mått

En presentation och diskussion kring ett antal lämpliga statiska mått på koden. Att vi inte specificerar exakt vilka mått som ska tas upp beror på att olika verktyg har olika uppsättningar, men vi förväntar oss fler och mer intressanta mått än bara rena storleksmått som LOC, #klasser, #metoder, etc. Även här är det viktigt att förhållas sig till måtten, inte bara lista dem.

Pizza metric index: 1.

Täckningsgrad

En översikt över vilken täckningsgrad era testfall uppnått. Denna kan antagligen tas rakt av från verktyget ni använt för att mäta den. Om ni inte uppnått fullständig täckning så ska detta förklaras och motiveras.

Profiler

En kort presentation av hur ni gått tillväga för att testa koden med en profiler och vilka resultat ni fick fram.

Byggscript

Byggscriptets första (seriösa) version, och den slutliga.

I den första versionen av byggscriptet vad ambitionen främst att det skulle göra ungefär samma sak som den standard-byggscripten som eclipse skapar för ett projekt. Med andra ord, kompilera alla klasser och dess beroenden. Det gick ganska bra fram till dess att testfallen inkluderades, då det krävdes externa bibliotek (JUnit) för att kompilera dessa. Så småningom fick vi även det att fungera.

I den färdiga versionen så körs även testfallen, och om desa går igenom skapas även en jar-fil för respektive paket.

Kod 1: build.xml, första versionen

```
<?xml version="1.0"?>
<project name="texas" default="buildTest" basedir=".">
<property name="build.dir" location="bin" />
<property name="srcCards.dir" value="src/cards" />
<property name="srcTexasHoldem.dir" value="src/texasholdem" />
<property name="srcTest.dir" value="test" />

<property name="test.report.dir" location="testreports" />

<path id="junit.class.path">
        <pathelement location="${eclipse}/plugins/org.junit*/*junit.jar" />
        <pathelement location="/home/anton/dokument/eclipse/plugins/org.junit_4.11.0.v201303080030/junit.jar" />
        <pathelement location="${eclipse}/plugins/*org.hamcrest.*.jar" />
        <pathelement location="${build.dir}" />
</path>

<target name="buildCards" description="Build package cards">
        <mkdir dir="${build.dir}" />
        <javac includeantruntime="false" destdir="${build.dir}" source="1.7" target="1.7">
                <src path="${srcCards.dir}"/>
        </javac>
</target>

<target name="buildTexas" description="Build package texas" depends="buildCards">
        <javac includeantruntime="false" destdir="${build.dir}" source="1.7" target="1.7">
                <src path="${srcTexasHoldem.dir}"/>
        </javac>
</target>

<target name="buildTest" description="Build test cases" depends="buildTexas">
        <mkdir dir="${build.dir}/test" />
        <javac includeantruntime="false" destdir="${build.dir}/test" source="1.7" target="1.7">
                <classpath refid="junit.class.path" />
                <src path="${srcTest.dir}"/>
        </javac>
</target>
</project>
```

Kod 2: build.xml, slutgiltliga versionen (1/2)

```
<?xml version="1.0"?>
<project name="texas" default="buildJarFiles" basedir=".">
<property name="build.dir" location="bin" />
<property name="cards.src.dir" location="src/cards" />
<property name="texasHoldem.src.dir" location="src/texasholdem" />
<property name="test.src.dir" location="test" />
<property name="test.report.dir" location="testreports" />
<property name="jar.dir" location="jar" />
<property name="eclipse" location="/home/anton/dokument/eclipse" />

<path id="junit.class.path">
        <fileset dir="${eclipse}/plugins" includes="org.junit*/*junit.jar" />
        <fileset dir="${eclipse}/plugins" includes="org.hamcrest*/*junit.jar" />
        <fileset dir="${eclipse}/plugins" includes="org.hamcrest.core*.jar" />
        <pathelement location="${build.dir}" />
</path>

<target name="clean" description="Delete␣previous␣jar␣and␣class␣files">
        <delete includeemptydirs="true" dir="jar" />
        <delete includeemptydirs="true" dir="bin" />
</target>

<target name="compileCards" description="Compile␣package␣cards" depends="clean">
        <mkdir dir="${build.dir}" />
        <javac includeantruntime="false" destdir="${build.dir}" source="1.7" target="1.7">
                <src path="${cards.src.dir}"/>
        </javac>
</target>

<target name="compileTexas" description="Compile␣package␣texasholdem" depends="compileCards">
        <javac includeantruntime="false" destdir="${build.dir}" source="1.7" target="1.7">
                <src path="${texasHoldem.src.dir}"/>
        </javac>
</target>
```

Kod 3: build.xml, slutgiltliga versionen (2/2)

```
<target name="compileTest" description="Compile␣test␣cases" depends="compileTexas,␣compileCards">
        <javac includeantruntime="false" destdir="${build.dir}" source="1.7" target="1.7">
                <classpath refid="junit.class.path" />
                <src path="${test.src.dir}"/>
        </javac>
</target>

<target name="runTestCases" description="Run␣test␣cases" depends="compileTest">
        <junit showoutput="true" printsummary="true" fork="yes" haltonfailure="yes">
                <classpath refid="junit.class.path" />
                <classpath>
                        <pathelement location="${build.dir}"/>
                </classpath>
                <formatter type="plain" usefile="false" />
                <batchtest todir="${test.report.dir}">
                        <fileset dir="${build.dir}">
                                <include name="*Test.class" />
                        </fileset>
                </batchtest>
        </junit>
</target>

<target name="buildJarFiles" description="Build␣.jar-files" depends="compileCards,␣compileTexas,␣compileTest,␣
    runTestCases">
        <mkdir dir="${jar.dir}" />
        <jar destfile="${jar.dir}/cards.jar" basedir="${build.dir}/cards" includes="*.class" />
        <jar destfile="${jar.dir}/texasholdem.jar" basedir="${build.dir}/texasholdem" includes="*.class" />
</target>

</project>
```