Slutrapport Utvecklingsprojekt

OOP1, h1 2012, distant period 2

Den 13 augusti 2013

LOTTEN WESTER

lottenwester@hotmail.com

SLUTRAPPORT UTVECKLINGSPROJEKT

**Problembeskrivning**

Bakgrund

Jag valde att skapa ett ordspel där man utefter sju givna bokstäver skulle hitta alla möjliga ord man kunde skapa av dessa bokstäver. För varje hittat ord får man poäng. Om man hittar det sjubokstaviga ordet innan den utsatta tiden för spelomgången får man sju nya bokstäver och tiden nollställs.

Jag hade från början gjort ett någorlunda fungerande sådant ordspel i java, men det bestod bara av tre klasser.

Frågeställningar

Det som jag behövt lösa i detta projekt är i första hand hur jag på bästa sätt skulle strukturera upp de i stort sett sekventiella kodblocken i de tre gigantiska ursprungsklasserna för att istället skapa en mer objektorienterad struktur. Jag behövde framförallt fundera på hur jag fick en så lös koppling som möjligt samt en så tät kohesion som möjligt. Spelet fungerade även utan detta men poängen med omstruktureringen var att använda mig av de kunskaper jag fått under kursens gång.

I andra hand behövde jag lösa datalagringen av bokstäverna och de möjliga orden knutna till specifika bokstäver. Detta för att inte varje gång programmet körs behöva göra alla de iterationer genom ordlistan som krävs för att få fram dessa data.

Jag har inte fokuserat på poängräkningen och tiden, varken funktionellt eller med tanke på spelkänslan. Orsaken är att jag anser att utifrån att detta är ett utbildningsprojekt är det viktigare att se till projektets övergripande klasstruktur än till funktionsdetaljer.

**Metod**

Jag arbetade mig fram stegvis mellan design och implementation.

Till att börja med gjorde jag en design, och sedan provade jag att skriva klasser på detta sätt. När jag sen stötte på problem med min ursprungliga idé av struktur tog jag ett steg tillbaka och ritade upp en ny design utifrån ett MVC-mönster. När jag under kodningens gång upptäckte behovet av att förändra de klasserna jag ursprungligen tänkte ha gick jag tillbaka till designen och tänkte ut hur jag bäst skulle förändra upplägget.

**Resultat**

Output och tydligare beskrivning av spelidén

Spelet består av en körbar jar-fil. På spelplanen finns sju bokstäver och alla möjliga rätta gissningar ritas upp ungefär som när man ska leta efter ett ord i hänga gubbe (fast här är det många ord man ska leta efter):

skärmdump_1.tiff

Sen skriver man in en gissning och klickar på enter. Ifall gissningen är rätt fylls ordet i:

skärmdump_2.tiff

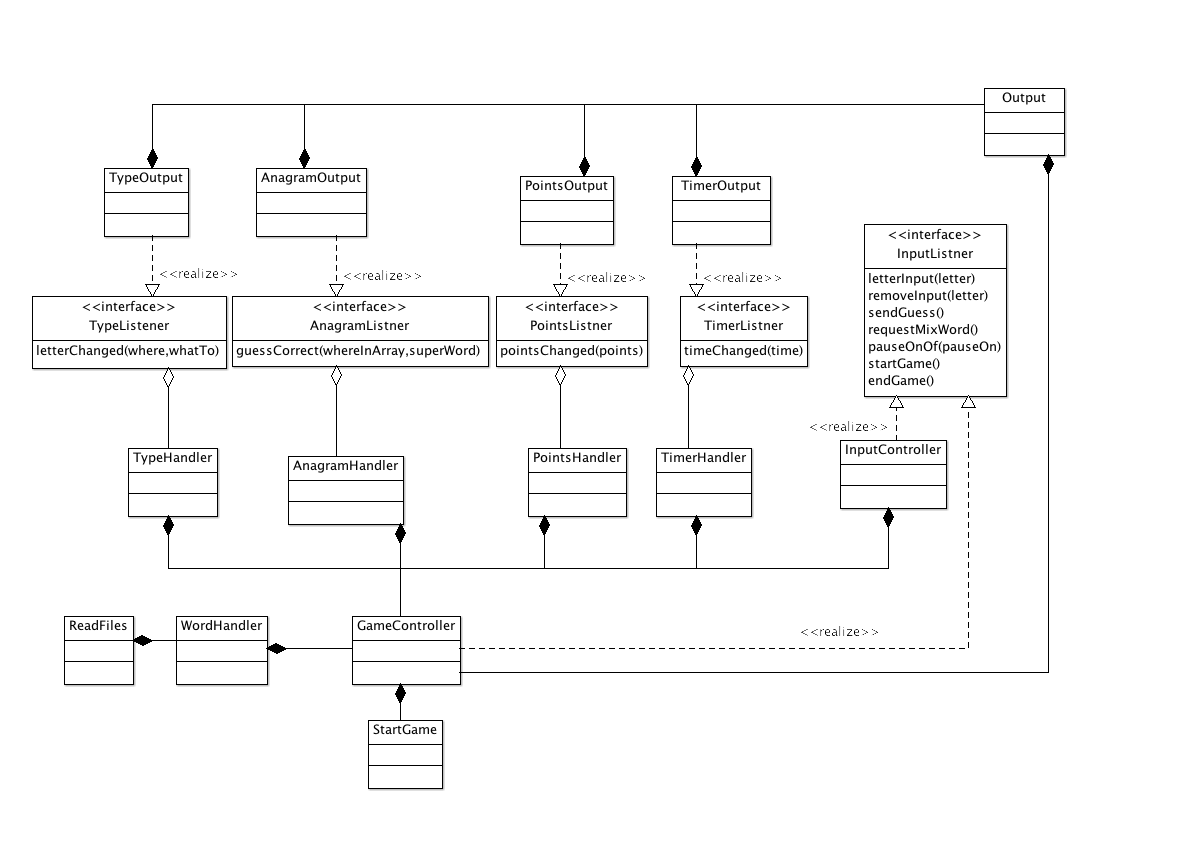
skärmdump_3.tiff

Man gör fler gissningar och fler ord fylls i:

skärmdump_4.tiff

För varje hittat ord får man poäng. Om man hittar det sjubokstaviga ordet innan den utsatta tiden för spelomgången får man sju nya bokstäver och tiden nollställs.

Klasser

****

*Controller*

Startgame innehåller en main-metod som skapar en GameController.

GameController innehåller ett output-objekt som är själva spelfönstret, och objekt som hanterar spellogiken.

Inputcontroller hanterar musklick och knapptryckningar från output och genererar utifrån det händelser till gamecontroller. Gamecontroller skickar sedan i sin tur vidare denna information till de handlerklasser som berörs av inputen.

*Model*

I modellen ligger klasserna som hanterar spellogiken. Handler-klasserna hanterar klasser som hör till deras interface, klasser som specifikt lyssnar till instruktionerna från respektive handler.

Till exempel hanterar TimerHandler klasser som har interfacet TimerListner. I modellen är TimerOutput en lyssnare till TimerHandler.

TypeHandler hanterar in- och utmatningen av bokstäver från spelaren. Konstruktorn i TypeHandler tar emot en String som är de rätta gissningsbokstäverna. Utifrån sin metod inputLetter(char c) som aktiveras av att inputListnern skickar en bokstavsinmatningshändelse till controllern bedömer den om den ska skicka en eller flera av följande händelser till sina lyssnare: skriva ut bokstav (i ett inmatningsfält i output), förändra statusen av en av bokstäverna spelaren kan gissa utifrån till använd eller oanvänd.

TypeHandler skickar även till sina lyssnare om bokstäverna i outputs gissning ska ändras.

GuessHandler hanterar spelarens gissningar, det vill säga i denna output när spelaren trycker på enter eller på knappen Gissa tar den TypeHandlers sparade input och kontrollerar om det är en korrekt gissning. Är det en korrekt gissning talar GuessHandler om det för sina lyssnare.

PointsHandler och TimeHandler hanterar och uppdaterar poäng-, och tidsuträkningar.

*View*

Klassen GameOutput ärver av klassen Frame och skapar ett fönster. Det innehåller andra klasser som är lyssnare till de olika händelsehanterarna. Output skickar händelser till GameControllern genom InputControllern.

Det finns ett antal design-klasser som ärver av JLabel respektive JFrames.

*Datalagring*

WordHandler innehåller ett ReadFiles-objekt som läser in orden och returnerar en ArrayList med Word-objekt. Word-objekt innehåller en String[] med ordet eller orden som finns till en specifik bokstavskombination, och också en String[] med kortare ord som kan skapas med hjälp av den specifika bokstavskombinationen.

WordHandler slumpar fram ett Word från sin Word-ArrayList.

ReadFiles läser en textfil som har sparat ord i följande format:

Absenta a,absent,absenta,an,ana,anas,anat,anats,ans,ansa,ansat,anse,anta,antas,ante,as,asa,asat,asen,aset,ban,bana,banas,banat,banats,bane,bans,banta,bantas,bas,basa,basat,basen,bast,basta,be,beat,beats,ben,bena,benas,benat,benats,bens,bes,best,bet,beta,betan,betans,betas,bets,betsa,en,ena,enas,enat,enats,ens,ensa,ensat,est,et,etsa,nasa,nasat,nate,nates,nesa,sa,sant,satan,sate,saten,se,sen,sena,senat,sent,set,sta,stab,staben,stan,sten,stena,ta,tas,te,ten,tens,tes

Det inklistrade stycket är en rad i textfilen, med blanksteg efter själva ordet och komma mellan de kortare orden. Textfilen består av alla sjubokstaviga ord och orden som kan skapas med de sju bokstäverna.

Det finns ett paket i projektet som heter Create där jag experimenterat med hur de färdiguträknade orden bäst ska framställas och lagras. Som det ser ut nu är bara de sjubokstaviga orden lagrade, och det är bara det första sättet jag

Jag provade en idé om att skapa en klass LetterCombo som sparar alla möjliga ord med en viss bokstavskombination, och vilka kortare bokstavskombinationer som också har möjliga ord. Sedan skulle jag på det sättet kunna bygga ett slags dataträd där man kunde välja antalet bokstäver spelet skulle använda. I LetterCombo la jag in denna metod:

**public** ArrayList<String> getAllWordsWithTheseLetters(){

**if**(allWords.size() == 0){

allWords.addAll(anagrams);

**for**(LetterCombo l : matchingLetterCombos){

allWords.addAll(l.getAllWordsWithTheseLetters());

}

Collections.*sort*(allWords, c);

}

**return** allWords;

}

matchingLetterCombos är då en ArrayList med de kortare bokstavskombinationerna, och anagrams är ett ArrayList innehållande textsträngar bestående av orden som går att göra med det specifika LetterCombo-objektets unika bokstavskombination.

Till exempel på bokstavskombinationen AKR innehåller anagrams ARK, KAR, RAK och matchingLetterCombos innehåller letterCombo-objekten AR och A. Hade jag fortsatt på projektet hade jag sedan satt mig in i hur serializable-interfacet fungerar för att kunna lagra alla letterCombo-objekten.

Efter detta skulle jag skapa klass som LetterComboHandler för att plocka fram en lista på spelbara LetterCombos med X bokstäver, X min - X max möjliga gissningsbara ord där de gissningsbara orden är av längden X min – X max och sorteras efter Stringcomparatorn X. På så sätt skulle det gå enkelt att skapa olika svårighetsgrader och nivåer ifall man byggde vidare på spelet.

**Diskussion**

Datalagring

Jag tror att det hade varit bättre att lagra data om bokstavskombinationerna i sparade bokstavskombinationsobjekt samt ett objekt som höll ordning på och plockade fram rätt bokstavskombination eftersom det hade inneburit en mycket större flexibilitet. Objektet som hanterade alla bokstavskombinationer skulle då kunna ge andra objekt data om såväl vilket som var det gällande spelordet som vilka inställningar som använts för att plocka fram ordlistan. Tyvärr kan jag fortfarande för lite om datalagring för att lista ut det bästa sättet detta skulle genomföras på praktiskt. Värdet av flexibla ordlistor blir att ifall jag skulle vilja ha fram en viss ordlista som till exempel en nybörjarnivå skulle jag inte behöva en specifik fil för detta.

Lös koppling

Jag försökte en lösare koppling genom att dela upp spelet i fler mer oberoende byggstenar, till exempel en del som hanterade tiden. Detta tyckte jag att jag lyckades helt ok bra med, uppdelningen känns naturlig.

Det hade varit fördelaktigt för att uppnå en lösare koppling om jag hade fått ordning på datalagringen. Eftersom ”spelord-objekt” är så centralt i spelet styr deras uppbyggnad många av de andra klasserna. Som spelet ser ut nu innebär det en del onödigheter. Till exempel tar klassen GuessHandler emot två parametrar, en int som

beskriver ordets längd och en String[] med de möjliga ordgissningarna. Detta beror på att jag inte från början tänkte ut hur jag skulle lagra olika bokstavskombinationer och möjliga ord. Det hade varit mer naturligt om GuessHandler plockade information från ett ordliste-objekt istället för att det är uppbyggt som nu, att GameControllern hämtar information från WordHandlern och sen skickar denna information vidare till GuessHandlern.

**Syftet**

Jag lyckades få en lösare koppling i min struktur och jag kände att jag förbättrade mig i mitt objektorienterade tänkande genom att verkligen fokusera på just objekt istället för funktion.

Även om jag lyckades skapa en fungerande datalagring var den väldigt enkel vilket ledde till en sämre flexibilitet. Jag löste alltså datalagringen till hälften.

Jag har uppnått mitt viktigaste mål som var att lära mig mer om struktur.