ColorHack

Inlämningsuppgift

Objektorienterad programmering

DA348G – VT 2012

Emma Jonsson

a11emmjo@student.his.se

Institutionen för kommunikation och information

Högskolan i Skövde

1 Introduktion

Uppgiften var att skapa antingen ett kort- eller brädspel med hjälp av objektorienterade metoder i programmeringsspråket Java. Detta skulle sedan gå att spela på antingen en dator eller en android-mobil. Kravet var att spelet skulle ha ett givet mål, kräva interaktion av spelaren och utgången skulle kunna påverkas av spelarens skicklighet och/eller erfarenhet. Utvecklingen av spelet skulle delas upp i fyra delmoment där man i det första skulle ta fram en kravspecifikation för spelet. I det andra skulle man skapa en så kallad ”quick & dirty”-prototyp och i det tredje skulle man skapa en objektorienterad design utifrån kravspecifikationen och prototypen. I det fjärde och sista delmomentet skulle man implementera spelet efter den design som man tagit fram.

Det spel som valdes var något som kallas MasterMind. Spelet går ut på att en ”färgkod” skapas genom att fyra av sex möjliga färger placeras ut på rad och sedan skall färgkoden listas ut genom flertalet gissningar. Om man valt en färg som ingår i koden men placerat den på fel plats markeras detta med en vit markering. Om man däremot placerat rätt färg på rätt plats markeras detta med en svart markering. Markeringarna sätts vid sidan om varje rad och säger därmed ingenting om vilken plats i ordningen som de representerar men är trots detta en hjälp till att klura ut lösningen. Målet är att lista ut färgkoden på så få gissningar som möjligt.

Syftet med denna rapport är att sammanställa hela utvecklingsprocessen i kronologisk ordning och därmed kommer första kapitlet därför att börja med att ta upp den kravspecifikation som tagits fram. Därefter följer ett kapitel som beskriver den design som ovan nämnda delmoment två och tre resulterat i. Detta kommer sedan att följas av ett kapitel som diskuterar det färdiga spelet, och till sist avslutas rapporten med en sammanfattning av hela uppgiften tillsammans med de erfarenheter och tankar som författaren av denna rapport har införskaffat sig.

2 Kravspecifikation

Detta kapitel är uppdelat i tre delar där den första delen tar upp hur originalspelet MasterMind går till i verkligheten. I den andra delen kommer den första digitala idén och föreställningen av spelet som skall skapas att beskrivas. Till sist så kommer en första kravlista att sammanställas.

2.1 Originalet MasterMind

Originalet är ett gammalt brädspel från 1970-talet som spelas av två spelare. Spelet består av en spelplan med massa hål i, massa stora piggar i sex olika färger och massa små vita och svarta piggar enligt Bild 1 nedan.

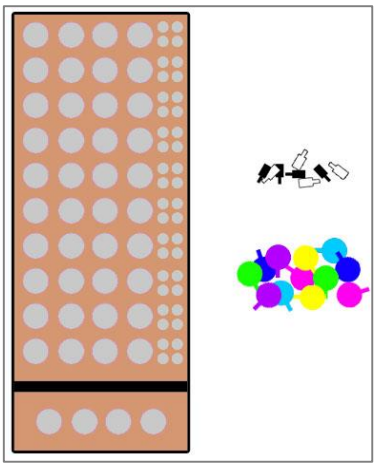

Bild 1 Spelplanen och dess tillbehör i brädspelet MasterMind

2.1.1 Förberedelser

Spelare 1 börjar med att välja fyra stora piggar i olika färg och placerar sedan dessa på rad i en del på spelplanen som är dold för spelare 2. I Bild 2 nedan kan man se ett tjockt svart streck som skall representera en skärm som döljer färgkoden. Bakom densamma så har en färgkod placerats ut.

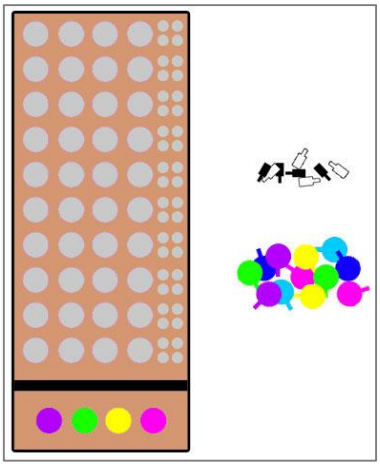

Bild 2 Utplacerad färgkod

2.1.2 Spelomgången

Spelare 2 börjar med att välja fyra stora piggar och placerar sedan ut dessa i valfri ordning i en rad på spelplanen. Spelare 2 placerar alltså ut sin gissning på vilka färger och dess placering som hon tror finns

i den dolda delen på spelplanen. Se exempel på en gissning på övre raden i Bild 3 nedan. På varje vågrät rad kan en ny gissning placeras.

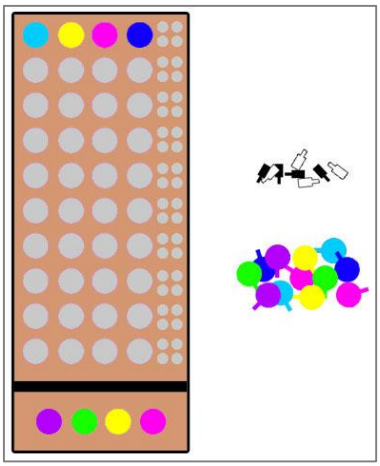

Bild 3 Första gissningen lagd

Spelare 1 kontrollerar därefter gissningen och ger feedback i form av svarta eller vita små piggar som placeras bredvid den rad som representerar gissningen. Om gissningen innehåller piggar som är av rätt färg på fel plats, så sätts lika många små vita piggar ut. Om gissningen innehåller piggar som är av rätt färg på rätt plats, så sätts lika många små svarta piggar ut. I exemplet i Bild 4 nedan så stämmer två utav de färger som föreslagits i gissningen, men de är placerade på fel plats, därav två vita piggar som feedback. Som synes på bilden så sätts feedbacken inte i en rad utan har istället formationen av en kvadrat, detta innebär att piggarna inte säger något om vilka färgpiggar som är korrekta.

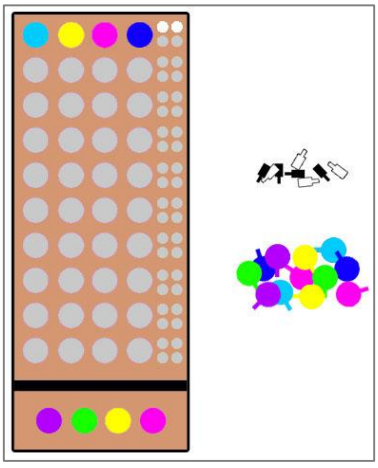

Bild 4 Feedback på första gissningen

2.1.3 Målet

Målet med spelet är att klura ut vilka färger som spelare 1 har placerat ut och i vilken ordning, och om spelare 2 inte lyckas placera ut en gissning som är identisk med den dolda färgkoden får hon försöka igen. Med hjälp av feedbacken så vet hon nu om hon skall byta ut några färger eller om det kanske räcker med att byta den inbördes ordningen på de färger hon redan använt. Spelare 2 placerar alltså ut en ny gissning på en ny rad, som spelare 1 därefter kontrollerar. Denna process upprepas sedan tills spelare 2 antingen har hittat färgkoden, eller tills platserna där man kan placera piggar på spelplanen tar slut. I exemplet i Bild 5 nedan så krävdes det fyra gissningar innan man kom fram till en korrekt kombination.

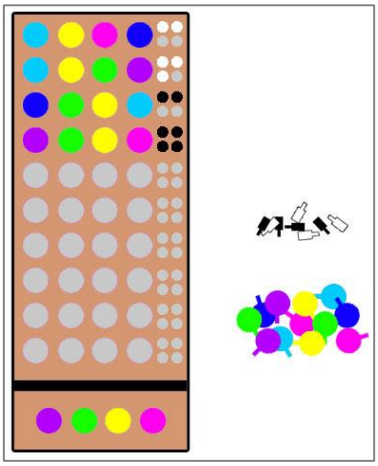

Bild 5 Iteration tills rätt färgkod hittas

2.1.4 Utmaningen

Utmaningen för Spelare 2 är att med logiskt tänkande försöka ta till vara på den feedback som ges och tolka den på rätt sätt för att med dess hjälp lösa färgkoden på så få gissningar som möjligt. Allt eftersom man blir mer erfaren så blir det lättare och lättare att ”se mönstret” och tolka feedbacken.

2.2 Den planerade digitala versionen

Då den planerade versionen av spelet var tänkt att fungera på en Android-mobil med mycket liten skärm (Sony Ericsson Xperia Mini) så innebar det att spelytan behövde vara ganska kompakt, samtidigt som man var tvungen att tänka på att objekten på skärmen inte blev för små för att kunna hantera med ett finger. En första prototyp av spelet skapades i all hast i ett kollegieblock (Se Bild 6).

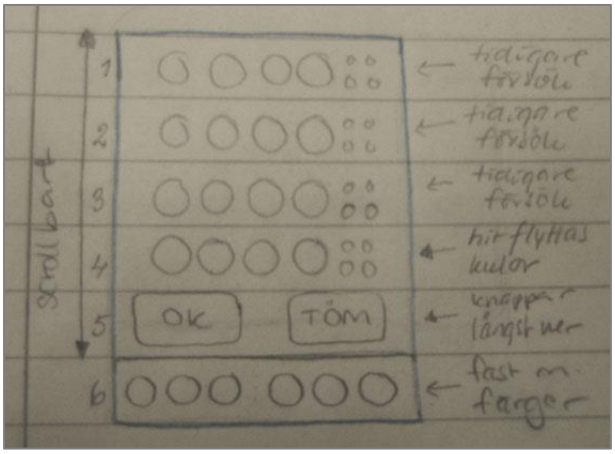

Bild 6 Första prototypen

Tanken som fanns när prototypen skapades var att fönstret skulle bestå av sex synliga rader enligt Bild 6 på föregående sida. Rad 1-3 skulle innehålla antingen tomma rader eller rader som innehöll tidigare gissningar. Rad 4 skulle vara den rad där man konstruerar sin gissning. Rad 5 skulle innehålla knappen ”OK” som man tryckte på när man var nöjd med sin gissning, och knappen ”Töm” som skulle tömma gissningen på rad 4 om man ville börja om med att konstruera sin gissning. På rad 6 skulle de tillgängliga kulorna finnas för i den digitala versionen skulle kulor istället för piggar användas.

Om fler än tre gissningar gick åt till att lösa färgkoden skulle de tidigare gissningarna flyttas uppåt och hela tiden ge plats på rad 3 för den senaste gissningen. Hela fönstret skulle sedan vara scrollbart så att man kunde scrolla upp för att se tidigare gissningar. När man scrollade upp skulle raderna som innehöll kulor, knappar och konstruktionsraden scrolla ner och försvinna ur synfältet i och med att man ändå inte behövde dessa när man ville kolla på tidigare gissningar.

En dold meny skulle finnas som innehöll möjligheten att läsa reglerna, highscore, starta om och avsluta spelet. Om man valde att starta om eller avsluta skulle lösningen visas. Man skulle även kunna nollställa highscorelistan.

Man skulle inte kunna flytta upp kulor till rad 1-3, men man skulle kunna flytta dem fram och tillbaka mellan olika platser både på rad 4 och 6.

2.3 Kravlista

Nedan sammanställs alla de krav som framkommit och diskuterats tidigare i detta kapitel i en punktlista tillsammans med ytterligare några krav. Dessa krav är baserade på den första prototypen och kan därför komma att ändras under projektets gång.

- Spelfönstret skall vara anpassat till Sony Ericsson Xperia Mini.
- Spelfönstret skall bestå av sex synliga rader.
- Spelfönstret skall gå att scrolla.
- Om man scrollar skall raderna som innehåller konstruktionsraden, knapparna och raden med kvarvarande kulor scrolla med och försvinna.

- Rad 1-3 skall visa tidigare gissningar.
- På rad 4 skall den nya gissningen konstrueras.
- Rad 5 skall innehålla knapparna ”OK” och ”Töm”.
- Rad 6 skall innehålla de kulor som finns till förfogande.

- Knappen ”OK” skall man trycka på när man är nöjd med konstruktionen av sin gissning.
- Knappen ”Töm” skall man trycka på när man vill börja om med sin konstruktion.

- Nya gissningar skall hamna på rad 3 och därför flytta upp eventuella tidigare gissningar.
- Man skall ha obegränsat antal gissningar.
- Gissningar skall sparas och överföras till en highscorelista när spelet är slut.

- Det skall finnas sex spelbara färger.
- Färgkoden skall slumpas fram av datorn.
- Färgkoden skall bestå av fyra färger.
- Rätt färg men fel plats skall markeras med vitt.
- Rätt färg och rätt plats skall markeras med svart.
- Kulorna skall gå att flytta mellan alla platser både på och emellan rad 4 och rad 6.

- En dold meny skall finnas med funktionerna regler, highscore, omstart och avsluta.
- Highscoren skall gå att nollställa.
- Om man väljer att starta om eller avsluta skall lösningen visas.

3 Design och implementation

Detta kapitel är uppdelat i två delar och handlar om hur spelet byggts upp både i utseende och funktion. Den första delen handlar därmed om slutproduktens design och varför den blev som den blev, medan den andra delen går igenom resultatet av implementationen, dvs. alla de delar som spelet slutligen består av.

3.1 Designen

I denna del presenteras designens olika delar och jämförs mot kravspecifikationen.

3.1.1 Inte Android

Från början var det som nämnts tidigare tänkt att programmet skulle skapas som en androidapplikation men eftersom författaren av denna rapport (och därmed producenten av slutprodukten) hade väldigt knappa kunskaper inom objektorientering, så ansåg hon det inte lämpligt att ge sig på produktionen av en androidapplikation då produktionen av ett program baserat i Java var nog så svårt. Spelet anpassades därför till att köras på en dator istället.

3.1.2 Spelplanens design

Då spelet var tänkt att istället köras på en dator så behövde inte längre placeringen av knappar, gissningsrad, feedback osv. bestämmas utifrån upplösningen på en skärm på en Sony Ericsson Xperia Mini, utan kunde nu istället designas att ta upp mera plats. Dock utgick designen ifrån samma grund som tidigare och resultatet blev enligt Bild 7.

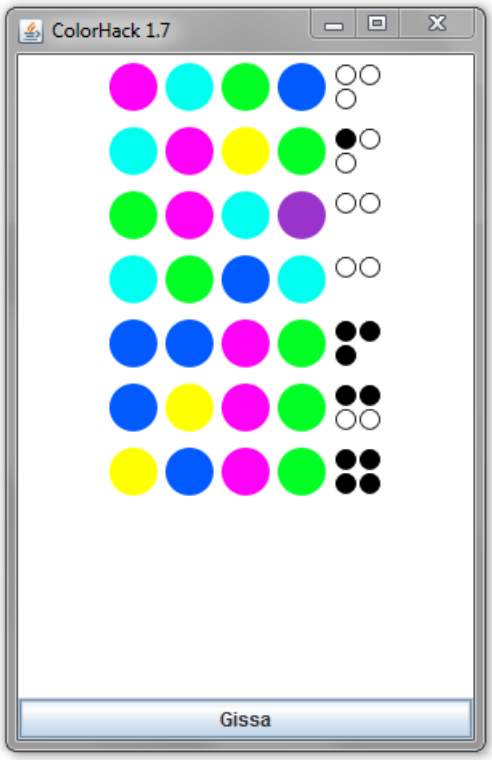

Bild 7 Slutproduktens design

Som synes består inte längre spelplanen utav sex rader utan utav totalt tio spelbara rader. På Bild 7 så kan man se att sju stycken rader har i detta fall använts innan koden har knäckts och omgången har nått sitt slut. Man behöver inte heller längre kunna scrolla mellan sina tidigare försök.

Knapparna ”OK” och ”Töm” har ersatts utav en enda ”Gissa”-knapp. ”Töm”-knappen upplevdes som överflödig då det inte är så många kulor att nollställa om man vill börja om med att placera ut sin gissning och därmed fick den så kallade ”OK”-knappen istället heta ”Gissa” som mer direkt beskriver den funktion den faktiskt har.

Raden som var planerad längst ner, som skulle visa de kulor som fanns tillgängliga att placera ut och därmed gissa med, togs bort av en anledning som beskrivs i nästa stycke, 3.1.3.

3.1.3 Kulorna och deras funktion

Från första början var det tänkt att gissningsraden skulle fyllas med kulor som drogs dit ifrån raden med tillgängliga kulor. Då detta gjorde programmet onödigt komplicerad för en nybörjare i Java och objektorientering att implementera, så löstes detta istället med att färglösa (grå) kulor direkt placeras ut i gissningsraden. Dessa kulor kan man sedan, var och en, klicka på, och i tur och ordning växlar kulan färg mellan de befintliga färger som finns att välja på. När man väl har klickat på en kula en gång så

återgår den aldrig till att bli färglös (grå) igen. Precis som det var tänkt, så finns det sex färger att iterera emellan.

3.1.4 Koden

Koden skapas, som tänkt, av fyra kulor som slumpas fram av datorn. Den består dock inte av fyra olika färger utan kan bestå av ända upp till fyra stycken av samma färg. Detta gör koden svårare att knäcka, vilket sågs som en fördel att ta vara på när implementationen råkade resultera i detta, så därför behölls denna egenskap.

3.1.5 Feedbacken

Även feedbacken följer kravspecifikationen. När användaren trycker på gissa knappen, se Bild 7 på förra sidan, så placeras feedbacken ut. Svarta feedbackkulor markerar antal kulor som har rätt färg på rätt plats, och vita feedbackkulor markerar antal kulor som har rätt färg men som har placerats på fel plats.

3.1.6 Popup-fönster

Då programmet anpassades för en dator så valdes information till användaren under spelets gång, att visas som popup-fönster. Det finns i programmet fyra olika popup-fönster. Det första, se Bild 8, välkomnar användaren till spelet och informerar denne om spelets regler och hur programmet fungerar.

Bild 9 visar det popup-fönster som dyker upp om användaren försöker skicka in en ogiltig gissning, dvs. att alla kulor i gissningsraden inte har tilldelats en färg.

Det tredje popup-fönstret som användaren kan stöta är det på Bild 10. Det visas om användaren knäcker koden innan spelomgången tagit slut, dvs. innan de tio gissningsförsök som användaren har på sig, tagit slut.

Det sista popup-fönstret som användaren kan råka ut för är det som visas om användaren inte knäcker koden på tio försök. Då visas Bild 11.

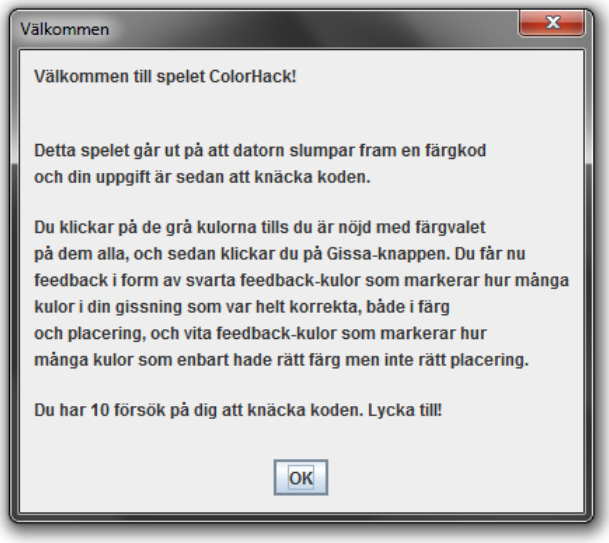

Bild 8 Dialogfönstret för Välkommen

Bild 9 Dialogfönstret för Ej giltig gissning

Bild 10 Dialogfönstret för Vinst

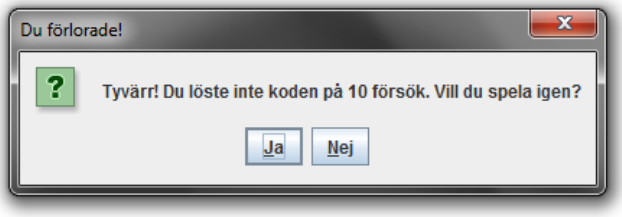

Bild 11 Dialogfönstret för Förlust

3.1.7 Inte obegränsat med gissningar

Önskemålet i kravspecifikationen var att man skulle ha obegränsat antal med gissningar för att lösa koden men detta blev ett problem vid implementationen. Då en layout-variant som godtog ett okänt antal rader inte kunde hittas blev lösningen istället att specificera antalet till tio och därmed så låstes även antalet gissningsrader som kunde få plats

på skärmen till tio. Detta innebar naturligtvis att antalet gissningsförsök som användaren fick på sig fick också det begränsas till tio.

3.1.8 Ingen highscore-lista

Programmet fick tyvärr inte någon highscore-lista som det var tänkt. För att en highscore-lista skulle ha fungerat korrekt så skulle man behöva låta Java spara resultatet i en fil som sedan lästes och skrevs om och det var något som hamnade längst ner på önskelistan och som till slut inte hanns med att implementera.

3.1.9 Ingen meny

Då programmet implementerades för att kunna köras på en dator var kravet på en meny även det, något som prioriterades bort. På en dator kan man istället enkelt klicka på fönstrets X-knapp när man vill avsluta programmet och det finns ingen highscore-lista som skulle kunna ha hämtats fram via menyn.

3.2 Implementationen

I denna del diskuteras implementationen av slutprodukten, de problem som dök upp under tiden, och hur allt hänger samman.

3.2.1 UML

Ett krav på uppgiften var att slutprodukten skulle presenteras i ett UML-diagram så i Bilaga A kan man se just detta. I följande text kommer varje klass att beskrivas mer ingående.

3.2.1.1 Klassen ColorHack

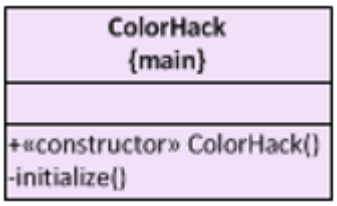

Bild 12 Klassen ColorHack

Grunden till spelet och den klass som innehåller main-metoden, är ColorHack, se Bild 12. Klassen innehåller förutom main-metoden även en konstruktor som har som enda uppgift att kalla på metoden initialize(). Det initialize gör är att skapa programmets fönster genom att skapa ett nytt JFrame-objekt och initiera dess olika egenskaper. Därefter skapas ett nytt GameBoard-objekt som placeras i JFrame-fönstret. Det är detta som leder spelet vidare. Även en knapp skapas som initieras till att få sina instruktioner av GameBoard-objektet. Som synes i Bilaga A så **har** klassen ColorHack en GameBoard, dvs. den består mer eller mindre utav GameBoard.

3.2.1.2 Klassen GameBoard

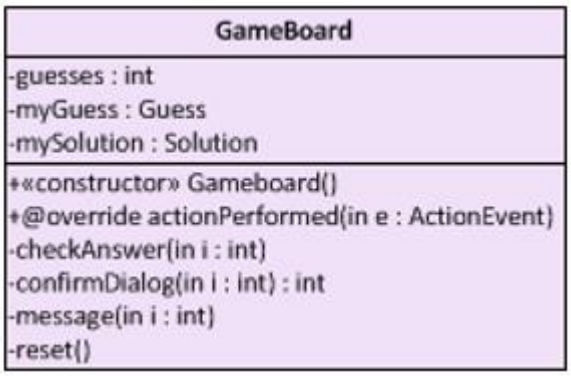

Bild 13 Klassen GameBoard

GameBoard är den klass som bland annat talar om hur innehållet i spelet skall placeras ut på skärmen, se Bild 13. GameBoard ärver ifrån JPanel och kan med dess metoder initiera fönstrets innehåll.

När ett nytt GameBoard-objekt skapas så körs dess konstruktor som initierar spelplanens layout, och därmed det som blir synligt i programmets fönster. Med hjälp utav den ärvda metoden setLayout() och metoder i klassen GridLayout skapas en layout i form av en tabell med tio rader och en kolumn. Därefter kallar konstruktorn på metoden reset() som sätter räknaren som håller koll på antalet gissningsförsök till 0. Sedan skapas en ny färgkod genom att skapa ett nytt Solution-objekt, se 3.2.1.5. Till sist så skapar reset() en ny gissningsrad via ett nytt Guess-

objekt (se 3.2.1.6) och placerar ut det på skärmen och sedan avslutas reset() och programmet återgår till konstruktorn. Konstruktorn utför som sista uppgift att kalla på metoden message() för att skriva ut ett meddelande. Eftersom heltalet 1 skickas med som argument så skrivs ett välkomstmeddelande ut på skärmen och sedan kan spelet börja.

Den överskuggade metoden actionPerformed() är den metod som kallas på när användaren trycker på knappen i programmets fönster. Detta görs automatiskt tack vare att klassen GameBoard implementerar gränssnittet ActionListener. Det den gör först av allt är att kontrollera om kulorna i gissningsraden har blivit tilldelade en färg. Har de inte det så kallas även här metoden message() men nu är argumentet istället satt till heltalet 2 vilket resulterar i att metoden message() istället skriver ut ett meddelande om att alla kulor måste tilldelas en färg innan spelet kan gå vidare.

Om alla kulor i gissningsraden har blivit tilldelade en färg så skapas ett nytt FeedBack-objekt (se 3.2.1.7) och initieras till resultatet av att man skickar in Solution-objektet och Guess-objektet för jämförelse. När detta gjorts så adderas FeedBack-objektet till slutet på den rad som innehåller gissningsraden dvs. Guess-objektet, och sedan skrivs den nya datan ut på skärmen.

Därefter kontrolleras feedbacken med instansmetoden isCorrect() i klassen FeedBack (se 3.2.1.7) för att se om användaren löst färgkoden. Om detta inträffat så kallas metoden confirmDialog() som skriver ut ett frågemeddelande på skärmen. Eftersom argumentet 1 skickas med så skrivs information om att användaren löst koden ut följt av frågan om användaren vill spela igen. Beroende på användarens svar så returneras ett heltal som direkt skickas vidare till metoden checkAnswer(). Om svaret var ja så returneras heltalet 1 vilket resulterar i att metoden reset() återigen körs och spelet förbereds för en ny spelomgång med allt vad det innebär. Om svaret däremot var nej så returneras istället heltalet 2 som resulterar i att programmet avslutas.

Om användaren inte löst koden så kontrolleras istället om det maximala antalet gissningar, som användaren har tillgodo, har uppnåtts. Har det inte gjort det så ökar gissningsräknaren med 1 och därefter skapas ett nytt Guess-objekt som sedan placeras ut på skärmen. Om däremot det maximala antalet gissningar har uppnåtts, så kallas även här metoden confirmDialog() men nu skickas istället heltalet 2 med som argument vilket leder till att ett meddelande om att användaren har förlorat skrivs ut följt av frågan om användaren vill spela en ny omgång. Beroende på användarens svar, precis som tidigare, så returneras ett heltal som sedan direkt skickas vidare till metoden checkAnswer() och detta resulterar antingen i att en ny spelomgång förbereds eller att spelet avslutas.

Då grundprincipen för metoderna message(), checkAnswer och confirmDialog redan nämnts tidigare så beskrivs dessa metoder inte ytterligare. Det kan dock nämnas att alla tar in ett heltal som kontrolleras med en if-sats och sedan körs lämplig åtgärd.

För att tydliggöra klassen GameBoard ytterligare så kan det tilläggas att GameBoard består av minst en och max tio gissningsrader, dvs. Guess-objekt. Se bilagan.

3.2.1.3 Klassen Peg

Denna klass definierar kulorna i spelet, se Bild 14. Den ärver ifrån JLabel och tack vare det så kan man i konstruktorn kalla på den ärvda metoden setIcon() för att sätta en bild på objektet när det ritas ut på skärmen. Klassens har två instansvariabler där color håller ordning på den färg som kulan skall ha och image håller ordning på den bild som kulan skall ritas ut med. När ett nytt Peg-objekt skapas så kallar det först på metoden setIcon() som utifrån variabeln colors värde bestämmer vilken bild som skall representera kulan på skärmen. Därefter så kallas metoden addMouseListener() som gör så att varje kula blir klickbar.

I klassen Peg deklareras sedan ytterligare några metoder som går att använda på ett Peg-objekt. Metoden setIcon() används inte bara av konstruktorn utan även av klassen Solution (se 3.2.1.5) för att kunna bestämma vilken bild som respektive kula i lösingen skall få sig tilldelad. Detta har framförallt använts då man kan skriva ut lösningen på skärmen vid implementation av programmet för att kunna felsöka.

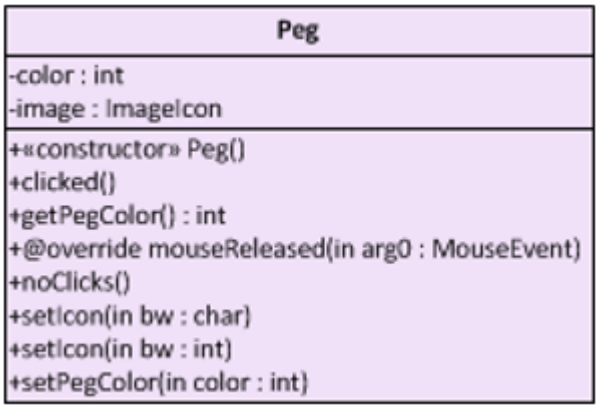

Bild 14 Klassen Peg

Metoden setIcon() finns även som en överlagrad version. Den första varianten tar in ett heltal som argument medan den andra varianten tar in en char. Den senare varianten används av FeedBack-objektet (se 3.2.1.7) för att sätta den bild som istället ska visas för en feedbackkula.

Metoden setPegColor() används även den av Solution för att bestämma värdet på instansvariabeln color och därmed tala om vilken färg som kulan i lösningen har. Den tar in ett heltal som argument och initierar instansvariabeln color med detta värde.

Metoden getPegColor() används av FeedBack för att plocka fram färgen på respektive kula i gissningsraden och lösningen så att dessa går att jämföra mot varandra. Den returnerar därmed heltalet som variabeln color innehåller.

Den överskuggade metoden mouseReleased() som finns tillgänglig i klassen tack vare att Peg implementerar gränssnittet MouseListener, talar om att kulan skall anses som klickad när musen har klickat på den och släppt.

Till sist har vi metoderna noClicks() som används för att göra kulorna i en gammal gissningsrad samt kulorna i feedbacken till icke klickbara och därmed icke förändringsbara, och clicked() som talar om att om användaren klickar på en kula så skall kulans värde, dvs. instansvariabeln color öka med 1 och bilden som representerar kulan växla till en ny färg.

Som synes på bilagan krävs det fyra kulor för att skapa en rad.

3.2.1.4 Klassen Row

Klassen Row (se Bild 15) definierar en rad i spelet. Den ärver, precis som GameBoard, från klassen JPanel. När ett nytt Row-objekt skapas så körs klassens konstruktor som i sin tur skapar en rad där objekt kan fyllas på från höger. Därefter initierar den instansvariabeln pegArray till en ny arraylist och fyller den med fyra nya Peg-objekt.

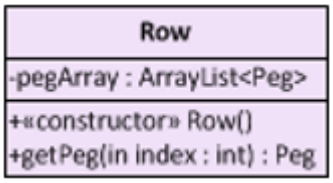

Bild 15 Klassen Row

Metoden getPeg() används av klasserna Solution, Guess och FeedBack för att plocka fram respektive kula i arrayen och sedan modifiera eller använda dess data.

Som synes på bilagan så ärver Row ifrån JPanel som nämnts och är superklass åt klasserna Solution, Guess och FeedBack.

3.2.1.5 Klassen Solution

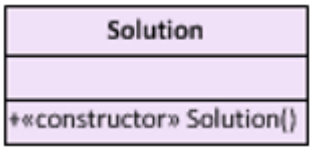

Bild 16 Klassen Solution

Klassen Solution (Bild 16) definierar en slumpad kod i spelet. Som redan nämnts i 3.2.1.4 så ärver klassen Solution ifrån Row, vilket innebär att den naturligtvis ärver alla Rows egenskaper och därmed består av fyra Peg-objekt. Det som däremot tillkommer i konstruktorn för Solution är att ett Random-objekt skapas och får en lokal variabel som referens. Därefter används en for-loop för att iterera igenom alla objekt i pegArray och ge dem var och en, ett nytt slumpat värde som skapats med Random-objektet. Även dess ikon byts och dessutom görs de till icke klickbara.

Som synes på bilagan så ärver alltså Solution av Row och både GameBoard- och FeedBack-objekten har tillgång till den.

3.2.1.6 Klassen Guess

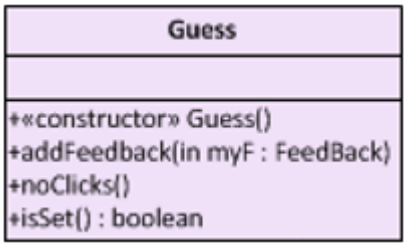

Bild 17 Klassen Guess

Klassen Guess (se Bild 17) definierar en gissningsrad i spelet. Även klassen Guess ärver från Row (se 3.2.1.4). Utöver egenskaperna som deklarerats i Row så har den ytterligare tre metoder.

Metoden addFeedback() tar in ett FeedBack-objekt som argument och adderar det sist i raden, alltså på femte plats.

Metoden noClicks() itererar igenom alla Peg-objekt i pegArray och sätter var och ett till icke klickbart.

Den sista metoden isSet() används för att iterera igenom alla Peg-objekt i pegArray och om vardera objekt har ett värde större än noll så ökar den lokala variabeln set för var och ett med 1. Metoden returnerar sedan true om variabeln set har fått värdet 4, vilket innebär att alla kulor har fått en färg.

Som synes på bilagan så ärver Guess ifrån Row och känner till klassen FeedBack. Dessutom krävs minst ett Guess-objekt för att skapa en GameBoard.

3.2.1.7 Klassen FeedBack

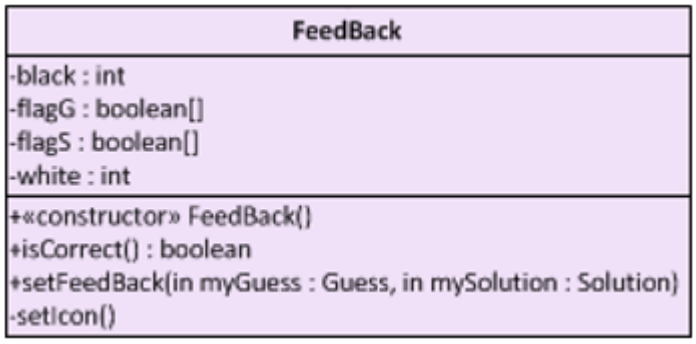

Bild 18 Klassen FeedBack

Den sista klassen i programmet är klassen FeedBack (se Bild 18) som definierar den feedback som returneras när man skickar in en gissning i spelet. Även FeedBack ärver av klassen Row (se 3.2.1.4). Utöver dessa egenskaper har den till att börja med fyra stycken egna instansvariabler. Variablerna black respektive white lagrar antalet svarta och vita feedbackkulor som skall ritas ut. Boolean-arrayerna lagrar istället vilka kulor i gissningsraden (flagG) respektive lösningsraden (flagS) som redan parats ihop med en annan kula.

När ett nytt FeedBack-objekt skapas så körs först allt som deklarerats i Row's konstruktor och därefter byts layouten på kulorna till en tabellayout med två rader och två kolumner. Till sist så kallas instansmetoden setIcon().

Metoden setIcon() kontrollerar först om variabeln black's värde är större än 0. Om så är fallet itererar den igenom pegArray så många gånger som värdet på black och byter bild på kulan till en svart feedback-kula. Därefter görs samma kontroll och process för de vita feedbackkulorna. Till sist placeras tomma bilder ut på de kulor som blir över när alla svarta och vita fått sin bild.

Metoden isCorrect() används av GameBoard för att ta reda på om alla kulor har fått en svart markering. Har de det så returneras true, vilket alltså innebär att koden har lösts.

Den sista metoden setFeedBack() är den metod som orsakat mest huvudbry under implementationen av programmet. Den tar in en Solution (se 3.2.1.5) och en Guess (se 3.2.1.6) och jämför sedan dessa för att se om användaren löst koden. Den börjar med att iterera igenom alla indexnummer i tur och ordning och jämför alltså objekten på samma indexnummer både i Solution och i Guess för att se om rätt färg hamnat på rätt plats. Om det finns en match på ett indexnummer så markeras samma indexnummer både i flagG och i flagS för att hålla reda på att dessa kulor redan har matchats och variabeln black ökar med 1. Därefter itereras alla objekt i Solution som inte har en true-markering på samma indexnummer i flagS, och var och ett jämförs med alla objekt i Guess som inte har en true-markering på samma indexnummer i flagG. Dock jämförs inte samma indexnummer på Solution och Guess nu eftersom det redan gjorts. Om några matchningar hittas så markeras detta precis som tidigare i flagS och flagG. När hela processen gåtts igenom kallas metoden setIcon() som byter ut bilderna på feedback-kulorna.

Som synes i bilagan så ärver alltså FeedBack ifrån Row och känner till både klassen Guess och Solution.

3.2.2 De fyra kraven

I uppgiftsbeskrivningen fanns fyra krav specificerade. Spelet skulle använda sig av arv, polymorfism, inkapsling och gränssnitt. Nedan listas bevis för att var och ett har uppfyllts:

- Arvs har använts i och med att Solution, Guess och FeedBack alla ärver ifrån Row.
- Polymorfism finns bland annat i och med att metoden setIcon i klassen Peg överlagras.
- Inkapsling har använts i och med att alla instansvariabler och även vissa instansmetoder är deklarerade med nyckelordet private.
- Gränssnitt har använts i och med att Peg implementerar MouseListener och GameBoard implementerar ActionListener.

4 Körexempel och analys

I detta kapitel kommer en spelomgång köras igenom och analyseras.

4.1 Genomkörning av spel

Spelet börjar med att ett välkomstfönster visas för användaren där spelreglerna presenteras och funktionen i spelet beskrivs (se Bild 19).

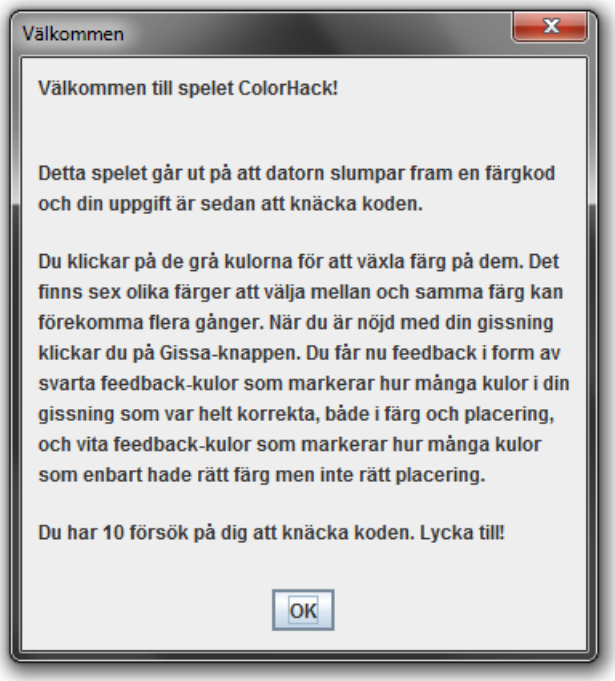

Bild 19 Start av spelet

När användaren trycker på OK-knappen så försvinner dialogfönstret och spelplanen med en förberedd gissningsrad dyker upp (se Bild 20 på nästa sida). Kulorna är grå eftersom de inte har blivit tilldelade en färg än. När användaren klickar på kulorna så växlar de mellan sex olika färger och börjar så småningom om på första färgen igen. Tanken var från början att man skulle ha kunnat dra alla färger ifrån en rad med kulor, men denna variant fungerar alldeles utmärkt och är betydligt enklare att implementera. Något som dock kunde ha gjorts bättre hade varit att låta användaren på något sätt se vilka färger som fanns tillgängliga, exempelvis genom att ha en rad längst ner i fönstret med kulor som inte var klickbara utan enbart fanns där för att användaren skulle se vilka som fanns att välja på och i vilken ordning som de skulle växla.

När användaren har klickat fram sin första gissning trycker hon på Gissa-knappen. I detta fall så talar spelet om att tre kulor har rätt färg men befinner sig på fel plats (se Bild 21 på nästa sida). Den informationen kan tydas utifrån att tre vita små feedbackkulor som har dykt upp till höger om gissningsraden. Användaren kan nu inte längre klicka på den gamla gissningsradens kulor och byta färg, men däremot har en ny gissningsrad dykt upp på skärmen som nu går att byta färg på.

På Bild 22 och Bild 23 på nästa sida så kan man se att spelet fortskrider och en av kulorna har nu hamnat med rätt färg på rätt plats, vilket markeras av den svarta feedback-kulan.

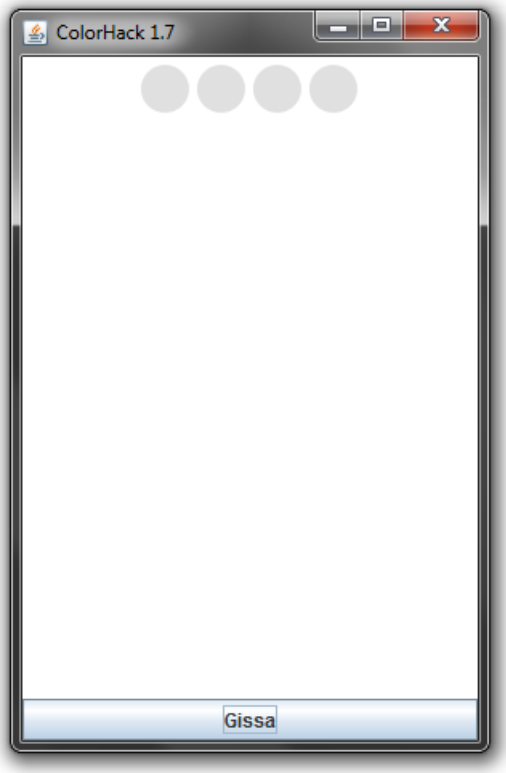

Bild 20 Spel del 1

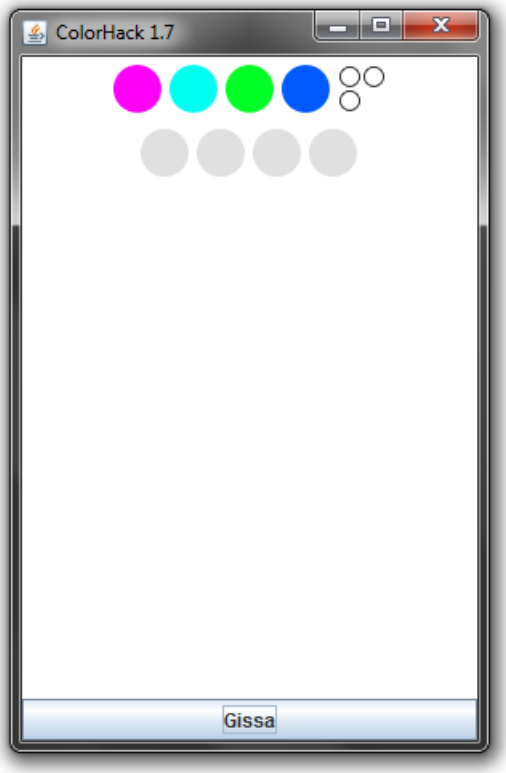

Bild 21 Spel del 2

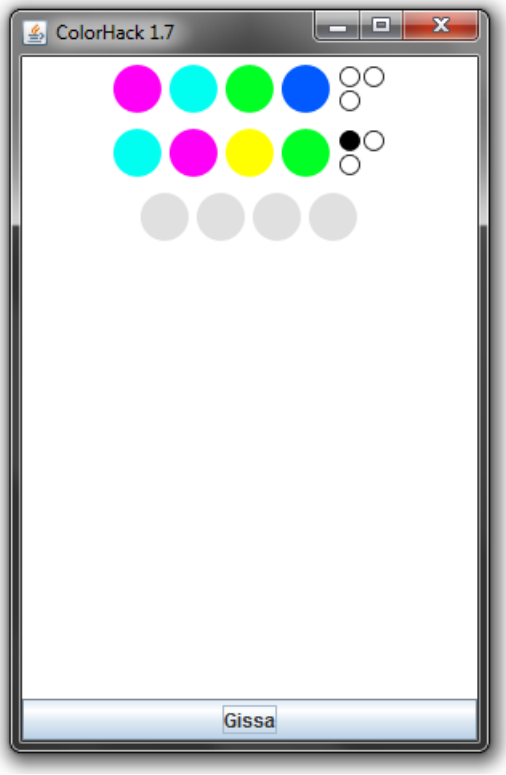

Bild 22 Spel del 3

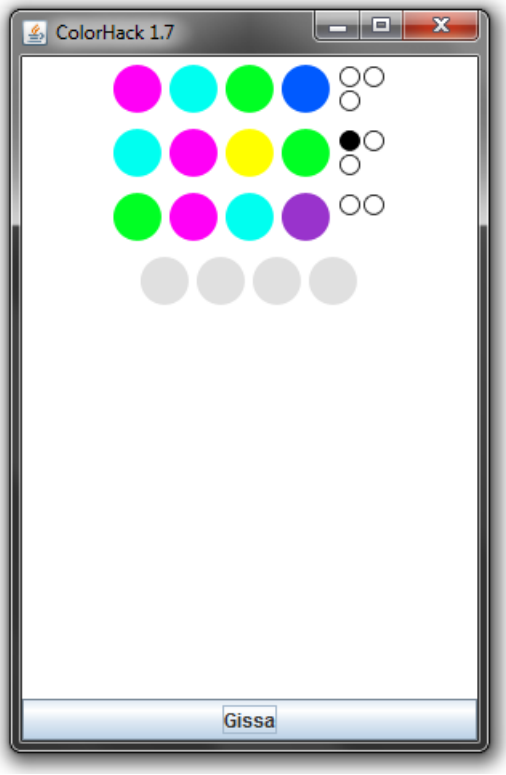

Bild 23 Spel del 4

Så småningom så löser användaren koden och man kan se hela processen på Bild 24. Ett vinstmeddelande skrivs nu ut på skärmen (se Bild 25) och användaren frågas om denne vill fortsätta spela. Användaren klickar i detta fall på nej och programmet avslutas därmed.

Som avslutning på detta kapitel kan sägas att programmet blev inte riktigt som tanken var från första början men det är fullt fungerande. Som tidigare nämnts så kunde en annorlunda design ha använts på spelplanen där valbara kulor fanns tillgängliga men i övrigt så finns inte mycket att tillägga. Dock borde något ha gjorts åt det faktum att gissningsraden verkar ”röra” på sig när feedbacken läggs till. Detta gjordes dock inte då tiden inte räckte till för detta.

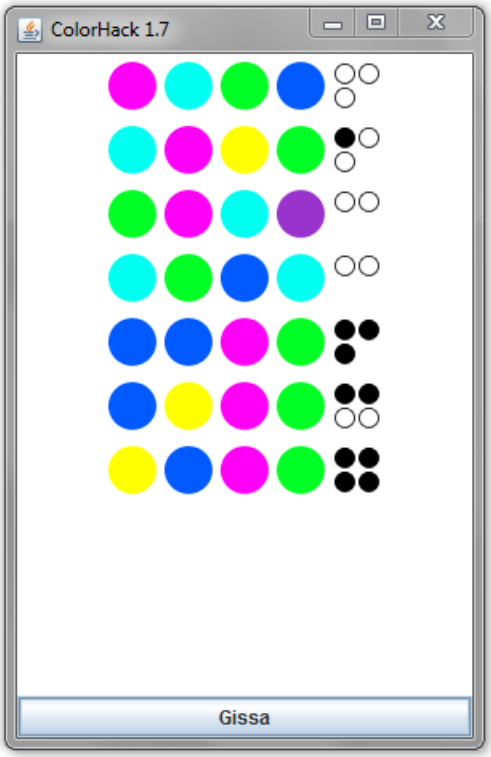

Bild 24 Spelet slut

Bild 25 Vinstmeddelande

5 Slutsats

Denna uppgift har varit riktigt besvärlig då jag först inte begrepp mig på objektorienteringen men så småningom tack vare en sommarkurs på distans så föll bitarna på plats och efter en hel veckas ytterligare arbete från morgon till kväll så kunde äntligen programmet anses vara färdigt. Jag är mycket nöjd med min insats och att jag äntligen förhoppningsvis skall kunna lägga denna uppgift bakom mig.

Bilaga A. UML-schema över ColorHack