Verónica Gaspes
Akademin för informationsteknologi
13 oktober 2015

HÖGSKOLAN I HALMSTAD

Mastermind

Projekt i programmering - höst termin 2015

Inledning

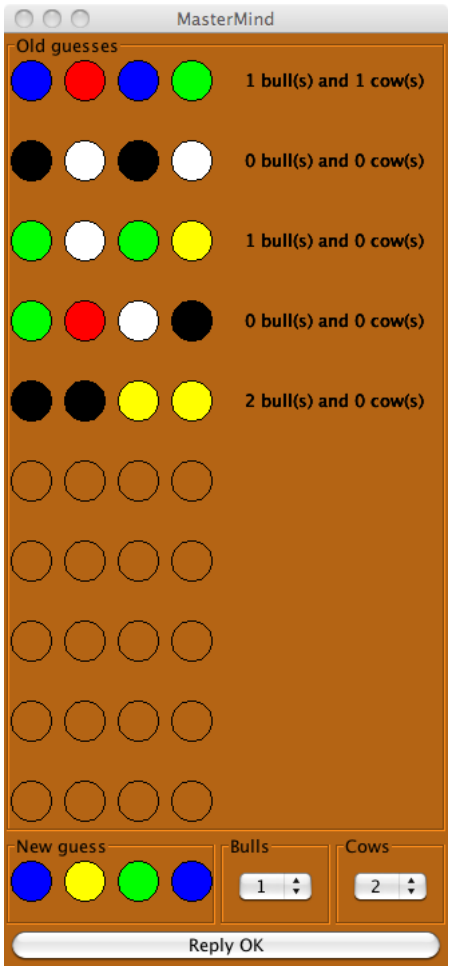
I detta projekt ska ni komplettera och förbättra ett Java-program som kan spela **Mastermind**.

Mastermind är ett spel mellan två spelare: den ene väljer en hemlig *kod*, som den andre ska gissa. I vår version av spelet är det **användaren som väljer kod och programmet som gissar**. En kod består av fyra färger, valda bland de sex färgerna svart, vitt, blått, rött, grönt och gult. En möjlig kod är blå, blå, gul, gul". Man kan alltså välja samma färg mer än en gång. Ordningen har betydelse; koden blå, gul, blå, gul"är inte densamma.

Spelet börjar med att användaren väljer sin kod. Därefter försöker programmet upprepade gånger gissa koden. För varje gissning måste användaren tala om hur bra gissningen är genom att tala om hur många *tjurar* (eng. *bulls*) och *kor* (eng. *cows*) gissningen innehåller. En tjur är ett färgval i gissningen som återfinns på exakt denna plats i den hemliga koden. En ko är en färg som finns i hemligheten, men på annan plats. En färg i hemligheten kan bara räknas som tjur eller ko en gång. Följande tabell ger några exempel på gissningar och användarens svar **om hemligheten är blå, blå, gul, gul"**:

Gissning	Tjurar	Kor
blå,gul,vit,röd	1	1
blå,blå,blå,blå	2	0
gul,gul,blå,blå	0	4
gul,blå,gul,blå	2	2

Ni bör nu ladda ner filen `mastermind.zip` samt kompilera och köra programmet. Först öppnas en dialogruta där ni uppmanas att välja en kod; därefter öppnas huvudgränssnittet, där programmet presenterar sina gissningar. Nedanstående bild ger ett exempel, där hemligheten är just blå, blå, gul, gulöch där programmet just presenterat sin sjätte gissning. Användaren ska välja rätt antal tjurar och kor med hjälp av valknapparna nere till höger och därefter trycka **Reply OK**.



Tyvärr är programmet inte särskilt smart; varje gång väljer det en helt slumpmässig gissning, med den enda inskränkningen att samma gissning inte görs igen. Programmet tar alltså inget intryck av användarens svar på de tidigare gissningarna. Resultatet är förstås att programmet nästan aldrig lyckas gissa hemligheten innan spelplanen fyllts.

Er uppgift är att förbättra programmets förmåga att gissa.

Interfacet `GuessEngine`

Er uppgift är närmare bestämt att ersätta klassen `NaiveEngine` med en mer sofistikerad klass. En titt på klassen `NaiveEngine` visar att den implementerar interfacet `GuessEngine`; ni måste alltså förstå detta interface. För detta behöver vi först förstå klasserna `Color`, `Code` och `Reply`.

- `Color` är en uppräkningstyp med de aktuella färgerna:

```
public enum Color {BLACK, WHITE, BLUE,
                   RED, YELLOW, GREEN}
```

- Objekten av klassen `Code` representerar möjliga gissningar, som ett fält av färger med fyra element:

```
public class Code {
    private Color[] cols = new Color[4];

    public Code(Color[] cols) {
        for(int i=0; i &lt; cols.length; i++)
            this.cols[i] = cols[i];
    }

    public Color getColor(int i) {return cols[i];}

    public String toString() {
        return cols[0] + " " + cols[1] + " "
                 + cols[2] + " " + cols[3];
    }
}
```

- Objekten av klassen `Reply` representerar möjliga svar från användaren på en gissning från programmet:

```
public class Reply {
    private int bulls, cows;

    public Reply(int bulls, int cows) {
        this.bulls = bulls;
        this.cows = cows;
    }

    public int getBulls() {return bulls;}

    public int getCows() {return cows;}

    public boolean equals(Reply other) {
       return bulls == other.bulls && cows == other.cows;
    }

    public String toString() {
        return bulls + " " + cows;
    }

}
```

Med hjälp av dessa typer kan vi nu gå igenom metoderna i interfacet `GuessEngine`:

- `void init();`
  Initiering av modellklassen. Denna metod måste anropas **först** i en spelomgång. Anledningen till att initieringen inte kan ske i konstrueraren är att en exekvering av programmet kan bestå av flera spelomgångar (en spelomgång är ett val av hemlig kod följt av ett antal gissningar från programmet).

- `Code getNewGuess();`
  Att anropa denna metod begär att få en gissning från modellklassen.

- `void answerNewGuess(Reply reply) throws ContradictionException;`
  Ett anrop till denna metod informerar modellklassen om vilket användarens svar (dvs antalet tjurar och kor) på den senaste gissningen var. Ett möjligt resultat är att modellklassen aktiverar undantaget
  `ContradictionException`
  detta får bara ske när modellklassen anser att användaren gett motstridiga svar på de gissningar som gjorts.

- `String explainContradiction(Code secret);`
  Om modellklassen aktiverat ett undantag från föregående metod kan gränssnittet här begära en motivering; som argument måste då den riktiga hemligheten ges.

- `public boolean moreGuessesAllowed();`
  Programmet får inte göra fler än tio gissningar; denna metod returnerar `true` om detta ännu inte skett och `false` annars.

- `public int oldGuesses();`
  Denna metod och de två återstående är till för att gränssnittet ska kunna rita upp de gjorda gissningarna; denna metod returnerar antalet besvarade gissningar.

- `public Code getOldGuess(int i);`
  `public Reply getOldReply(int i);`
  Dessa två metoder returnerar gissning nummer `i` och svaret från användaren på denna gissning.

Klassen `NaiveEngine`

Vi kan nu se hur klassen `NaiveEngine` implementerats. Dess instanssvariabler är följande:

```
private List<Code> codes;
private Code newGuess;
private Code[] oldGuesses;
private Reply[] oldReplies;
int guessCount;
```

De tre sista av dessa variabler håller reda på gamla gissningar och användarens svar på dessa. De två fälten skapas med 10 element vardera i konstrueraren. Variabeln `newGuess` håller reda på den sist gjorda gissningen, som ännu ej fått något svar, och listan `codes` innehåller alla ännu ej gjorda gissningar. Denna lista initieras i `init` till att innehålla alla de $6^4 = 1296$ möjliga koderna. En ny gissning väljs som ett slumpmässigt element i denna lista, som samtidigt tas bort ur listan.

`NaiveEngine` har inte förmåga att upptäcka om användaren ger motstridiga svar; undantaget `ContradictionException` aktiveras först när listan av ännu ej gjorda gissningar blir tom. I konsekvens med detta kan klassen inte ge någon rimlig förklaring till hur användaren svarat fel.

Er uppgift

Ni ska implementera en annan klass som kan ersätta `NaiveEngine`; er klass måste alltså implementera `GuessEngine`. Men den måste göra betydligt bättre gissningar: *varje* gissning programmet gör ska ha egenskapen att den inte kan uteslutas på grund av användarens hittillsvarande svar. I princip är det lätt att beskriva hur man gör detta: listan `codes` ska inför varje gissning bara innehålla de koder som *kan* vara hemligheten, dvs som ger precis de svar man faktiskt fått av användaren för alla gjorda gissningar.

Ni bör börja med att noga läsa igenom `NaiveEngine` och avgöra vilka delar ni kan behålla oförändrade och vilka ni måste förbättra. En hel del går att återanvända! Observera också att ingen av de övriga klasser ni får i zip-filen ska ändras på något sätt.

Den huvudsakliga förbättringen är alltså att reducera `codes` efter hand så att den bara innehåller möjliga hemligheter. Inför den första gissningen består `codes` av alla tänkbara koder; man har ännu inte fått några svar. För varje anrop av `answerNewGuess` får modellklassen veta hur rätt/fel den senaste gissningen var. Man kan då gå igenom listan `codes` och ta bort alla element som inte skulle gett just detta svar på den senaste gissningen.

För att implementera denna idé måste man, för varje element `code` i listan `codes` avgöra om den skulle leda till svaret `reply` på gissningen `newGuess`. Detta i sin tur kräver att programmet själv kan räkna ut vilket svar `newGuess` ger om hemligheten är `code`. Kan vi göra det räcker det sedan att jämföra med användarens givna `reply` för att avgöra om `code` ska få vara kvar i listan eller inte.

Att beräkna svaret för given hemlighet och gissning Detta kan göras på många sätt; vi beskriver ett. Låt hemligheten vara `secret` och gissningen `guess`.

- Antalet tjurar är lätt att beräkna genom att i en loop gå igenom `secret` och `guess` parallellt och räkna antalet gånger färgerna är lika.
- Vi kan beräkna *summan av* antalet tjurar och kor på följande sätt:

  För varje färg `col` beräknas hur många gånger denna färg förekommer, både i `secret` och i `guess`. Det minsta av dessa två antal är summan av antalet tjurar och kor av färgen `col`. Summering över alla färger ger totala antalet tjurar och kor.
- Har vi gjort de två föregående beräkningarna kan vi beräkna antalet kor med en subtraktion.

Vi rekommenderar er att definiera en metod som tar `guess` och `secret` som parametrar och returnerar ett `Reply` enligt detta. Vi rekommenderar också starkt att ni skriver ett litet testprogram som testar denna funktion så att ni är säkra på att den fungerar som den ska. Därefter använder ni den i filtrering av `codes` i `answerNewGuess`.

När ni kommit så långt kan ni ersätta `NaiveEngine` med er modellklass och testa programmet. Om användaren ger korrekta svar kommer programmet normalt att finna hemligheten inom högst sex gissningar. Om användaren däremot svarar fel för någon gissning kommer ni så småningom till en situation där `codes` är tom, dvs det finns ingen ny gissning att göra. Ni kan då inte göra

annat än att kasta ett undantag. För att bli godkänd på projektet behöver ni **inte** förbättra `explainContradiction` från `NaiveEngine`, dvs ni kan inte identifiera användarens misstag.

Frivillig uppgift

Som frivillig uppgift, som bokförs och kan hjälpa till i gränsfall till överbetyg i kursen, föreslår vi att ni implementerar en förbättrad `explainContradiction`. I denna metod vet ni hemligheten och ni vet alla era tidigare gissningar och vilka svar användaren gett. Med hjälp av detta kan man identifiera ett felaktigt svar; ett sådant måste finnas om `codes` är tom utan att hemligheten gissats. Metoden ska ge sitt resultat i form av en sträng. Exempel på ett tänkbart svar skulle kunna vara

```
When I guessed Red White Blue Red, you answered 1 0.
Correct answer should have been 0 1.
```

Det är förstås möjligt att användaren gett *flera* felaktiga svar; det räcker i så fall att identifiera och förklara ett av dem.