09-Nov-2012/FK

ID1004 Inlämningsuppgift 4.5hp

Inlämningsuppgiften görs, redovisas och examineras individuellt.

Texten nedan innehåller fyra olika uppgifter. **Välj och lös en av uppgifterna.** Uppgiftstexterna och tillhörande material kan komma att uppdateras under arbetet i syfte att rätta fel.

Den valda uppgiftens redovisning (källkod m.m.) ska vara insänd till Bilda (bilda.kth.se) senast 7 januari 2013 (i undantagsfall skicka in med epost till fki@kth.se) Redovisa gärna tidigare.

Bedömning och betygsättning

Uppgiften ger 4.5 hp. Betyget är P (pass) eller F (fail).

För betyg P gäller att:

- Lösningen skall uppfylla kraven för den valda uppgiften (se specifik uppgift).
- Källkoden skall vara kommenterad med syftet att en person som inte känner till den lättare ska kunna sätta sig in i den.
- Alternativ, död, eller experimentell kod som inte ingår i den inlämnade lösningen skall vara bortplockad (gäller dock inte test- och debugkod, t ex inaktiverade spårutskrifter).
- Det ska finnas ett testprogram som testar minst en och gärna flera kritiska funktioner.
- Det skall finnas en separat fil med en instruktion för hur programmet startas och körs, samt en lista över alla filer som ingår i lösningen.

Betyget F skall ges:

- Om något av kraven för P inte är uppfyllt.
- Om det finns anledning att tro att studenten inte har presterat större delen av lösningen själv.

Rest får ges:

- Om lösningen inte uppfyller kraven för P, men ändå bedöms kunna göra det med en insats om högst tre arbetsdagar.

En rest skall vara åtgärdad inom en vecka (eller avtalad tid), därefter sätts betyget F.

Om samarbete kring inlämningsuppgiften

Eftersom uppgiften examineras individuellt är det av största vikt att det tydligt framgår av lösningen att det är studentens egna och individuella arbete. Om två eller flera lösningar är så lika varandra att de bedöms vara identiska, så går det inte att avgöra vem som prestationen tillhör. Därmed kan ingen av lösningarna godkännas.

Samtidigt är det givetvis av godo att kunna diskutera strategier, tillvägagångssätt och modeller med andra, och detta är därför inte kontroversiellt. Det är också så att vissa typer av problem bara har ett fåtal lösningar och detta ökar givetvis likheten dem sinsemellan. Det är dock osannolikt att två

personer oberoende av varandra producerar i det närmaste identisk kod, även om de arbetar på samma problem.

Råd inför uppgiften

- Studera varje uppgift noga och försök se vilken som passar dig bäst. Uppgifterna är olika svåra, men för att kunna bedöma svårighetsgraden gäller det att först analysera problemet lite grand.
- Skriv en objektorienterad lösning! Programmet ska fungera, javisst, men all kod får inte ligga i statiska metoder.
- Skissa vilka funktioner som behövs, och titta på hur de hänger samman. Finns det komponenter där du kan tänka dig att någon ger dig ett objekt som bidrar med viktig funktionalitet med några få, enkla metodanrop? I så fall är det stor chans att dessa komponenter faktiskt hör hemma i en egen klass.
- Finns det komponenter som delar viktiga egenskaper? I så fall kanske de ska ärva från en gemensam superklass.

Resurser online

Java version 6 API dokumentation
http://download.oracle.com/javase/6/docs/api/

Java JDK download (SE version 6)
http://www.oracle.com/technetwork/java/javase/downloads/index.html

TextPad editor for Windows
http://www.textpad.com/

Eclipse IDE (välj Classic)
http://www.eclipse.org/downloads/

Uppgift INL1.1 Pokerhänder

Modellera och implementera på ett objektorienterat sätt följande:

En vanlig kortlek innehåller 52 kort (inga jokrar). I kortspelet poker erhåller varje spelare en giv om fem kort. Given, eller handen som den också kallas, värderas enligt regler som gör mindre sannolika kortkombinationer mer värdefulla. 2 är lägst, äss är alltid högst. Vanligtvis tillåts spelaren att kasta inget eller flera kort från sin hand och få nya kort i deras ställe. Genom att kasta listigt har spelaren chans att förbättra sin hand. Ingen spelare har information om de andra spelarnas kort. Därefter vidtar vanligtvis ett psykosocialt rackarspel om pengar eller pengars värde, men den delen ingår inte i uppgiften.

Programmet skall kunna hantera minst en och max fem händer (valbart av den enda användaren). Kortleken består av 52 kort och blandas väl. Varje hand får fem kort. Handens styrka klassificeras och visas för användaren tillsammans med korten. Eventuella kastade kort läggs undan och varje hand som kastat kort får ersättningskort från kortleken. Händerna klassificeras på nytt och visas, varefter programmet ska erbjuda att avsluta eller göra en ny giv (alla kort samlas in och cykeln upprepas).

Pokerhänder (i stigande värde):

1. Högt kort (den högsta valören i handen)
2. Ett par (två kort med samma valör)
3. Två par
4. Tretal (tre kort med samma valör)
5. Stege (fem kort med valörerna i strikt ordningsföljd)
6. Färg (alla kort i samma färg)
7. Kåk (ett par och ett tretal)
8. Fyrtal (fyra kort med samma valör)
9. Straight flush (stege och färg)

Logiken för att välja vilka kort som ska kastas kan göras mer eller mindre avancerad, och den enklaste grundregeln är att inte förstöra den hand man redan har (om man har ett par eller mer). I praktiken sitter man nöjd med en stege och uppåt (lyckas man få fyrtal i given bör man byta det enstaka kortet för att inte se alltför nöjd ut, men det är en annan historia). För lägre händer kan följande enkla heuristik användas:

- Högt kort: kasta de två eller tre lägsta korten (men se nedan)
- Ett par: kasta de tre kort som inte ingår i paret (men se nedan)
- Två par: kasta det ensamma kortet i hopp om en kåk
- Tretal: kasta de två korten i hopp om fyrtal

Har man har högt kort eller ett lågt par kan det vara värt att se om man är ett kort från en färg (9/47 = 0.19).

Den svåraste analysen är en hålstege, dvs att bestämma vilka kort som ska kastas i hopp om att få en stege. En hålstege är en hand klassad som högt kort som nästan är en stege. I det enklaste fallet kan man tänka sig handen 3, 4, 6, 7, 8. Där skulle man kunna kasta 3:an i hopp om en 5:a (4, 5, 6, 7, 8). I de allra flesta fall vill man kasta det lägsta kortet, men ibland inte. Den hand som har störst chans att

bli förbättrad har fyra kort i en sekvens som är öppen i båda ändar, t ex 5, 6, 7, 8, K. Kungen kastas i hopp om att man får en 4:a eller 9:a istället.

Ett sätt att göra analysen är att från handens valörer skapa en representation av sekvenser och hål. Varje sekvens och hål har en viss längd. Eftersom det bara finns 13 valörer och man har fem på handen med högt kort, så blir den längsta sekvensen 5 kort och det längsta hålet 8. Här är några exempel (’s’ står för sekvens, ’h’ för hål och symbolen följs av längden):

3,4,6,7,8	h1 s2 h1 s3 h6
5,6,7,8,K	h3 s4 h4 s1 h1
2,10,Kn,D,K	s1 h7 s4 h1
9,10,D,K,A	h7 s2 h1 s3
7,8,9,Kn,D	h5 s3 h1 s2 h2
2,4,6,8,9	s1 h1 s1 h1 s1 h1 s2 h5

Det visar sig att man kan söka efter mönster i resultatet av sh-analysen och med hjälp av dem skapa regler för vilket kort som skall kastas. T ex:

OM sh-analysen innehåller ’s2 h1 s3’ SÅ kasta kortet med lägst valör.

OM sh-analysen innehåller ’s3 h1 s2’ SÅ kasta kortet med lägst valör.

OM sh-analysen innehåller ’s1 h* s4’ SÅ kasta kortet med lägst valör.

OM sh-analysen innehåller ’s4 h* s1’ SÅ kasta kortet med högst valör.

Uttrycket ’h*’ ovan betyder att längden på hålet är ett eller större. Det kan finnas fler meningsfulla mönster och regler än de visade här. Det finns givetvis också många mönster där chansen att förbättra handen till en stege är ganska liten, som t ex ’s1 h* s3 h* s1’. Här skulle man kunna tänka sig att kasta både högsta och lägsta kortet och kanske i alla fall få ett par:

Chans för stege: 0.0148 = ((8/47)*(4/46))

Chans för par: 0.387 = 3*((3/47)+(3/46))

I uppgiften ingår att implementera analys och regeltillämpningar med hjälp av lämpliga klasser, datastrukturer och metoder. Det är inget krav att göra en grafisk lösning, allt kan skrivas ut på System.out. Det ska också påpekas att detta inte är ett komplett spel, utan endast stödfunktioner för en spelare hur en hand kan förbättras.

Programmet ska fråga efter antal händer, dela ut kort, klassificera och visa alla händer efter given, följa sina egna bytesrekommendationer i varje hand och till sist klassificera och visa alla händer igen. Händer som har förbättrats ska signaleras på tydligt sätt.

Uppgift INL1.2 Enarmad bandit

Denna uppgift består i att skriva ett objektorienterat program som återger beteendet hos en spelautomat, specifikt en sådan som brukar kallas för ”enarmad bandit”. Den typ av spelautomat som är aktuell för uppgiften har tre symbolhjul och en enda betallinje.

Varje hjul har tio olika symboler, och alla tre hjul är identiska med varandra. Maskinen har inga hjulstopp, ingen jackpot-mode men det finns ett (simulerat) myntinkast, och varje mynt ger kredit för 10 spel. Eventuell vinst adderas till krediten.

Vinsttabell

Maskinen ska ha en vinsttabell som i långa loppet ger minst 80 och som mest 99 % vinst. Om vi representerar symbolerna på hjulen med siffror så kan man till exempel utgå från denna tabell:

Första hjulet	Andra hjulet	Tredje hjulet	Vinst
9	9	9	180
8	8	8	160
7	7	7	140
6	6	6	60
5	5	5	25
4	4	4	25
3	3	3	25
2	2	2	25
1	1	1	25
0	0	0	5
Lika	Lika	5	10
Lika	Lika	2	5
		2	2

Det är givetvis tillåtet att experimentera med olika slags vinsttabeller, så länge den förväntade utbetalningen håller sig inom gränserna ovan.

Programmet skall skriva ut den förväntade vinsten på System.out när det startar. Det kan därför vara bra att ha vinsttabellen implementerad som en metod som granskar en viss symbolkombination och returnerar hur mycket den vinner. Eftersom maskinen bara har 1000 olika kombinationer så är det enkelt att gå igenom alla kombinationerna, summera vinsten och dividera med tio. Tabellen ovan ger tillbaka 98.5% till spelarna.

Maskinens beteende

Spelarens 'kommandon' inskränker sig till två handlingar: lägga i pengar eller att spela genom att snurra på hjulen. När man lägger i en slant får man 10 krediter. Varje spel drar av 1 kredit, så det måste finnas minst 1 kredit i maskinen för att det ska gå att spela. Det ska givetvis gå att se hur många krediter man har kvar. Om man vinner så adderas vinsten till krediterna.

För att göra spelet mer spännande och hålla kvar spelaren ska maskinen använda följande interna procedur för varje spel (detta är ganska nära hur det fungerar i verkligheten):

- Dra av en kredit.
- De tre hjulens nästa kombination slumpas fram med hjälp av en slumptalsgenerator.
- Kombinationen jämförs mot vinsttabellen.
- Om det är en förlorande kombination så kontrolleras om det är en *nästan* vinnande kombination, t ex: 9, 4, 9. Om så är fallet så ändras kombinationen i 50-75% av fallen till att visa en förlorande kombination som mer ser ut som nästan vinst, t ex 9, 8, 9. Tanken är att spelaren ska tycka att det var nära, bäst att spela igen.
- Hjulen snurras och stannas med den kombination som bestämts på betallinjen. Alla hjulen snurras i ca 1000 ms, därefter stannas det första hjulet. Efter ytterligare ca 300 ms stannas andra hjulet och efter ytterligare 300 ms det tredje hjulet.
- Om det var vinst så signaleras detta väl synligt.
- Vinsten räknar upp krediten med ca 50 ms mellan varje kredit.

Simulatorn SlotFrame

Det går att göra en ganska trevlig simulering med bara textgränssnitt, men det går inte att komma ifrån att det är lite torrt och tråkigt. Därför finns det en färdig grafisk simulator som man gärna får använda. Simulatorn heter SlotFrame och den implementerar ett interface som heter SlotControl. Genom att skapa en SlotFrame i sitt program visas simulatorn och sedan kan man använda den för att anropa metoderna definierade i SlotControl:

```
public int [][] getWheelModel()
public String getNextEvent()
public void setCredit(int n)
public void win(int amount)
public void roll(int [] ar)
```

Metoden getWheelModel returnerar en matris som beskriver hur många hjul simulatorn har (3) och hur symbolerna är arrangerade på varje hjul (10 olika symboler och hjulen är identiska).

Metoden getNextEvent() returnerar en sträng som beskriver vad användaren/spelaren har gjort: "creditbutton" betyder klick på knappen "Insert coin" och "rollbutton" betyder att spelaren tryckte på knappen "ROLL". Inga andra händelser rapporteras, och metoden återvänder inte från anropet förrän någon av dessa händelser har inträffat.

Metoden setCredit(int n) används för att visa hur många krediter spelaren har.

Metoden win(int amount) används för att tända (0 < amount) eller släcka (0 == amount) WIN-skylten på simulatorn.

Metoden roll(int [] ar) släcker WIN-skylten och rullar (animerar) fram hjulen till de symboler man vill visa på betallinjen. Om man t ex skickar in arrayen {0, 1, 2} så kommer första hjulet att stanna på plommon, andra hjulet på apelsin och tredje hjulet på körsbär.

Klassen SlotFrame (som är den man instansierar i sitt program) har en konstruktor som ser ut så här:

```
public SlotFrame(String title, boolean fallBackMode)
```

Parametern title är den text man vill ha på fönstret. Parametern fallBackMode väljer mellan två olika sätt att rita ram och betallinje över hjulen. Om det ser konstigt ut med fallBackMode=false så bör det fungera på alla plattformar med true istället.

Det är självklart tillåtet att modifiera simulatorn, lägga till fler hjul, ändra symboler, införa fler betallinjer, lägga till ljudeffekter etc. så länge de ovan beskrivna kraven är uppfyllda.

INL1.3 Självlärande månlandare

Denna uppgift är en gammal ide från maskininlärning och artificiell intelligens. Tekniken fungerar bäst för okomplicerade problem med få dimensioner.

Uppgiften består i att på ett objektorienterat sätt programmera en simulator för en självlärande månlandare. Månlandaren har en enkel konstruktion och är endast utrustad med en radarhöjdmätare, en bränsletank och en raketmotor som kan vara av eller på. Raketen är tänkt att kunna släppas från olika höjd över månytan, och uppgiften som den ska lära sig med hjälp av simulatorn är hur den ska starta och stoppa raketmotorn för en framgångsrik landning (fallhastighet > -10 m/s) med bränsle kvar.

En simuleringsrunda går i huvudsak till så här:

- Initiera raketens höjd till mellan 100 – 200 meter.
- Initiera raketens bränsle till mellan 75 – 125 liter.
- Initiera fallhastigheten till mellan -5 – +5

```
while (0 < höjd && 0 < bränsle) {
  if (motorPå(höjd, bränsle, fallhastighet)) {
    int kick = Math.min(2, bränsle);
    bränsle -= kick;
    fallhastighet += kick;
  }
  höjd += fallhastighet;
  fallhastighet += -0.97; // Månens gravitation
}
if (bränsle == 0 && 0 < höjd) {
  // FAIL: slut på bränsle före landning
}
else if (fallhastighet < -10) {
  // FAIL: för hög hastighet vid landning
}
else {
  // WIN!
}
```

Det kritiska beslutsfattandet sker i anropet till metoden

```
boolean motorPå(höjd, bränsle, fallhastighet)
```

Den metoden ska nämligen returnera `true` om motorn ska vara på och `false` om motorn skall vara av. Till hjälp för sitt beslut har metoden en tredimensionell matris där varje cell innehåller två räknare; en för motor på och en för motor av. Efter att ha lokaliserat rätt cell med hjälp av höjd, bränsle och fallhastighet, så tar man det motorbeslut vars räknare är störst. Om räknarna är lika så används slumpen med lika stor chans för båda möjligheterna.

Inlärning sker på det viset att för varje simulerat landningsförsök så byggs en lista över alla besökta celler i beslutsmatrisen, tillsammans med det fattade beslutet i den cellen. När simuleringen är över

går man igenom listan och om det var en lyckad landning så ökas räknaren för det fattade beslutet i den cellen med två, och om var en misslyckad landning så minskas räknaren med ett.

Det visar sig att en matris om 8x8x8 celler är tillräcklig, under förutsättning att indexen beräknas på lämpligt sätt. Här är några förslag på olinjär kvantisering av de ingående parametrarna:

Höjd	höjdIndex	Bränsle b	bränsleIndex	Fallhastighet v	fallhIndex
< 5	0	< 10	0	< -30	0
5 – 10	1	10 – 20	1	-30 – -20	1
10 – 15	2	20 – 30	2	-20 – -10	2
15 – 20	3	30 – 40	3	-10 – -1	3
20 – 25	4	40 – 50	4	-1 – +1	4
25 – 50	5	50 – 60	5	1 – 10	5
50 – 100	6	60 – 70	6	10 – 20	6
100 <	7	70 <	7	20 <	7

Förväntat beteende

Givetvis är det så att raketen initialt inte har en aning om vad den ska göra, och kommer att krascha spektakulärt många gånger. Men i och med att den får öva på låg höjd så kommer vissa landningar att lyckas och dessa belönas då i beslutsmatrisen. Efter en hel del övning så kommer den att bli riktigt duktig på att landa.

När man kör en enskild simulering ska raketen visas relativt månytan. Om man väljer en textversion så kan man göra en enkel liggande plot genom att i varje simuleringsvarv skriva ut:

```
|                <*
```

Där tecknet ’|’ representerar månytan, ’*’ raketen och ’<’ raketmotorns flamma om motorn är på.

I och med att beslutsmatrisen representerar raketens kunskap, så ska det också finnas räknare för matrisens prestanda: antal landningsförsök, antal lyckade landningar, antal krascher pga för hög fart och antal krascher pga bränslebrist. Efter varje simulering ska dessa uppdateras och presenteras, så att man kan om antalet lyckade landningar ökar. Om man vill, så kan man låta starthöjden initieras till 100 – 400 meter när mer än hälften av alla landningar lyckas.

Ibland kan raketen uppnå 75% lyckade landningar, ibland bara ca 50%. Att det varierar beror på att beslutsmatrisen inledningsvis är instabil och riskbeteende blir inlärt. Det kan också vara så att beslutsmatrisen behöver ha högre upplösning i en eller flera dimensioner. Experimentera gärna!

Som ett helt frivillig tillägg utanför själva uppgiften: fundera ut något sätt att presentera beslutsmatrisen så att man kan förstå hur raketen resonerar.

Om någon tycker att gravitationen är för hög, så betänk att raketen kanske inte övar för våran måne.

Uppgift INL1.4 Korsordsgenerator

Skriv ett objektorienterat program som genererar en korsordsfläta ungefär så här:

		2				3		
1	R	O	S	E	T	T		
T		S		7		O		
U		T		L		M		
N	6	E	K	A	4	A	L	N
G		N		K		T		
A				A		E		
N	5	K	A	N	O	N	E	N

Tabell 1 - korsordet

	Vågrätt	Lodrätt
1	På skosnöret	… på vågen
2		Skivar man helst ny
3		Röd, rund och god till pasta
4	Gammalt längdmått	
5	Skjuter man långt med	
6	Liten träbåt	
7		Att sova mellan

Tabell 2 – gåtor

Programmet ska alltså generera tabell 1 och tabell 2 (en människa förväntas hitta på och fylla i gåtorna i tabell 2).

Till er hjälp har ni en fil som innehåller mer än 400 000 svenska ord, inklusive böjningsformer. Filen ligger på kth/social, under Material H11 och länken ligger i ordet ’ordlista’ längt ner på sidan. Här är dock själva länken:

https://www.kth.se/social/upload/4ef0a9d1f276545a12000007/ordlista.zip

Programmet får alltså lägga ut ett ord vågrätt eller horisontellt, och sedan ett till, och ett till. Ganska raskt så visar det sig att man måste börja söka efter ord som passar, och ju tätare korsordet är, desto svårare blir det att lägga till ett ord till. Till slut går det inte längre att finna en ledig placering där ett ord i ordlistan passar in, och då är man klar.

Utmatningen från programmet kan vara text eller grafik, och en textlösning är givetvis helt tillfyllest.