Laboration 4 TDA143

Syfte

Öva på klasser med grafik och lyssnare. Jobba med en separat modell och vy/kontroller-del. Arbeta inkrementellt med tydliga delmål. Göra ett roligt spel att spela med era kompisar :).

Redovisning

När hela uppgiften är klar ska ni redovisa för en kursassistent. När ni redovisat får ni en fembokstäverskod. Därefter ska ni skicka in alla .java filer ni har skrivit eller ändrat, och koden. Följ instruktionerna på inlämningssidan på kurshemsidan.

Uppgift 1: Fyra i rad

Ni ska implementera spelet fyra i rad (inte varianten med brickor som man släpper i en plastställning, utan den sortens man kan spela i ett rutigt block där spelare X och spelare O turas om att rita sin symbol i en ruta tills någon av dem får fyra i rad). En vinnande rad kan vinna horisontell, vertikal eller diagonal (snett uppåt eller neråt).

Labb4.jar innehåller en komplett körbart kopia av spelet! Om ni dubbelklickar på filen i en filhanterare (utforskaren på windows, finder i iOS) så startar det (förutsatt att Java är installerat korrekt på systemet). Tyvärr har källkoden kommit bort så ni har fått i uppdrag att skriva om den... Men ni har i alla fall en mall (en s.k. referensimplementation) att utgå från om ni är osäkra på hur något ska fungera.

Ni startar inte heller från noll vad gäller källkoden, utan ni har en ganska bra början med en Main-klass (som bara har en main-metod för att köra igång spelet, och lite testkod som kan vara intressant att kika på), en klass GameModel som innehåller *modellen* för spelet, alltså allt som inte har med grafiken att göra utan själva spelet och en klass PlayField som är spelets "vy" och har hand om all grafik och styrning. GameModel och PlayField är bara påbörjade, men det finns "stubbar" (påbörjade definitioner) för alla publika metoder. Vi återanvänder också Location-klassen från labb 2.

Notera att PlayField.java innehåller två klasser, en huvudklass (PlayField) och en klass CellLabel som representerar enskilda celler på spelplanen (ärver från JLabel - en klass för att visa text, så varje cell kan visa en bokstav som X eller O). CellLabel lyssnar dessutom efter mushändelser på sig själv.

Förutom en del kod så finns det en hel del kommentarer i klasserna. Flera av dem är endera markerade med TODO, som anger att ni ska göra något (vid varje TODO står vilken deluppgift det gäller) eller markerade med en asterisk ('*') som anger att det är ett tips om hur ni kan göra. Samtliga de kommentarerna ska ni ta bort efterhand som ni inte längre behöver dem.

I varje del av uppgiften ska ni utöka koden med ytterligare funktionalitet tills er implementation fungerar som referensimplementationen. Varje deluppgift består av tre steg: Ändra modellen, ändra vyn, och testa att den ni gjort fungerar.

Deluppgift a)

Provkör ett par omgångar med Labb4.jar. Se vad som händer om någon spelare får fyra i rad.

Lägg alla .java filer i en mapp och öppna Main.java och kompilera och kör den. Som ni ser gör klassen inte så mycket ännu, den skapar inte ens en spelplan, men om allt fungerar ska den öppna ett litet fönster utan innehåll. Studera Main.java och se hur den först skapar en modell, sedan använder modellen för att skapa en vy (så vyn därmed har tillgång till all information den kan få från modellen). Notera också förslagen på hur programmet kan testas, och särskilt metoden test(). Ni kan provköra test() redan nu, men metoden kommer krascha.

Utför sedan följande:

Modell: Studera hur klassen GameModel håller reda på:

- Aktiv spelare (vems tur det är)
- Hur stor spelplanen är (höjd och bredd)
- Vad varje ruta på spelplanen innehåller (variabeln board)

Vy: Skriv klart konstrueraren för klassen PlayField. Den behöver skapa en CellLabel för varje cell i spelplanen, och lägga till dem på spelplanen så de syns.

Testa: När du kör Main.java ska du se en spelplan med de mått du angett. Inget händer när man rör musen över cller. När man klickar på en cell skrivs ett debug-meddelande ut till System.out.

Deluppgift b)

Nu ska vi fixa modellen så den faktiskt håller reda på vad som finns i varje cell, och lägga till metodeer för att läsa av och manipulera brädet. Sist fixar vi vyn så att spelare kan klicka på celler för att "erövra" dem.

Modell:

- Skriv klart konstuktorn för GameModel. Den måste initiera instansvariabeln board (som annars sätts till null).
- Skriv metoden clearBoard. Använd den i konstruktorn för att rensa brädet när modellen skapas.
- Skriv metoden nextPlayer
- Skriv metoden claim
- Skriv metoden isInBounds
- Skriv metoden getSymbol

Vy: Skriv klart en metod i CellLabel som lyssnar efter musklick. Bäst verkar det fungera i lyssnarmetoden mouseReleased (mouseClicked är lite för känslig och missar lätt klick verkar det

som). Metoden ska först erövra rutan (med metoden claim) om den är ledig, sedan ska rutans text ändras så den speglar ändringarna i modellen.

Testa: När du kör Main.java ritas spelbrädet ut, och klickar du på någon cell så "erövras" den av den aktiva spelaren och turen övergår till nästa spelare. Att klicka på någon upptagen cell påverkar inte brädet alls. Initialt kan det vara bra att ha lite debug-utskrifter på väl valda ställen så du ser vad som händer. Se också till att det inte uppstår några exceptions när ni kör.
Tips1: om du kör System.out.println(model) så skrivs en liten representation av modellen ut snyggt (i GameModel skriver du bara System.out.println(this)). Funkar bäst för modeller med få rader.
Tips2: I Main.java ligger en metod test() som testar modellen helt utan grafiskt gränssnitt. Det kommer ännu inte fungera helt men kan vara ett snabbt sätt att testköra er kod.

Deluppgift c)

Nu ska vi utöka så att vyn så att när man för musen över en ledig cell visas den nuvarande spelarens symbol i röd text. På så vis vet också spelarna vems tur det är.

Modell: Allt som behövs finns redan :).

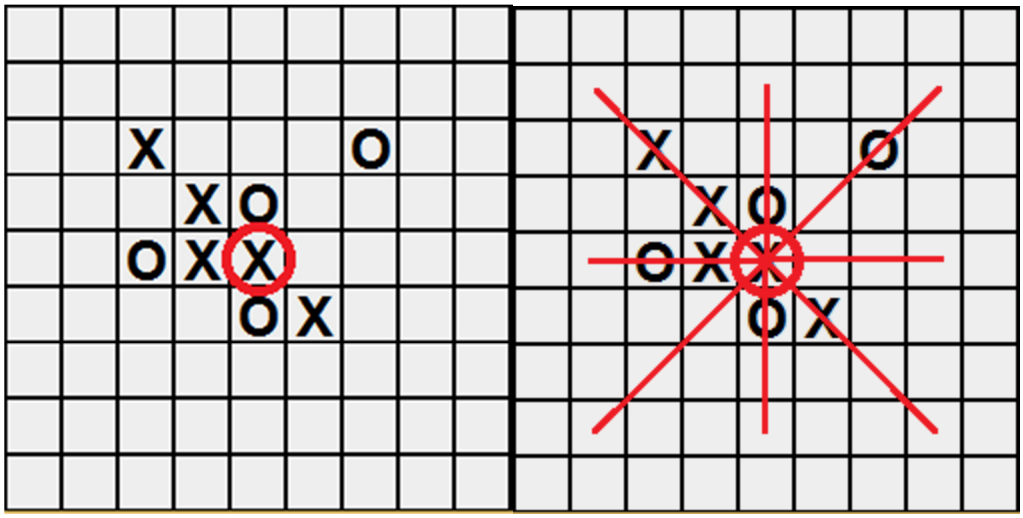
Vy: Implementera mouseEntered och mouseExited i CellLabel. Om cellen musen förs över är tom ska texten bytas till röd färg och nuvarande spelares symbol visas. När musen lämnar cellen ska färgen bytas tillbaks till svart och texten bytas tillbaks till " ".

Testa: Att föra musen över en blank cell visar den nuvarande spelarens tecken. När musen lämnar cellen återgår den till att vara blank. Att föra musen över redan erövrade celler ger ingen synlig effekt.

Deluppgift d)

Nu ska vi avgöra vem som vinner spelet! När någon vinner ska ett meddelande säga vilken spelare som vann.

För att utveckla en algoritm för att se om någon vunnit kan man notera följande: Efter att någon spelare erövrat en ruta räcker det med att kontrollera om spelaren därmed format en rad om fyra tecken längst någon av de fyra tillåtna riktingarna: rad, kolumn, nedåtdiagonal eller uppåtdiagonal. Längst varje sådan riktning finns flera sätt att vinna, men man behöver kontrollera som mest sju celler längst varje riktning (till exempel på en rad kan det vara så att den erövrade rutan är i slutet eller början av en fyra . Bilderna nedan illustrerar ett exempel. Den röda cirkeln visar cellen som just erövrats av spelare X, och de röda strecken vilka celler som behöver kontrolleras för att se om spelaren vann.



Notera att vi inte kan anta att den nya symbolen är i slutet eller början av den vinnande raden utan kan (som i det här fallet) vara någon av cellerna i mitten.

Den mystiske uppdragsigivaren åt vilken ni utvecklar spelet gjorde följande geniala observation: Eftersom varje cell är ett enda tecken kan varje rad representeras av en sträng. Då kan problemet reduceras till att se om någon av de fyra strängarna innehåller "XXXX" eller "OOOO" beroende på vem som erövrade cellen.

Till exempel skulle rad-strängen för exemplet ovan vara strängen "`-OXX---`" (där bindestrecken egentligen är mellanslag) och inte vara en vinst för den innehåller inte "XXXX". Nedåtdiagonalen skulle vara "`-XXXX--`" och alltså en en vinst.

Det här är inte det enda sättet att avgöra vem som vann, men det har fördelen att String-klassen redan har en metod för att avgöra om en sträng innehåller en annan sträng.

Modell: Implementera wonBy. Metoden ska anropas efter claim med samma Location som parameter, och kontrollerar om spelaren som erövrade cellen vann spelet.

Vy: Ändra mouseReleased så den kontrollerar om spelaren vann, och i så fall skriver ut ett meddelande.

Testa: Vinn ett par omgångar. Testa att vinna i alla fyra riktningar (under separata körningar). Testa att vinna när sista rutan placeras närmast en kant. Se till att ni inte får några exceptions när ni testar.

Deluppgift e)

Städa upp er kod. Ta bort alla TODO och tipskommentarer. Se till att alla era variabler har vettiga namn och så vidare innan ni redovisar. Gör ett sista test och jämför att er implementation gör samma sak som referensimplementationen i Labb 4.jar.

Medan ni väntar på att få redovisa kan ni spela ett par omgångar mot varandra och avgöra vem som är bäst på fyra i rad :).

Icke-obligatoriska uppgifter/övningar:

Alla uppgifter nedan är helt valfria och behöver inte redovisas.

Old school: En av fördelarna med att hålla isär modell och grafik är att det är enkelt att byta ut grafiken. I Main.java ligger ett enkelt textbaserat gränssnitt (metoden play). Prova att köra det och kanske fixa till så det inte kraschar så fort man skriver fel.

Cusom skins: Kan varje spelare få välja sin egen symbol istället för X och O? Kanske en egen färg till och med?

X i rad: Skulle ni kunna ändra så man i början av spelet anger hur många i rad man behöver för att vinna? Så man kan spela till exempel fem i rad eller 3 i rad.

Massively multiplayer edition: Skulle ni kunna ändra så det går att ha mer än två spelare?

AI: Skulle ni kunna göra en variant av spelet där datorn tar över O-spelaren? (Kan vara ganska svårt eller väldigt svårt att implementera beroende på hur duktigt ni vill att datorn ska spela)

Savegame: Utöka det textbaserade gränssnittet så man kan skriva "save" eller "load" för att skriva spelmodellen till en fil eller läsa in den från samma fil (ni måste bestämma ett format för att spara nuvarande spelare, mått på planen och själva planen). Lägg gärna till save/load-metoder i modellen och vill ni göra det svårt kan ni utöka ert grafiska gränssnitt med knappar för Save och Load.

Release: Om ni vill skapa en egen körbar jar-fil för det här eller något annat program ni skapat kan ni göra det i JGrasp:
Först måste ni skapa ett projekt
http://www.cs.unc.edu/~adyilie/comp14/Help/projects.html
Därefter skapas någorlunda enkelt en jar-fil:
http://www.cs.unc.edu/~weiss/COMP14/jar