Labbrapport

Laboration 1: Tic Tac Toe

Anders Ullström
940803
6 april 2022

Introduktion

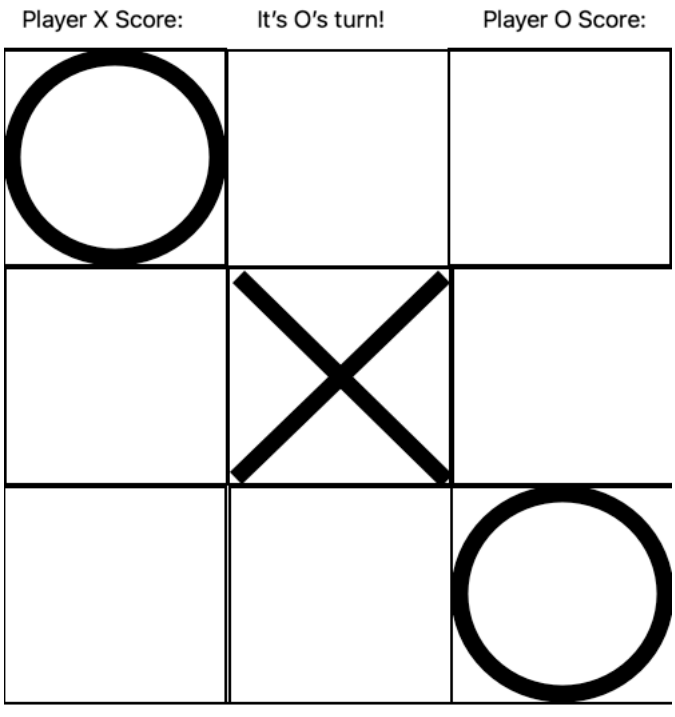
I denna laboration var syftet att skapa ett spel vid namn Tic Tac Toe. Detta har inneburit att skapa ett grafiskt användargränssnitt. Denna laboration innehöll följande kravspecifikation:

- Spelet ska vara spelbart med två spelare. Om en spelare vinner ska ett vinstmeddelande visas upp och spelet ska kunna återställas. Om spelbrädet blir fullt ska ett meddelande om att spelet är oavgjort visas upp.
- Det ska finnas räknare på toppen av skärmen som håller koll på hur många gånger X, respektive O har vunnit.
- Det ska finnas en turindikator, som visar vems tur det är.
- Varje gång X och O visas någonstans i gränssnittet ska de ha den korrekta färgen (blå eller röd), all annan text ska ha en neutral färg.

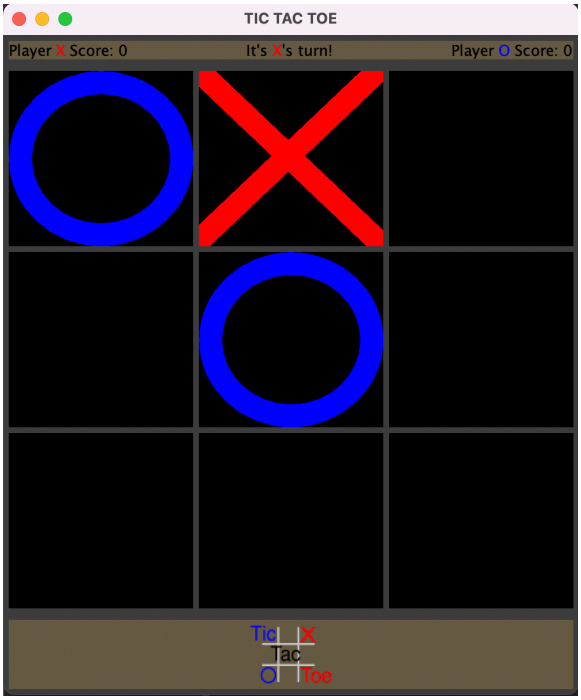
Under spelets utveckling så har artistisk frihet tagits under vissa kreations-moment. Detta kommer att nämnas i detalj under labbrapportens gång.

Grafisk design

För att uppnå en kosmetiskt tilltalande grafisk design i detta spel så skapades bland annat två stycken klasser som ärver av den inbyggda klassen JComponent. Det fanns en tydlig vision redan innan hur spelet skulle se ut när det var klart. Det som ville uppnås var brickor som fyllde hela pjäs-fältet när man placerade ut dem. En demonstration visas i bilden till höger. Detta utgjorde själva grunden för designen och det byggdes på ytterligare finess under

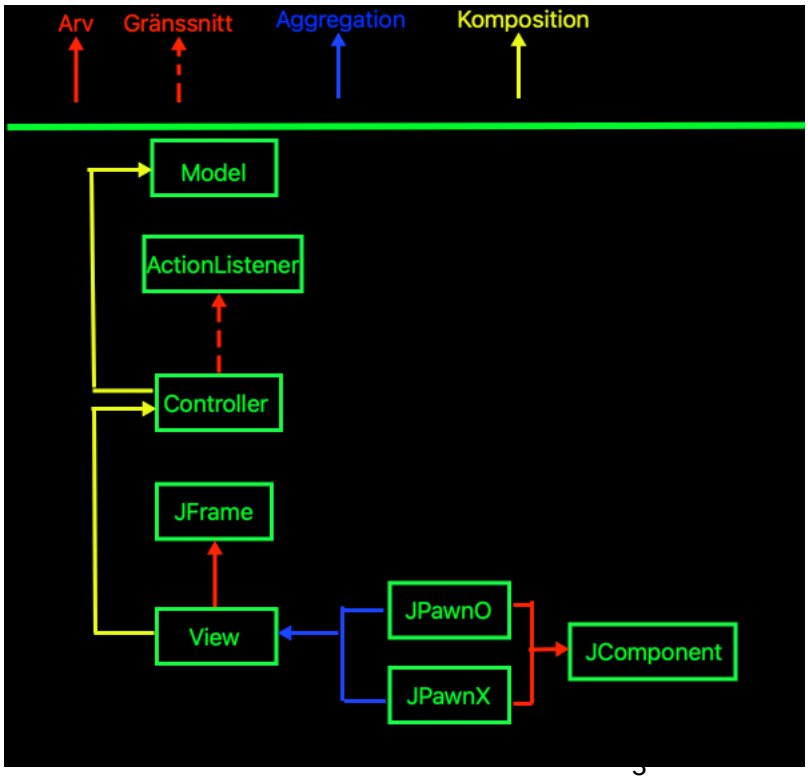


spelets utveckling som mestadels bestod av att färglägga de olika komponenterna så att det blev estetiskt tilltalande. Komponenten JOptionPane ändrades också i designen genom att ändra operativsystemets standard-ikon vid uppvisning. Denna ikon byttes ut mot laborationens bifogade PNG-fil. Den färdiga produkten blev betydligt vackrare och kan ses till höger. Bilden överst visar själva spelplanen och bilden under visar dialogrutan när en runda är över.



Programdesign

Programdesignen följer i grund och botten MVC (Model-View-Controller) vilket innebär att det finns en distinkt separation mellan själva modellen, den grafiska vyn och den logiska aspekten av programmet. I det bifogade UML-diagrammet uppvisas klassernas relation till varandra.



Implementation

Utifrån dem redan nämnda klasserna ovan i UML-diagrammet så innehåller dessa även en mängd olika objekt och metoder.

Klassen View:

Denna klass innehåller alla grafiska komponenter som är plockade ifrån Swing- eller AWT-biblioteket. Denna klass ärver ifrån JFrame som utgör själva spelskärmen när användaren startar programmet. Flertaliga objekt finns i denna klass såsom instanser av JPawnO, JPawnX, JPanels, JButtons, JLabels, ImageIcon och Color.

Views konstruktor tar emot en parameter av ett Controller-objekt som används för att skicka logisk info tillbaka till klassen Controller. Detta objekt används för att Controller ska kunna få en instans av View skickad till sig så att Controller ska kunna modifiera vyn under spelets gång.

Själva spelskärmen fylls med tre stycken JPanels med hjälp av en BorderLayout där poängpanelen hamnar på position "NORTH", spelbrädspanelen hamnar på position "CENTER" och den bifogade ikonens panel hamnar på position "SOUTH". Spelbrädets panel delas därefter in med hjälp av en GridLayout av tre rader och tre kolumner där 9 stycken instanser av JButton läggs in. Dessa instanser kallar även på metoderna addActionListener() och setActionCommand(). Varje JButton får ett unikt ID i form av en siffra så att klassen Controller sedan ska kunna urskilja vilken knapp som blivit nedtryckt.

Poängpanelens underliggande objekt struktureras med hjälp av en BoxLayout som följer X-axeln. I denna panel läggs otaliga instanser av JLabels in med klister emellan för att forma en vacker poängpanel.

Ikonpanelens enda innehåll är den bifogade ikonen som implementeras med klassen ImageIcon. Slutligen så finns några metoder implementerade i klassen View vid namn loadIcon(), setPawn(), setScore(), clearBoard() och setJOptionPane(). Metoden setPawn() placerar ut en bricka av antingen JPawnO

eller JPawnX beroende på vems tur det är. Metoden setScore() ökar poängen vid slutet av varje runda med ett för den som vann. Metoden clearBoard() rensar brickorna av typen JPawnO och JPawnX från spelbrädet vid slutet av varje runda. Metoden setJOptionPane() visar upp en dialogruta vid en spelrundas slut med en eventuell vinnare.
De flesta JComponents modifieras något med hjälp av klassen Color och Border för att göra programmet vackrare till ögat.
JPawnO och JPawnX instansieras som arrays med en storlek av nio (varför detta gjordes till arrays nämns i kapitlet "Problem").

Klassen Controller:

Denna klass har redan diskuterats något ovan, men ytan har bara blivit skrapad. Detta är den viktigaste delen av hela programmet per se eftersom den gör precis som det låter – kontrollerar hela programmet. Först och främst så skapas en instans av Model och en instans av View. Klassen har även en ActionListener implementerad och därav även actionPerformed-metoden. När View passar över ett action-command av vilken JButton som har blivit nedtryckt så använder klassen Controller detta i metoden actionPerformed(). Hur metoden kan urskilja vart parametern ska ta vägen så använder den sig av nio stycken if-satser (en för vardera knapp). Efter actionPerformed() har bestämt vilken if-sats som ska triggas så utförs dess kod som är nästintill identisk i varje if-sats. Denna if-sats ändrar först en boolean till sann som gör att knappen inte går att trycka ned igen. Sedan så skickar den data till klassen Model med dess metod setClickedBox() för att ändra modellens array av knappar för just den klickade knappen. Sedan kallas en metod ifrån instansen av View som heter setPawn() som placerar ut pjäsen som Controller skickar med som parameter. Sedan kallas metoden checkGameState() som först kollar om någon har fått tre i rad med hjälp av metoden getClickedBox() ifrån instansen av klassen Model. Om detta är sant så kollar programmet vems runda det är från modellens metod getPawn() och delar sedan ut poäng, visar en dialogruta med vinnaren och

startar om all nödvändig logik till nästa runda. Det kan även bli oavgjort och detta följer samma mönster som nämnts i meningarna innan förutom att ingen poäng delas ut.

Klassen Model:

Denna klass består enbart av variabler av booleans som instansierats som arrays och en int som lagrar vems runda det är. Vems runda det är avgörs i metoden getPawn() som utgörs av en if-sats med en modulo-operator som har modulo 2. Denna modulo-operator jämför med variabeln ”turn” varje gång metoden kallas på och ökar sedan ”turn” med ett. Metoden returnerar 0 om resten är 0 och returnerar 1 om resten är 1 vid division med 2. Denna klass innehåller andra metoder också som har täckts i sektionerna för klasserna View och Controller ovan.

Problem

Det uppstod problem med uppritningen av instanserna av JPawnO och JPawnX. När storleken ändrades på fönstret så försvann alla uppritade JPawns förutom den sist uppritade av både JPawnO och JPawnX. Det tog ett tag innan problemet hittades. Det insågs efter ett tag att det inte gick att ha en enda instans av klasserna JPawnO och JPawnX för att Java Virtual Machine inte sparar minne om det inte behöver. De senast uppritade objekten slog alltså ut dem gamla objekten och detta blev först märkbart när skärmstorleken ändrades. Detta löstes genom att skapa en instans av en array med 9 platser för både JPawnO och JPawnX. Sedan blev varje index i arrayen en egen JPawnO och JPawnX. Sedan fick varje index en bestämd plats i spelbrädet där den skulle placeras ut om någon klickade på den knappen. Detta gjorde så att alla JPawns ritades upp under hela spelets gång utan att slå ut varandra.

Slutsats

Denna laboration har varit väldigt givande. Den gav en stor förståelse för hur arbete i Swing kan utföras med rätt tänk. Model-View-Controller var ett nytt

tankesätt som tog ett tag att greppa, men det blir bättre att följa denna modell insågs rätt fort. Denna modell gör så att problem lättare kan hittas eftersom koden blir separerad i olika sektioner med olika syften.
En sak som skulle kunna gjorts bättre är att implementera flera klasser. En klass som skulle kunna göras är GameBoard som skapar hela spelbrädet. Allt detta för att göra koden mer lättläst för andra programmerare.

Skriv för att lägga till text

Referenser

- www.stackoverflow.com
- Karlstads Universitet. (2022). *Föreläsning 1: Introduktion.* https://kau.instructure.com/courses/13875/pages/forelasning-1-introduktion?module_item_id=386737

  Kurs: Grafiska Användargränssnitt
- Karlstads Universitet. (2022). *Föreläsning 2: Grundläggande Java och Objektorientering.* https://kau.instructure.com/courses/13875/pages/forelasning-2-grundlaggande-java-och-objektorientering?module_item_id=390976

  Kurs: Grafiska Användargränssnitt
- Karlstads Universitet. (2022). *Föreläsning 3: Swing Widgets.* https://kau.instructure.com/courses/13875/pages/forelasning-3-swing-widgets?module_item_id=386738

  Kurs: Grafiska Användargränssnitt
- Karlstads Universitet. (2022). *Föreläsning 4: Swing Layouts.* https://kau.instructure.com/courses/13875/pages/forelasning-4-swing-layouts?module_item_id=386739

  Kurs: Grafiska Användargränssnitt
- Karlstads Universitet. (2022). *Föreläsning 5: Rita i Swing.* https://kau.instructure.com/courses/13875/pages/forelasning-5-rita-i-swing?module_item_id=386740

Kurs: Grafiska Användargränssnitt

- Karlstads Universitet. (2022). *Föreläsning 6 MVC och Observer Pattern.* https://kau.instructure.com/courses/13875/pages/forelasning-6-mvc-och-observer-pattern?module_item_id=386741

  Kurs: Grafiska Användargränssnitt

- Karlstads Universitet. (2022). *Föreläsning 5: Objektorientering 3. UML-Diagram [Video].* https://kau.instructure.com/courses/14375/pages/f5-objektorientering-3?module_item_id=368598

  Kurs: Programutvecklingsmetodik