```
------------------------------------------------------------
-- Connect Four (TM) GNAPPLET
--
-- By:  Barry Fagin and Martin Carlisle
-- US Air Force Academy, Department of Computer Science
-- mailto:carlislem@acm.org
--
-- Adapted for JVM-GNAT GPL 2009 by Pascal Pignard
-- http://blady.pagesperso-orange.fr
--
-- This is free software; you can redistribute it and/or
-- modify without restriction.  We do ask that you please keep
-- the original author information, and clearly indicate if the
-- software has been modified.
--
-- This software is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty
-- of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
------------------------------------------------------------

with Java.Lang.String;   use Java.Lang.String;
with Java;               use Java;
with Java.Applet.Applet; use Java.Applet.Applet;
with Java.Awt.Color;
with Java.Awt.Dimension;
with Java.Awt.Graphics;  use Java.Awt.Graphics;
with Java.Net.Url;
with Java.Awt.Component;
--  The following comments are adapted from the JGNAT Tic-Tac-Toe
--  example program (C) Ada Core Technologies
--
--     * In the Init function of your applet, you need to call
--       Adainit subprogram to elaborate your program. In the
--       case of this applet it's ada_connectfour.adainit.
--
--   To execute the applet, you need to run the Java appletviewer
--   or a Java-capable browser on an html file such as the following:
--
--     <html>
--     <head>
--     <title>Connect Four (TM) Game, written in Ada</title>
--     </head>
--
--     <body>
--
--     <h1>Built using JVM-GNAT, The Ada 2005 to JVM compiler </h1>
--
--     <P ALIGN=center>
--
--     <APPLET CODEBASE="."
--             CODE="connectfour$typ.class"
--             ARCHIVE="connectfour.jar"
--             WIDTH=500 HEIGHT=300>
--     Sorry, can't show the applet</APPLET>
--
--     </P>
--
--     <hr>
--     </body>
--     </html>
--
--   See the Makefile provided in the directory for this demo
--   for specific details on how to create and run the applet.

package body Connectfour is
   ------------------------
   -- Local Types & Data --
   ------------------------

   Num_Rows    : constant Integer := 6;
   Num_Columns : constant Integer := 7;
   -- Constants for the board size
```

```
   type Player_Kind is (None, Computer, User);
   --  None means that neither the computer, nor the user have selected that
   --  circle, Computer indicates that the circle has been selected by the
   --  computer and User means that the circle has been selected by the user.

   -- data type for the Connect Four board
   type Board_Array_Type is array (1 .. Num_Rows, 1 .. Num_Columns) of Player_Kind;

   -- This image is used for double-buffering.  See the update
   -- method.
   Off_Screen_Buffer : access Java.Awt.Image.Typ'Class;
   use type Java.Awt.Image.Ref;

   -- globals that maintain state
   Board        : Board_Array_Type;  -- the current board

   -- Is the game over, and if so who won?
   Computer_Won : Boolean          := False;
   User_Won     : Boolean          := False;
   Tie          : Boolean          := False;

   -- if user clicks in full column, computer should not take a turn
   -- Also computer should not take turn if user wins or tie.
   Ignore_Turn : Boolean := False;

   -- The following constants are used to define the layout of the
   -- board.  They define the horizontal and vertical spacing of the
   -- circles drawn on the screen.

   Ytop         : constant := 20;  -- highest pos on screen
   Ybottom      : constant := 279; -- lowest pos on screen
   Xleft        : constant := 0;   -- leftmost pos on screen
   Xright       : constant := 499; -- rightmost pos on screen
   Title_Offset : constant := 4;   -- left move from center of column
   Title_Height : constant := 12;  -- height of column numbers

   -- Both horizontally and vertically, there are circles and intervals
   -- (or spaces) between the circles.
   -- If there is half an interval on the left and right ends, then
   -- there are 7 full intervals (because there are 6 full intervals
   -- not counting the ends) horizontally
   -- Now, assuming the intervals are 1/4th as wide as the circles
   -- (The 1/4th is completely arbitrary),
   -- then we need 7 + 7/4  (8 3/4) circles worth of space across the
   -- screen.

   Circle_Width : constant Float := Float (Xright - Xleft + 1) / 8.75;

   X_Space : constant Float := Float (Xright - Xleft + 1) / 7.0;
   -- the horizontal space between circle centers

   X_First : constant Integer := Xleft + Integer (0.625 * Circle_Width);
   -- the first x coordinate is to the middle of the first circle,
   -- which is 1/8 circle size + 1/2 circle size (5/8)


   -- Similarly, vertically there will be 6 full intervals, and again
   -- assuming an interval is 1/4 as tall as the space for the circle,
   -- we get 6 + 6/4 (7 1/2) circles vertically on the screen
   Circle_Height : constant Float   := Float (Ybottom - Ytop + 1) / 7.5;

   Y_Space       : constant Float   := Float (Ybottom - Ytop + 1) / 6.0;

   Y_First       : constant Integer := Ytop + Integer (0.625 * Circle_Height);


   -- column_breaks holds the x coordinates where the transition from
   -- one column to the next occurs
   -- That is, column_breaks(1) is the rightmost x coordinate where
   -- you can click and still be in column 1

   type Column_Breaks_Array_Type is array (1 .. Num_Columns) of Integer;
```

```
   Column_Breaks : constant Column_Breaks_Array_Type := (
      Integer (1.25 * Circle_Width),
      Integer (2.5 * Circle_Width),
      Integer (3.75 * Circle_Width),
      Integer (5.00 * Circle_Width),
      Integer (6.25 * Circle_Width),
      Integer (7.5 * Circle_Width),
      Integer (8.75 * Circle_Width));

   ----------------------
   -- Initialize_Board --
   ----------------------

   -- Initializes board to all none

   procedure Initialize_Board (
         Board :    out Board_Array_Type ) is
   begin
      Board := (others => (others => None));
   end Initialize_Board;

   ------------------------------------------------------------------------
   --
   -- Name : Place_Disk
   -- Description : Determines the row in the given column at which
   --               who's disk should be placed (in the lowest empty
   --               row, where a lower row has a higher index).  Puts
   --               who at that row/column in the board, then calls
   --               Draw_Position to update the screen.
   --
   ------------------------------------------------------------------------

   procedure Place_Disk (
         Board  : in out Board_Array_Type;
         Column : in     Integer;
         Row    :    out Integer;
         Who    : in     Player_Kind           ) is

   begin

      Row := 1;

      -- starting at the top, loop until you find an non-empty row
      -- in this column
      while ( Row <= Num_Rows ) and then
            ( Board(Row,Column) = None ) loop
         Row := Row + 1;
      end loop;
      -- the new disk will be placed just above the first non-empty row
      Row := Row - 1;

      -- place the disk
      Board(Row, Column) := Who;
   end Place_Disk;

   ---------------
   -- Check_Won --
   ---------------

   -- Checks to see if Who won

   procedure Check_Won (
         Board : in     Board_Array_Type;
         Who   : in     Player_Kind;
         Won   :    out Boolean           ) is
   begin

      --    Set Won to false
      Won := False;

      --    Loop through all rows
      for Row in Board'Range(1) loop
```

```
         --       Loop through all columns
         for Column in Board'Range(2) loop

            --          (checking row to the right)
            --          If column <= Num_Columns - 3
            if ( Column <= Num_Columns - 3 ) then

               --             If current location and row, column+1;
               --                row, column+2; and
               --                row, column+3 belong to who
               if ( Board(Row, Column) = Who ) and
                     ( Board(Row, Column + 1) = Who ) and
                     ( Board(Row, Column + 2) = Who ) and
                     ( Board(Row, Column + 3) = Who ) then

                  --                Set Won to true
                  Won := True;

               end if;

            end if;

            --          (checking column down)
            --          If row <= Num_Rows - 3
            if ( Row <= Num_Rows - 3 ) then

               --             If current location and row+1, column;
               --                row+2, column; and
               --                row+3, column belong to who
               if ( Board(Row, Column) = Who ) and
                     ( Board(Row + 1, Column) = Who ) and
                     ( Board(Row + 2, Column) = Who ) and
                     ( Board(Row + 3, Column) = Who ) then

                  --                Set Won to true
                  Won := True;

               end if;

            end if;

            --          (checking diagonal up to right)
            --          If row >= 4 and column <= Num_Columns - 3
            if ( Row >= 4 ) and
                  ( Column <= Num_Columns - 3 ) then

               --             If current location and row-1, column+1;
               --                row-2, column+2;
               --                and row-3,column+3 belong to who
               if ( Board(Row, Column) = Who ) and
                     ( Board(Row - 1, Column + 1) = Who ) and
                     ( Board(Row - 2, Column + 2) = Who ) and
                     ( Board(Row - 3, Column + 3) = Who ) then

                  --                Set Won to true
                  Won := True;

               end if;

            end if;

            --          (checking diagonal down to right)
            --          If row <= Num_Rows - 3 and column <= Num_Columns - 3
            if ( Row <= Num_Rows - 3 ) and
                  ( Column <= Num_Columns - 3 ) then

               --             If current location and row+1, column+1;
               --                row+2, column+2;
               --                and row+3,column+3 belong to who
               if ( Board(Row, Column) = Who ) and
                     ( Board(Row + 1, Column + 1) = Who ) and
                     ( Board(Row + 2, Column + 2) = Who ) and
                     ( Board(Row + 3, Column + 3) = Who ) then
```

```
                --                Set Won to true
                Won := True;

             end if;

          end if;

       end loop;

    end loop;

end Check_Won;

---------------
-- Check_Tie --
---------------

--  Checks to see if the game has ended in a tie (all columns are full)

procedure Check_Tie (
      Board  : in     Board_Array_Type;
      Is_Tie :    out Boolean           ) is

begin

   --    Set Is_Tie to True
   Is_Tie := True;

   -- If we find any row with top column empty, then
   -- it is NOT a tie.
   for Index in Board'Range(2) loop
      if ( Board (1,Index) = None ) then
         Is_Tie := False;
      end if;
   end loop;

end Check_Tie;


-------------------
-- Computer_Turn --
-------------------

--  Uses lookahead and live tuple heuristic

procedure Computer_Turn (
      Board  : in     Board_Array_Type;
      Column :    out Integer           ) is

   Lookahead_Depth : constant Integer := 5;
   type Column_Breaks_Array_Type is array (1 .. Num_Columns) of Integer;

   type Value_Type is --need two ties for symmetry
         (Illegal,
          Win_For_User,
          Tie_For_User,
          Unknown,
          Tie_For_Computer,
          Win_For_Computer);

   type Value_Array_Type is array (1 .. Num_Columns) of Value_Type;

   --------------------
   -- Make_New_Board --
   --------------------

   procedure Make_New_Board (
         New_Board :    out Board_Array_Type;
         Board     : in     Board_Array_Type;
         Who       :        Player_Kind;
         Column    :        Integer           ) is
      Row : Integer;
```

```
begin
   New_Board := Board;
   Place_Disk(New_Board,Column,Row,Who);
end Make_New_Board;

----------------------
-- Find_Best_Result --
----------------------

function Find_Best_Result (
      Evaluations : in     Value_Array_Type;
      Who         :        Player_Kind       )
  return Value_Type is
   Best_Result : Value_Type;
begin
   if Who = Computer then
      --find "largest" move
      Best_Result := Win_For_User;
      for I in Evaluations'range loop
         if Evaluations(I) > Best_Result and Evaluations(I) /=
               Illegal
               then
            Best_Result := Evaluations(I);
         end if;
      end loop;
   else
      --Who = User, find "smallest" move
      Best_Result := Win_For_Computer;
      for I in Evaluations'range loop
         if Evaluations(I) < Best_Result and Evaluations(I) /=
               Illegal
               then
            Best_Result := Evaluations(I);
         end if;
      end loop;
   end if;
   return Best_Result;

end Find_Best_Result;

------------------------
-- Weighting_Function --
------------------------

function Weighting_Function (
      Arg : in     Integer )
  return Integer is
begin
   return(Arg*Arg*Arg);
   --use cubic for now
end Weighting_Function;

---------------------------
-- Evaluate_Unknown_Board --
---------------------------

function Evaluate_Unknown_Board (
      Board : in     Board_Array_Type )
  return Integer is
   Owner          : Player_Kind;
   Cell           : Player_Kind;
   User_Count,
   Computer_Count,
   Board_Value    : Integer;
   Dead           : Boolean;
begin
   Board_Value := 0;

   for Row in Board'range(1) loop

      for Column in Board'range(2) loop

         --          (checking horizontal tuples)
```

```
if ( Column <= Num_Columns - 3 ) then
   Owner := None;
   User_Count := 0;
   Computer_Count := 0;
   Dead := False;
   for I in 0..3 loop
      Cell := Board(Row, Column+I);
      if Owner = None and Cell /= None then
         Owner := Cell;
      end if;
      if (Cell = User and Owner = Computer) or (Cell
            = Computer and Owner = User) then
         User_Count := 0;
         Computer_Count := 0;
         Dead := True;
      end if;
      if Cell = User and not Dead then
         User_Count := User_Count+1;
      elsif
            Cell = Computer and not Dead then
         Computer_Count := Computer_Count+1;
      end if;
   end loop;

   --  Computer count is positive, User count is negative so
   --  that larger values are better for computer

   Board_Value := Board_Value + Weighting_Function(
      Computer_Count) -
      Weighting_Function(User_Count);

end if;

--          (checking vertical tuples)

if ( Row <= Num_Rows - 3 ) then
   Owner := None;
   User_Count := 0;
   Computer_Count := 0;
   Dead := False;

   for I in 0..3 loop
      Cell := Board(Row+I, Column);
      if Owner = None and Cell /= None then
         Owner := Cell;
      end if;
      if (Cell = User and Owner = Computer) or (Cell
            = Computer and Owner = User) then
         User_Count := 0;
         Computer_Count := 0;
         Dead := True;
      end if;
      if Cell = User and not Dead then
         User_Count := User_Count+1;
      elsif
            Cell = Computer and not Dead then
         Computer_Count := Computer_Count+1;
      end if;
   end loop;
   Board_Value := Board_Value + Weighting_Function(
      Computer_Count) -
      Weighting_Function(User_Count);
end if;

--          (checking diagonal tuples up to right)

if ( Row >= Num_Rows/2+1 and Column <= Num_Columns-
      3 ) then
   Owner := None;
   User_Count := 0;
   Computer_Count := 0;
   Dead := False;
```

```
               for I in 0..3 loop
                  Cell := Board(Row-I, Column+I);
                  if Owner = None and Cell /= None then
                     Owner := Cell;
                  end if;
                  if (Cell = User and Owner = Computer) or (Cell
                        = Computer and Owner = User) then
                     User_Count := 0;
                     Computer_Count := 0;
                     Dead := True;
                  end if;
                  if Cell = User and not Dead then
                     User_Count := User_Count+1;
                  elsif
                        Cell = Computer and not Dead then
                     Computer_Count := Computer_Count+1;
                  end if;
               end loop;
               Board_Value := Board_Value + Weighting_Function(
                  Computer_Count) -
                  Weighting_Function(User_Count);
            end if;

            --          (checking diagonal tuples down to right)

            if ( Row <= Num_Rows - 3 ) and (Column <= Num_Columns
                  -3) then
               Owner := None;
               User_Count := 0;
               Computer_Count := 0;
               Dead := False;

               for I in 0..3 loop
                  Cell := Board(Row+I, Column+I);
                  if Owner = None and Cell /= None then
                     Owner := Cell;
                  end if;
                  if (Cell = User and Owner = Computer) or (Cell
                        = Computer and Owner = User) then
                     User_Count := 0;
                     Computer_Count := 0;
                     Dead := True;
                  end if;
                  if Cell = User and not Dead then
                     User_Count := User_Count+1;
                  elsif
                        Cell = Computer and not Dead then
                     Computer_Count := Computer_Count+1;
                  end if;
               end loop;
               Board_Value := Board_Value + Weighting_Function(
                  Computer_Count) -
                  Weighting_Function(User_Count);
            end if;

         end loop;

      end loop;

      return Board_Value;
   end Evaluate_Unknown_Board;

   --------------------
   -- Evaluate_Board --
   --------------------

   function Evaluate_Board (
         Board          : in     Board_Array_Type;
         Who_Just_Moved : in     Player_Kind;
         Current_Depth  : in     Integer          )
     return Value_Type is
      Computer_Won,
```

```
      User_Won,
      Is_Tie         : Boolean;
      Value          : Value_Type;
      Who_Moves_Next : Player_Kind;
      New_Board      : Board_Array_Type;
      Evaluations    : Value_Array_Type;
   begin

      Check_Won (
         Board => Board,
         Who   => Computer,
         Won   => Computer_Won);
      if not Computer_Won then
         Check_Won(
            Board => Board,
            Who   => User,
            Won   => User_Won);
         if not User_Won then
            Check_Tie(Board,Is_Tie);
         end if;
      end if;
      if Computer_Won then
         Value := Win_For_Computer;
      elsif
            User_Won then
         Value := Win_For_User;
      elsif
            Is_Tie and Who_Just_Moved = User then
         Value := Tie_For_User;
      elsif
            Is_Tie and Who_Just_Moved = Computer then
         Value := Tie_For_Computer;
      elsif
            Current_Depth = 1 then
         Value := Unknown;
      else
         --Not a terminal node or end of lookahead, so recurse

         if Who_Just_Moved = Computer then
            Who_Moves_Next := User;
         else
            Who_Moves_Next := Computer;
         end if;

         for Col in Evaluations'range loop
            Evaluations(Col) := Illegal;
         end loop;

         for Col in Board'range(2) loop

            if Board(1,Col) = None then

               Make_New_Board(New_Board,Board,Who_Moves_Next,Col);
               Evaluations(Col) := Evaluate_Board(
                  New_Board,Who_Moves_Next,Current_Depth-1);

               --a/b pruning

               exit when Evaluations(Col) = Win_For_Computer and
                     Who_Moves_Next = Computer;

               exit when Evaluations(Col) = Win_For_User and
                       Who_Moves_Next  = User;

            else
               Evaluations(Col) := Illegal;
            end if;
         end loop;
         Value := Find_Best_Result(Evaluations, Who_Moves_Next);
      end if;

      return Value;
   end Evaluate_Board;
```

```
      --------------------
      -- Find_Best_Move --
      --------------------

      function Find_Best_Move (
            Evaluations : in     Value_Array_Type;
            Who         :        Player_Kind        )
        return Integer is
         Best_Move   : Integer;
         Best_Result : Value_Type;
      begin
         if Who = Computer then
            --find "largest" move
            Best_Result := Win_For_User;
            for I in Evaluations'range loop
               if Evaluations(I) > Best_Result and Evaluations(I) /=
                     Illegal
                     then
                  Best_Result := Evaluations(I);
                  Best_Move := I;
               end if;
            end loop;
         else
            --Who = User, find "smallest" move
            Best_Result := Win_For_Computer;
            for I in Evaluations'range loop
               if Evaluations(I) < Best_Result and Evaluations(I) /=
                     Illegal
                     then
                  Best_Result := Evaluations(I);
                  Best_Move := I;
               end if;
            end loop;
         end if;

         return Best_Move;

      end Find_Best_Move;

      -----------------------
      -- Find_All_Unknowns --
      -----------------------

      procedure Find_All_Unknowns (
            Evaluations : in     Value_Array_Type;
            Moves       :    out Column_Breaks_Array_Type;
            Count       :    out Integer                    ) is
      begin
         Count := 0;
         for I in Evaluations'range loop
            if Evaluations(I) = Unknown then
               Count := Count + 1;
               Moves(Count) := I;
            end if;
         end loop;
      end Find_All_Unknowns;

      --variables and body for "Computer_Turn"

      New_Board        : Board_Array_Type;
      Evaluations      : Value_Array_Type;
      Moves_To_Unknown : Column_Breaks_Array_Type;
      Count_Unknowns   : Integer;
      Value,
      Max_Value,
      Best_Move        : Integer;
   begin
      Evaluations := (others => Illegal);

      for Col in Board'range(2) loop
         if Board(1,Col) = None then
            Make_New_Board(New_Board,Board,Computer,Col);
```

```
            Evaluations(Col) := Evaluate_Board(New_Board,Computer,
               Lookahead_Depth);

            --a/b pruning

            exit when Evaluations(Col) = Win_For_Computer;

         else
            Evaluations(Col) := Illegal;
         end if;
      end loop;

      Column := Find_Best_Move(Evaluations,Computer);

      --Check if trapped, if so take best move at shallower depth
      --and hope for a mistake

      if Evaluations(Column) = Win_For_User then
         for Col in Board'range(2) loop
            if Board(1,Col) = None then
               Make_New_Board(New_Board,Board,Computer,Col);
               Evaluations(Col) := Evaluate_Board(New_Board,Computer,2);
            else
               Evaluations(Col) := Illegal;
            end if;
         end loop;
         Column := Find_Best_Move(Evaluations,Computer);
      elsif Evaluations(Column) = Unknown then

         --If choosing from multiple unknown boards, apply heuristics.  This
         --is where most of the strategy is.

         Find_All_Unknowns(Evaluations,Moves_To_Unknown,Count_Unknowns);

         Max_Value := -1000;
         for I in 1..Count_Unknowns loop
            Make_New_Board(New_Board,Board,Computer,Moves_To_Unknown
               (I));
            Value := Evaluate_Unknown_Board(New_Board);
            if Value > Max_Value then
               Max_Value := Value;
               Best_Move := Moves_To_Unknown(I);
            end if;
         end loop;

         --unknown boards

         Column := Best_Move;
      end if;
      --picking from multiple unknown boards

   exception
      when others =>
         Column := 1;
         loop
            exit when Board(1,Column) = None;
            Column := Column + 1;
         end loop;
   end Computer_Turn;


   ----------
   -- Init --
   ----------

   procedure Init (
         This : access Typ ) is
      procedure Adainit;
      pragma Import (Ada, Adainit, "ada_connectfour.adainit");
   begin
      Adainit;
      --  The above call is needed for elaboration
```

```
      Addmouselistener (This, This.I_Mouselistener);
      Initialize_Board(Board => Board);
      Computer_Won := False;
      User_Won     := False;
      Tie          := False;
      Ignore_Turn  := False;
      This.User_Turn   := True;
      Showstatus(This, +
         "Connect Four (TM) by Barry Fagin and Martin Carlisle");
   end Init;


   -----------
   -- Paint --
   -----------

   procedure Paint (
         This : access Typ;
         G1   : access Java.Awt.Graphics.Typ'Class ) is
      D    : access Java.Awt.Dimension.Typ'Class := Getsize (This);
      Xoff : Int                    := D.Width / 3;
      Yoff : Int                    := D.Height / 3;
      ------------------------------------------------------------
      -- procedure Display_Text
      --
      -- display text in black at the given coordinates
      ------------------------------------------------------------
      procedure Display_Text (
            X    : in     Integer;
            Y    : in     Integer;
            Text : in     String   ) is
      begin
         Setcolor(G1,Java.Awt.Color.Black);
         Drawstring(G1,+Text,X,Y);
      end Display_Text;

      ------------------------------------------------------------
      -- procedure Draw_Line
      --
      -- display line in given color at the given coordinates
      ------------------------------------------------------------
      procedure Draw_Line (
            X1   : in     Integer;
            Y1   : in     Integer;
            X2   : in     Integer;
            Y2   : in     Integer;
            Hue :      access Java.Awt.Color.Typ'Class ) is
      begin
         Setcolor(G1,Hue);
         Drawline(G1,X1,Y1,X2,Y2);
      end Draw_Line;

      ------------------------------------------------------------
      -- procedure Draw_Circle
      --
      -- Draw a circle of the given color with given center
      -- and radius.  Will be filled based on filled parameter
      ------------------------------------------------------------
      procedure Draw_Circle (
            X      : in     Integer;
            Y      : in     Integer;
            Radius : in     Integer;
            Hue    :      access Java.Awt.Color.Typ'Class;
            Filled : in     Boolean             ) is
      begin
         Setcolor(G1,Hue);
         if Filled then
            Filloval(G1,X-Radius,Y-Radius,2*Radius,2*Radius);
         else
            Drawoval(G1,X-Radius,Y-Radius,2*Radius,2*Radius);
         end if;
      end Draw_Circle;
```

```
      ------------------------------------------------------------
      -- procedure Draw_Box
      --
      -- display rectangle in given color at the given coordinates
      -- will be filled (vs. outline only) based on filled parameter
      ------------------------------------------------------------
      procedure Draw_Box (
            X1     : in     Integer;
            Y1     : in     Integer;
            X2     : in     Integer;
            Y2     : in     Integer;
            Hue    :      access Java.Awt.Color.Typ'Class;
            Filled : in     Boolean             ) is
      begin
         Setcolor(G1,Hue);
         if Filled then
            Fillrect(G1,X1,Y1,X2-X1,Y2-Y1);
         else
            Drawrect(G1,X1,Y1,X2-X1,Y2-Y1);
         end if;
      end Draw_Box;


-------------------------------------------------------------------------
      --
      -- Name : Draw_Position
      -- Description : Draws a disk with the appropriate color for the
      --               given player at the given row and column
      --

-------------------------------------------------------------------------

      procedure Draw_Position (
            Who    : in     Player_Kind;
            Row    : in     Integer;
            Column : in     Integer  ) is

         -- for later
         Color         :          access Java.Awt.Color.Typ'Class;
         -- color of disk
         Circle_Radius :          Integer;
         -- radius of disk
      begin
         -- Determine radius based on minimum of possible height/width
         if Circle_Width < Circle_Height then
            Circle_Radius := Integer(Circle_Width * 0.5);
         else
            Circle_Radius := Integer(Circle_Height * 0.5);
         end if;

         -- Determine color of disk
         case Who is
            when None =>
               Color := Java.Awt.Color.White;
            when Computer =>
               Color := Java.Awt.Color.Red;
            when User =>
               Color := Java.Awt.Color.Blue;
         end case;

         Draw_Circle(
            X      => X_First + Integer (Float (Column - 1) * X_Space),
            Y      => Y_First + Integer (Float (Row - 1) * Y_Space),
            Radius => Circle_Radius,
            Hue    => Color,
            Filled => True);
      end Draw_Position;


-------------------------------------------------------------------------
      --
      -- Name : Print_Board
      -- Description : Prints the board for the start of the game.  This
```

```
      --               procedure should NOT be called repeatedly.  Rather,
      --               this procedure is called once to draw the game board,
      --               then draw_position is used to add player's disks as
      --               the game progresses.
      --

-------------------------------------------------------------------------

      procedure Print_Board (
            Board : in     Board_Array_Type ) is
      begin
         -- change the screen color if the game is over.
         if User_Won or Tie then
            Draw_Box(
               X1     => 0,
               Y1     => 0,
               X2     => 499,
               Y2     => 299,
               Hue    => Java.Awt.Color.Pink,
               Filled => True);
         elsif Computer_Won then
            Draw_Box(
               X1     => 0,
               Y1     => 0,
               X2     => 499,
               Y2     => 299,
               Hue    => Java.Awt.Color.gray,
               Filled => True);
         else
            Draw_Box(
               X1     => 0,
               Y1     => 0,
               X2     => 499,
               Y2     => 299,
               Hue    => Java.Awt.Color.Lightgray,
               Filled => True);
         end if;

         --    Print column numbers
         for Column in 1 .. Num_Columns loop
            Display_Text(
               X    => X_First + Integer (Float (Column - 1) * X_Space) -
Title_Offset,
               Y    => Title_Height,

               Text => Character'Val (Column + 48) & "");
            -- Draw vertical line between columns
            if Column < Num_Columns then
               Draw_Line(
                  X1  => Column_Breaks (Column),
                  Y1  => Ytop,
                  X2  => Column_Breaks (Column),
                  Y2  => Ybottom,
                  Hue => Java.Awt.Color.Black);
            end if;
            for Row in 1 .. Num_Rows loop
               Draw_Position(
                  Who    => Board (Row, Column),
                  Row    => Row,
                  Column => Column);
            end loop;
         end loop;

         -- Print message if the game is over
         if Computer_Won then
            Display_Text(
               X    => 0,
               Y    => 285,
               Text => "I win! - Press left mouse button");
         elsif Tie then
            Display_Text(
               X    => 0,
               Y    => 285,
```

```
            Text => "Tie Game! - Press Left Mouse Button");
      elsif User_Won then
         Display_Text(
            X    => 0,
            Y    => 285,
            Text => "You win! - Press left mouse button");
      end if;
   end Print_Board;
begin
   Print_Board(Board);
end Paint;

-------------------------------------
-- procedure Update
--
-- uses an off screen image to double
-- buffer, thus smoothing drawing.
-------------------------------------

procedure Update (
      This : access Typ;
      G    : access Java.Awt.Graphics.Typ'Class ) is
   Gr : access Java.Awt.Graphics.Typ'Class;
   Ignore : Java.Boolean;
begin
   -- need to allocate Off_Screen_Buffer only once
   if Off_Screen_Buffer = null then
      Off_Screen_Buffer := CreateImage(This,500,300);
   end if;

   -- draw into the offscreen buffer
   Gr := Java.Awt.Image.GetGraphics(Off_Screen_Buffer);
   Paint (This, Gr);

   -- copy offscreen buffer onto applet window
   Ignore := Java.Awt.Graphics.DrawImage(
      G,
      Off_Screen_Buffer,
      0,
      0,
      This.I_ImageObserver);
end Update;

-------------------
-- GetAppletInfo --
-------------------

function Getappletinfo (
      This : access Typ )
  return Java.Lang.String.Ref is
begin
   return +("This Connect Four (TM) game was coded in Ada 2005, "
      & "and compiled with the JVM-GNAT compiler");
end Getappletinfo;

-------------------
-- mouseReleased --
-------------------

procedure Mousereleased (
      This : access Typ;
      E    : access Java.Awt.Event.Mouseevent.Typ'Class ) is
   X      : Int                    := Java.Awt.Event.Mouseevent.Getx (E);

   Y      : Int                    := Java.Awt.Event.Mouseevent.Gety (E);

   D      : access Java.Awt.Dimension.Typ'Class := Getsize (This);
   Column,
   Row    : Integer;
begin
   -- need to do this before checking won, since we use
   -- this for user won.
   if Ignore_Turn then
```

```
         return;
      end if;

      if User_Won or Computer_Won or Tie then
         Initialize_Board(Board => Board);
         Computer_Won := False;
         User_Won     := False;
         Tie          := False;
         Ignore_Turn  := False;
         if This.User_Turn then
            This.User_Turn := False;
            Showstatus(This, +"I am thinking...");
            --          Let computer take turn
            Computer_Turn (
               Board  => Board,
               Column => Column);

            --    Place computer disk in the column
            Place_Disk (
               Board  => Board,
               Column => Column,
               Who    => Computer,
               Row    => Row);
         else
            This.User_Turn := True;
         end if;
         Repaint(This);
         Showstatus(This, +
            "Connect Four (TM) by Barry Fagin and Martin Carlisle");
         return;
      end if;


      Showstatus(This, +"I am thinking...");
      --          Let computer take turn
      Computer_Turn (
         Board  => Board,
         Column => Column);

      --    Place computer disk in the column
      Place_Disk (
         Board  => Board,
         Column => Column,
         Who    => Computer,
         Row    => Row);
      --          Check if computer won
      Check_Won (
         Board => Board,
         Who   => Computer,
         Won   => Computer_Won);

      --       Check for a Tie
      Check_Tie (
         Board  => Board,
         Is_Tie => Tie);
      Repaint(This);
      Showstatus(This, +
         "Connect Four (TM) by Barry Fagin and Martin Carlisle");
   end Mousereleased;


   ------------------
   -- mousePressed --
   ------------------

   procedure Mousepressed (
         This : access Typ;
         E    : access Java.Awt.Event.Mouseevent.Typ'Class ) is
      X      : Int                    := Java.Awt.Event.Mouseevent.Getx (E);

      Y      : Int                    := Java.Awt.Event.Mouseevent.Gety (E);

      D      : access Java.Awt.Dimension.Typ'Class := Getsize (This);
```

```
      Column,
      Row    : Integer;
   begin
      -- don't place disk if game over
      if User_Won or Computer_Won or Tie then
         Ignore_Turn := False;
         return;
      end if;

      -- look to see if this is a valid click location
      -- if not, just ignore this click.
      Column := -1;
      for I in Board'range(2) loop
         if X <= Column_Breaks(I) then
            if Board(1,I) = None then
               Column := I;
            end if;
            exit;
         end if;
      end loop;
      if Column <= 0 then
         Ignore_Turn := True;
         return;
      else
         Ignore_Turn := False;
      end if;

      --    Place user disk in the column
      Place_Disk (
         Board  => Board,
         Column => Column,
         Who    => User,
         Row    => Row);
      --          Check if user won
      Check_Won (
         Board => Board,
         Who   => User,
         Won   => User_Won);
      Check_Tie (
         Board  => Board,
         Is_Tie => Tie);
      if User_Won or Tie then
         Ignore_Turn := True;
      else
         Showstatus(This, +"I am thinking...");
      end if;
      Repaint(This);
   end Mousepressed;

   --  The functions below do nothing, but are required to override the ones
   --  defined in the interface we are implementing (when they abstract).
   --  Otherwise, the JVM would complain.

   ------------------
   -- mouseClicked --
   ------------------

   procedure Mouseclicked (
         This : access Typ;
         P1   : access Java.Awt.Event.Mouseevent.Typ'Class ) is
   begin
      null;
   end Mouseclicked;

   ------------------
   -- mouseEntered --
   ------------------

   procedure Mouseentered (
         This : access Typ;
         P1   : access Java.Awt.Event.Mouseevent.Typ'Class ) is
   begin
      null;
```

```
    end Mouseentered;

    -----------------
    -- mouseExited --
    -----------------

    procedure Mouseexited (
          This : access Typ;
          P1   : access Java.Awt.Event.Mouseevent.Typ'Class ) is
    begin
       null;
    end Mouseexited;

end Connectfour;
```