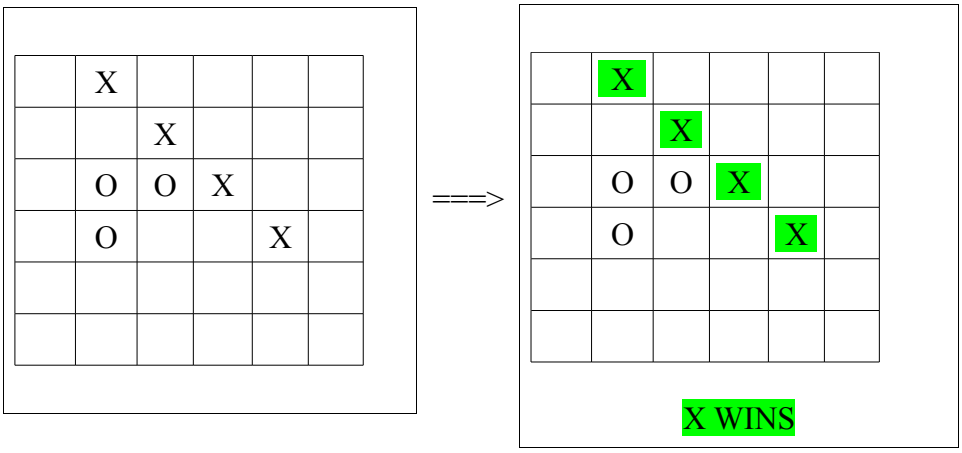
Interface Prototype – What should the program DO?

… In the real project, do this AFTER you have chosen a PROBLEM and INTENDED-USER …
… This is an initial run-through, to get a better impression of what needs to be done ….

Initial Automation for Interface

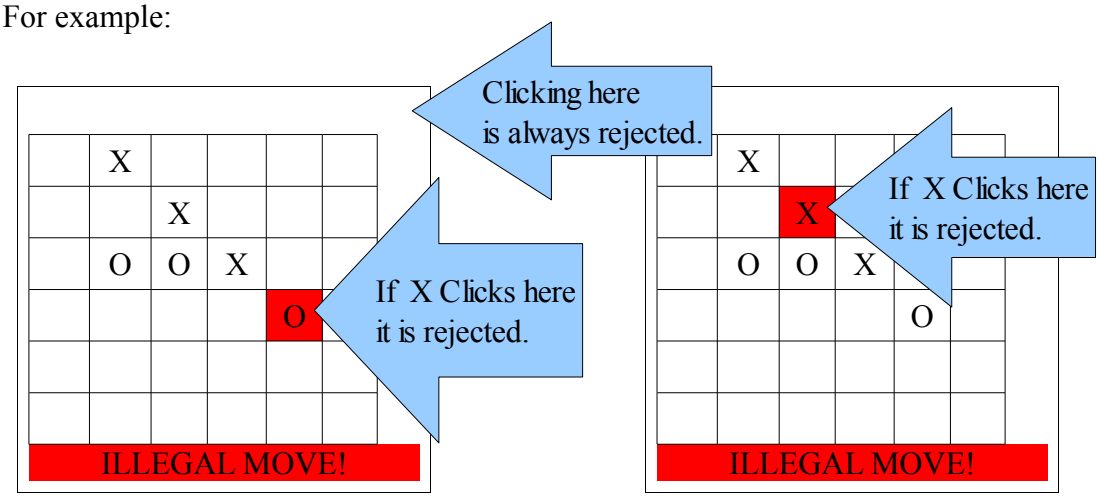
This is a game of tic-tac-toe, played on a 6x6 board.
You win by getting 4 in a row, across or up-and-down or diagonally.
In this example, X wins.



The program should:

- draw the 6x6 board
- accept player input, X or O, by responding to mouse-clicks
- after each click, place the correct letter in the clicked square
- reject forbidden input, like clicking on a square that is not empty
- announce when one of the players wins by getting 4 in a row (above)

For example:



Starting with a game is a good first step, even if you will not actually make a game for your Project. The advantage of a game is that you can do some basic work without talking to an intended-user, so you can get this done in minimum time.

Design – what pieces are needed?

Modules Needed

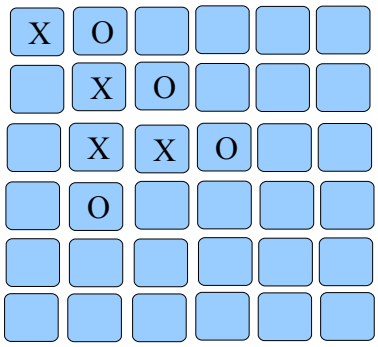
Board = 2D Array

~~~ tildas indicate Methods (algorithms)

~ showBoard
~ action-MouseClicked
~ enterMove – place an X or O in the Board array at the correct square
~ checkLegalMove
~ checkWinner – check for 4 in a row and announce "Winner"
~ checkBoardFull

CurrentPlayer : an int counter that goes 1,2,1,2,1,2,1,2,...
NamePlayerX : String
NamePlayerO : String
GamesWonX : an int counter
GamesWonO : an int counter
GamesTied : an int counter

So the real board will look like this (those boxes represent clickable Buttons):



Coding – HOW can I create it?

Pseudocode for Functional Prototype

Extends GUI - to respond to actions (mouse) and to create Buttons for squares

```
player = "X"
board = new String[6][6]
buttons = new Button[6][6]

makeButtons()      // calculates positions for grid of Buttons
{
   loop  row  from 0 to 5
      loop  col   from  0 to 5
          buttons[row][col] = addButton(“”, col*40,row*40,40,40,this)
       end  loop  col
   end  loop  row
}

actions(Object source, String command)
{  //  should write loops for this //
   if(source == buttons[0][0]){    board[0][0] = player }
   if(source == buttons[0][1]){    board[0][1] = player }
   if(source == buttons[0][2]){    board[0][2] = player }
   …
   if(source == buttons[5][5]){    board[5][5] = player }
  showBoard()
   if( player = "X" ) then      // change to other player
      player = "O"
  else
     player = "X"
}

showBoard()    // copy all X's and O's from board to corresponding buttons
{
   loop  row  from 0 to 5
      loop  col   from  0 to 5
          copy board[row][col]  to  buttons[row][col]
       end  loop  col
  end  loop  row
}

checkWinner()
{  // this will be difficult.  Need to check ALL possible combinations.
   //  Could make it shorter by only checking combinations
   //  that include the move that was just made
}
 …. do the rest of the code for a simple version ….
```

Testing – how will I know what to fix, and know when it's finished?

- Check that it shows the correct number of Buttons
- Click a button and check that the right letter appears
- Click another button and check that the other letter appears (changes players)
- Check that illegal clicks are rejected by clicking an occupied button
- Check that checkWinner works correctly by getting 4 in a row
  - across
  - up/down
  - diagonal and the other diagonal
  - at the edges as well as in the middle

It's useful to complete one module at a time and then test it. So the testing list above outlines the correct order for writing the methods.

==========

The next page shows an initial, very simple prototype. It only records moves (clicks), alternating between players. It does not prevent illegal moves, and it does not check for a winner.
You might find it good practice to add methods for checking illegal moves and check for a winner.

==========

Initial Functional Prototype - only records moves, doesn't check anything

~~~~~~

Click here to download the prototype program.

~~~~~~

```
import java.awt.*;

public class TicTacToe6x6 extends GUI
{
    String[][] board = new String[6][6];
    Button[][] buttons = new Button[6][6];

    String player = "X";

    public TicTacToe6x6()
    {
        super(600,600);
        makeButtons();
    }

    public void actions(Object source, String command)
    {
        for(int row = 0; row < 6; row = row+1)
        {
            for(int col = 0; col < 6; col = col+1)
            {
                if(source == buttons[row][col] && board[row][col]==null)
                {
                    board[row][col] = player;
                    if(player.equals("X"))
                    {  player = "O"; }
                    else
                    {  player = "X"; }
                }
            }
        }
        showBoard();

    }

    public void makeButtons()
    {
        for(int row = 0; row < 6; row = row+1)
        {
            for(int col = 0; col < 6; col = col+1)
            {
                buttons[row][col] = addButton("", col*40, row*40, 40, 40, this);
            }
        }
    }

    public void showBoard()
    {
        for(int row = 0; row < 6; row = row+1)
        {
            for(int col = 0; col < 6; col = col+1)
            {
                if(board[row][col]!=null)
                {  buttons[row][col].setLabel(board[row][col]); }
            }
        }
    }
}
```

First Step Changes

Try to make the following additions and changes in the Prototype program.

(1) Change the program so that the O moves are made automatically by the computer.
The computer must choose a RANDOM square. If that square is occupied,
it must try again, until it randomly chooses and empty square.

(2) After each move, the program needs to check whether a player already has 4-in-a-row.
This is quite tricky. As a first step, check only HORIZONTAL rows.
Start at the most recent move. Check to the left to see whether there is a matching letter.
If so, continue checking to the left until the letter does not match.
Then go back to the right, counting how many of that letter occur in a row.
If you get to 4, it's a win.

(3) Add checking VERTICAL columns.

(4) Add checking DIAGONALS.

(5) Add checking for a full board. This is unlikely, but not impossible.
If there are no empty squares remaining, and nobody has won, then it is a draw.

(6) Make the computer choose "smarter" moves. For example, it can search all over the board
to see whether it has 3 O's in a row, and then choose the 4th spot to complete the win.
If not, it should search for 3 X's in a row. If so, it finds and empty spot at the end of the
3 X's and plays there to block the X player from winning.

Planning for the Future

You might be thinking about some of the following at the same time that you are completing the basic **prototype**. You can keep writing down ideas as they occur to you. But DON'T try to implement these right away – just doing the basic prototype is enough work.

== Further Automation and Flexibility Possibilities ==

- The game must end when the board is full, even though nobody has won.
  This should be automatic – it should not require the players to "kill" the program.
- The computer can keep track of wins and losses over multiple games.
- Pause – save a game in the middle and resume later
- Players take turns playing first.
- The computer can keep a long term record of wins and losses, over many days or weeks.
- Allow betting – e.g. like the doubling cube in Backgammon
- Many players could participate in a TOURNAMENT, where each player plays against each other player, and a winner is determined from the most wins.
- The computer could "help" a player by making suggestions -
  for example, if he already has 3 in a row, suggest the winning move,
  or if the other player has 3 in a row, suggest how to block them.
  Perhaps each player could get 2 "jokers", so they can only ask for help twice in a game.
- The computer could play "intelligently" against a human player,
  so it's not necessary to find an opponent in order to have a game.
- Computer offers to play at various levels of ability -
  (1) make random moves (2) find winning and blocking moves (3) think ahead
- The game could run online so the 2 players can play each other over the Internet.
- The game could be played on various sizes of boards – 5x5, 6x6, 8x8, infinite?
- Keep a record of moves in a game, then replay the game later

You won't succeed in doing ALL of these – you just don't have that much time, and it's not expected. However, you will be assessed on the ***complexity and ingenuity*** *of your solution. The simplest way to achieve this is through lots of* ***Automation and Flexibility*** *features.*