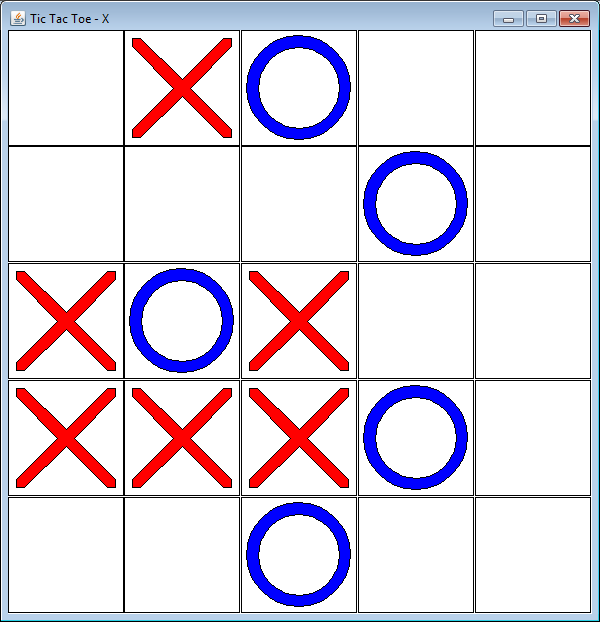
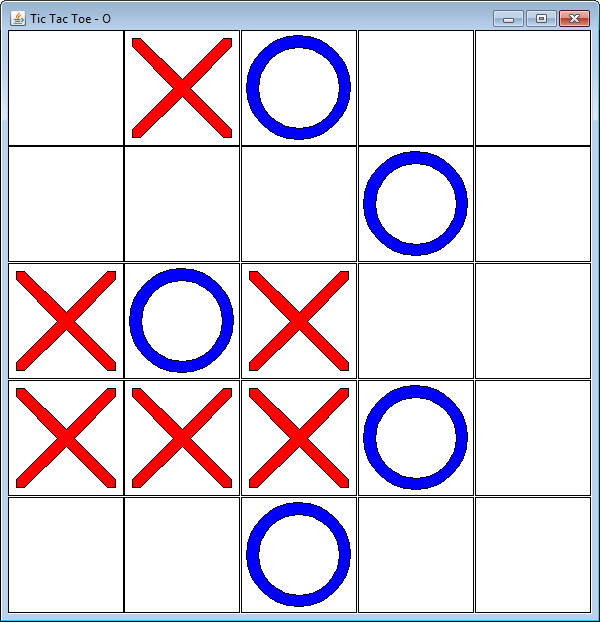
TicTacToeGame Lab

In this lab, you’ll be implementing a Tic-Tac-Toe game. Unlike typical Tic-Tac-Toe, our game will allow variable board sizes. That is, you could play standard TTT on a 3x3 board, or you can increase the size to 4x4 or 5x5 etc. The game will take place in two windows, one for X and one for O.

The two windows are shown above. X’s moves are made in the window on the left (note the X in the window title) and O’s moves are made on the window on the right. Note that the boards show the same position. When X makes a move, his window reflects that move, but so does O’s window (and vice-versa).

First, let’s spend some time designing this program. We’re going to use the CVM design paradigm to help us. In this paradigm, the program is divided into three different parts: Control, View and Model. The job of the Model part is to keep track of all the data in the program as well as to implement the rules of the program to ensure that the data is manipulated only in ways that are consistent with these rules. The View part of the program displays the state of the program. It retrieves all the data from the Model and displays a visual representation on the screen. The Control part of the program handles the processing of user inputs (like mouse clicks and key presses) and other events (like time passage). These inputs are translated into commands which are passed to the Model for processing. In our CVM programs, each of these parts of the program is a separate object (it’s not unusual to combine the Control and View objects into one object, but the functionality of each is kept in separate functions).

Exercise 1 – Model – Data and Access functions

The Model object needs to keep track of the state of the game and provide two sets of functions (approximately, getters and setters) which manipulate the state of the game. We need a class to bind all this code together. Let’s call our class TicTacToeBoard. Using the pictures above, what information do you think class TicTacToeBoard needs to keep track of? What instance variables will TicTacToeBoard need?

1. Add instance variables of the appropriate types to class TicTacToeBoard. (Hint: I had 2 instance variables)
2. Create a constructor for TicTacToeBoard. Since we want to be able to play games of different sizes, the constructor should take one argument indicating the size. Write the body of the constructor so that it initializes its two instance variables appropriately. (Actually, implement the private function reset to handle the initialization and then call reset from the constructor).
3. The TicTacToeBoard class needs to provide getters for each piece of information that it keeps track of. There should be 3 getter functions. Think carefully about why there are 3 instead of 2 when we only have 2 instance variables. Remember that EACH PART of the data needs to be accessible via getters. Also think carefully about the inputs for these getter functions… two of the getter functions shouldn’t have any inputs (like most getter functions) but the third function needs one or more inputs.
4. We also need functions that manipulate the state of the game… we’ll get to those in a later exercise.

Exercise 2 – View part 1

We need a class for the View functionality. This class will be called TicTacToeView. It is already set up for you and will eventually be in charge of displaying the game state. First, within our program, there will be two instances of TicTacToeView… one to paint the window and accept inputs for X and one to paint the window and accept inputs for O. These tasks are very similar so we’ll be using one class and creating two instances. However, each instance needs to know which player it is displaying. TicTacToeView will need an instance variable to keep track of this. Add this instance variable and then write a constructor that takes a value for the instance variable as input.

The View objects will also need to keep track of the window’s dimensions. Add instance variables for these pieces of data. Do NOT add inputs to the constructor since they will be specified later (when paint is called).

Finally, implement the paint method. The paint method consumes a TicTacToeBoard and paints it to the window. It should draw a square for each square in the Board object and, if that square is not empty, then draw either an X or an O in that square. Helper functions for drawing Xs and Os have already been implemented for you. Also, at the start of the paint method, store the width and height of the window in your instance variables.

To test, run launchEmptyOnePlayerGame from the main function.

Exercise 3 – Model – Making a move

The second set of functions for the Model concern making a move and other manipulation and calculations for the state of the game.

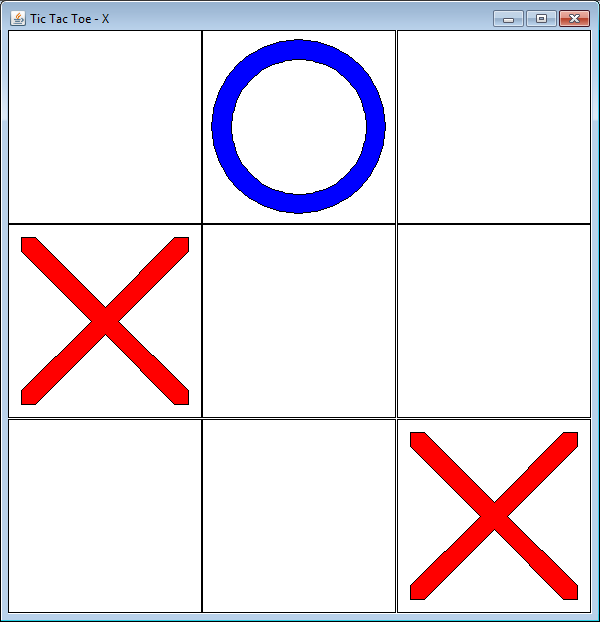
First, the main thing that players can do in a game of TicTacToe is to place their mark on one of the squares of the board. We’ll need a function for that. Let’s call it markSquare. Since the mark can be made in one of several different squares, we need to have inputs which will indicate which square they are marking. Do we need an input for which mark to make? Why or why not?

There are four rules governing the placement of a mark:

1. The mark must go in an empty square.
2. The mark made is the mark of whoever’s turn it is.
3. Once the mark is made, it is the next player’s turn.
4. No marks may be made after the game has finished.

We’ll worry about rules 1-3 now, and 4 later. Implement function markSquare so that, if the indicated square is empty, it marks that square with the current player’s mark and then switches the turn to the other player. If the square is not empty, do nothing.

To test, run function launchDummyOnePlayerGame from the main function. You should see this picture:



Exercise 4 – Control

The control functionality will be part of the view object. Implement function handleMouseClick (instructions are in the function). Rerun launchEmptyOnePlayerGame and click on one of the squares. You should see an X in that square. Click again and make sure that you can’t make another move (it’s now O’s turn and this window is for X).

Exercise 5 – Model – auxiliary functions

The TicTacToeBoard class has several additional functions for checking who wins:

rowWinner

colWinner

mainDiagonalWinner

offDiagonalWinner

isBoardFull

isGameOver

getWinner

getWinningLine

Instructions are in the functions. Implement them now.

Also, change the markSquare function so that no moves can be made if the game is over.

To test this new functionality, you need to implement function launchTwoPlayerGame. In this function, you need to create two view objects (one for X and one for O) and then launch two windows (use function launchTTTWindow which consumes a view and a board). Hint: How many board objects do you need? At this point, you should be able to play the game in two windows.

Exercise 6 – Dumb AI

Class TicTacToeDumbAI is another implementation of the Control/View class. It has a method called chooseAndExecuteMove which consumes a TTTBoard and chooses one of the empty squares at random to play in. Implement this function. To test this exercise, implement launchDumbAIGame to launch two windows – one for a regular player like you did in launchTwoPlayerGame and one using an instance of TicTacToeDumbAI instead of the instance of TicTacToeView.

Exercise 7 – Smart AI (bonus)

Implement class TicTacToeSmartAI so that it plays TicTacToe (somewhat) intelligently. That is, if it has a move which will win, it will take that move. If the other player has a move that will win, it will block that move.