TicTacToe code review
10/21/2017
Tony Gao

Issue 1
The string of conditional statements starting on line 74 of TicTacToe.java ending on line 441 is redundant. The intended behavior is to check which player made the game-winning move and check for win conditions on the board. This behavior does not need to be divided amongst the two players. The code checks for the same winning conditions in the case of each player, which is unnecessary.

Fixing the issue:
A win condition is defined as any 3 of the same player's squares vertically, horizontally, or diagonally. There is enough information provided by checking which player made the last move and then checking if that move caused a winning state on the board for that player. This shortens the conditionals by half the code. The player number can also be flipped in the conditionals at line 68-72, and then the following code would use the player number that played last. After this change the player variable would store the current player, rather than the player that just made the last move, and the conditional would assign the win condition to the player that is not the current player stored by the player variable.

Issue 2
The initialization of the blocks[][] variable starting on line 65 is overly verbose. The intended behavior is to enable or disable the button depending on if there is a player that placed an 'X' or 'O' in the respective square. The buttons are only all disabled in a loop if a win/draw condition is detected.

Fixing the issue:
All of the blocks[][] can be initialized and reset in a loop in a single method that checks all of the blocks and enables or disables the buttons based on if the blocks are populated.

Issue 3
The blocks[][] variable is used as the data container for the players moves. The blocks[][] variable is an array of graphical button elements, which should not be accessed to check for data relevant to the game. This is a violation of the separation of data representation from storage. The blocks[][] variable is also checked 9 times for each block, to resolve the correct index of the button event that occurred. This is unnecessary repetition of code that has the same responsibility.

Fixing the issue:
A separate array should be used to store the player move data that blocks[][] is representing. When a block button is pressed, the array should be updated, triggering an event that updates the button based on the updated data array. A doubly-nested for loop can be used to find the index of the button that is pressed, which can then be used as a variable to determine the [row] [col] coordinates of the button that was pressed.

Issue 4
The logic for checking a win condition, storage of the game data, and presentation of the game data are all centralized in one main class. This violates the principles of Model-View-Controller. MVC dictates that the data storage, representation, and business logic should be separated into their own classes that perform actions and update the objects in response to events that occur from user interaction.

Fixing the issue:
The win condition should be checked in the controller, in its own method. The controller should update the model with relevant game data such as populated game tiles and current player. The graphical view should be updated based on the model, and should only serve as a presentation of the data.