**Connect Five**

This was my favorite game when I was in junior high. I could play this game well and beat people who were much older than me. I sometimes had to intentionally lose one or two games to keep my opponents happy, so they would play more games with me, but that’s then and sadly it’s different now :). Anyway, this project is not about how to win a connect five game; it’s a project for me to learn AngularJS. If you are interested in playing a real connect five game, you can find one [here]( <http://www.i-gamer.net/play/1494.html>). See how many times you can beat the computer!

As I’m going through the O’Reilly book, **AngularJS Up & Running**, I’ve written the code in this project, and I’m continuing to improve it whenever I have time. Other than learning about how to write a module, a controller, a directive with partial html, and a service, the main things I’ve learned in this project are two-way biding, scope, digest cycle, and performance, which are described in detail in later sections.

**Top Level Design**

The connect five game consists of two chip bowls with black and white chips and a game board with, by default, 25 X 40 small squares. Two players, one with black chips and one with white chips, drag and drop chips from the pots to the squares one chip at a time. Each turn the player can place only one chip. The game ends when a player connects five chips on the game board vertically, horizontally, or diagonally. The player who does this first is the winner.

When a chip is placed on the game board, it has a red circle around it to mark it as the most current chip. Players can change their mind, and move the chip with the red circle around it. Once the other player places a chip on the game board and makes their chip the most current chip, the previous most current chip is no longer moveable. This is done because a player may want to use this feature to teach others how to play the game, and show what the difference is when placing a chip in different locations.

When five chips are connected together, either accidently or intentionally, the code detects it and marks five squares containing the five connected chips with blue borders. The message box on the top also animates a winning message from left to right, and changes the text styling from black text to red bold text. During the game, the top message box displays text to indicate which player’s turn it is.

The link “New Game” on the right side of the message box allows the player to start a new game, and returns all the chips back to their pots. The “Demo” link on the far left can auto play a demo game (if you’re too lazy to play :P). The “Demo” link is hidden once it’s clicked to avoid multiple clicks. Starting a new game with the “New Game” link can bring back the “Demo” link. When the first chip is placed on the game board to start playing with drag and drop, it also hides the “Demo” link.

A “Configure” tab is added to allow configuring the game board and number of chips. The number of chips for each pot can be from 10 to 1000 and the dimensions of the game board can be anywhere from 5 x 5 to 25 x 40. By default, there are 100 chips in each pot and a 25 x 40 game board.

**Project Files**

Under the connect\_five directory, there are these HTML and JS files.

* /index.html – Starting page with two tabs, “Play Game” and “Configure”. It uses the “tabs” and “tab” directives to render tabs.
* /partials/play\_game.html – A partial HTML page to display in the “Play Game” tab. It uses ng-repeat to create chips and the game board, and uses the “chipWidget” and “cellWidget” directives to render chips and cells (squares). It also handles the message box with the “msgWidget” directive and the “Demo” link with the “demoWidget” directive. The “New Game” link uses the “ng-click” directive to call controller.newGame().
* /partials/configure.html – A partial HTML page for setting configurable options.
* /controllers/play\_game.js – A controller to create JS arrays and objects for all chips and the game board with an injected service. It provides the newGame() method for the “New Game” link to call; instead of recreating all the chips and the game board, it returns the chips back to their pots and resets cells to their original states to avoid a delay.
* /services/demo\_service.js – A service containing a set of “moves” in an array. It sends “move” events to “cellWidget” and places chips on the game board with a one second delay in between each chip.
* /services/game\_setup\_service.js – A service to create the JS arrays and objects for the entire game board including chip objects in pots, cell objects in the game board, and the message, current chip, and demo objects.
* /services/win\_service.js – A service to be called when a chip is placed on the game board. It checks whether the chip is a winning chip that connects with four other chips.
* /widgets/cell\_chip\_widget.js – A widget containing the “cellWidget” and “chipWidget” directives. The “chipWidget” directive places the chips in their pots, sets the chip CSS styles, and handles the “dragstart” and “resetChipState” events. The “cellWidget” directive sets cell size and handles the “dragover”, “drop”, and “MakeMoveEvent” events, so it can allow placing a chip in a cell with drag and drop, or with an event.
* /widgets/connect\_five\_widget.js – A widget containing the “tabs” and “tab” directives. The “tabs” directive renders and handles tab selections and provides a function for the “tab” directive to add new tab objects. The “tab” directive uses ng-include to include a partial HTML file, and uses ng-show to hide and show the page depending on the tab selection.
* /widgets/demo\_widget.js – A widget to hide and show the “Demo” link with ng-show.
* /widgets/msg\_widget.js – A widget to set message text and its CSS styles for animation.

**Performance Issues**

**Game UI Updates**

This issue is related to the AngularJS two-way biding feature. By default, the connect five game contains 200 HTML DIV elements, one for each chip, and 1000 HTML TD elements, one for each cell. Each chip and cell has its own directive to handle its rendering. For example, marking the most current chip with a red circle, requires updating its CSS styles. If we simply were to call $apply() after each drag and drop, AngularJS would go to the root scope and call the watchers for the entire page to check for changes. In our case there are watchers for 1200+ HTML elements, which impacts performance.

The solution is to save the isolated scope of each chip or cell directive in each chip or cell’s JS object, so that after updating the data of a chip or cell object, it calls scope.$digest() to update its UI. As an example, this solution reduces the number of function calls for setting chip CSS styles from hundreds to two for marking a chip as the most current chip.

This same technique is used in handling the message box, so we don’t need to call $apply() to update the message text after setting the animation CSS style. However, when selecting the “New Game” or “Demo” link, it calls $apply() internally in AngularJS code, so it still goes through all watchers for the entire page. The good thing is that the “Demo” link can only be clicked before making any changes, so it doesn’t need to repetitively call $digest() to reach a stable state and the “New Game” link only needs to deal with as many changes as the players have done to the game board (see the next section).

Here is a document about the digest cycle (Scope Life Cycle)

<https://docs.angularjs.org/guide/scope>

and $digest()

<https://docs.angularjs.org/api/ng/type/$rootScope.Scope#$digest>

**New Game**

The easiest way to implement a function to start a new game is to recreate all data objects including chips, cells and message data objects, and call $apply() which is called internally by the AngularJS code when clicking a link. However, the game recreation performed poorly in IE10; it took a long time to reset back to its original state.

The solution is to move the chips back to their pots by only updating the chip and cell data objects that were changed when placing chips in cells with drag and drop, and sending an event to the “chipWidget” directive to reuse the code that places chips in their pots with random positioning. See gameSetupService.resetBoardAndChips() in game\_setup\_serivce.js and the “resetChipState” handler in cell\_chip\_widget.js for more details.

**Future Work**

* Add the configure tab page.
* Add Jasmine unit tests.
* Make the code mobile friendly.
* Add code to monitor chips on the game board, so it can determine the best move.
* Add code to play with computer.
* Design a way for players to add game rules with simple JS code, so two players can play against each other with their own preset game rules.