<div class="content">

<table class="table table-bordered">

<!--

Throw buttons. Notice the biding to userThrow method (when click)

and throwEnabled (full column)

-->

<thead>

<tr>

<td><button data-bind="click: function(){userThrow(0);},

enable: throwEnabled(0)" type="button" />?</td>

<td><button data-bind="click: function(){userThrow(1);},

enable: throwEnabled(1)" type="button" />?</td>

<td><button data-bind="click: function(){userThrow(2);},

enable: throwEnabled(2)" type="button" />?</td>

<td><button data-bind="click: function(){userThrow(3);},

enable: throwEnabled(3)" type="button" />?</td>

<td><button data-bind="click: function(){userThrow(4);},

enable: throwEnabled(4)" type="button" />?</td>

<td><button data-bind="click: function(){userThrow(5);},

enable: throwEnabled(5)" type="button" />?</td>

<td><button data-bind="click: function(){userThrow(6);},

enable: throwEnabled(6)" type="button" />?</td>

<td><button data-bind="click: function(){userThrow(7);},

enable: throwEnabled(7)" type="button" />?</td>

<td><button data-bind="click: function(){userThrow(8);},

enable: throwEnabled(8)" type="button" />?</td>

</tr>

</thead>

<!--

foreach bind render an array from modelView

For more information, check knockout documentation:

http://knockoutjs.com/documentation/foreach-binding.html

Notice the nested foreach. One for rows and the other one for columns

-->

<tbody data-bind="foreach: { data: board, as: 'row' }">

<tr data-bind="foreach: cols">

<td><span data-bind="text: $data, attr:{'class': $data}" /></td>

</tr>

</tbody>

</table>

<!--

Binding for play again button and winner text when game ends

-->

<input type="button" data-bind="click: newGame,

visible: playAgainVisible" value="Play again" />

<label class="endGame" data-bind="text:winner" />

</div>

<!-- you must link knockout library and the viewmodel after declare DOM binding -->

<script src="knockout-2.1.0.js"></script>

<script src="fourInARowVM.js"></script>

/\*

Knockout ViewModel binding sample with array

To check KnockOut documentation visit: http://knockoutjs.com/documentation/introduction.html

Sample focuses on array binding,check documentation.

http://knockoutjs.com/documentation/foreach-binding.html

\*/

//The model object. File fourInARow.js

//This class defines the structure (board array) and implements the logic (AI) of the game

var fourInARow = new FourInARow();

//The View Model class

function AppViewModel() {

var self = this;

//Define observable properties

//The board: It's an array

self.board = ko.observableArray(fourInARow.newBoard());

//Label to winner text

self.winner = ko.observable();

//We use Ko.computed as a readonly property

self.playAgainVisible = ko.computed(function () { return (self.winner()!=null); }, self);

//self.board() it's a binding property

//To get the array model we need to map it by using utils arrayMap

self.getDashboard = function () {

var dashboard = ko.utils.arrayMap(self.board(), function (item) {

return ({

index: item.index

, cols: ko.utils.arrayMap(item.cols, function (item) {

return (item);

})

});

});

return (dashboard);

};

//Method new game. Clear board and starts new game

self.newGame = function () {

self.board(fourInARow.newBoard());

self.winner(null);

}

//Method userThrow. Called when user click the button of column

self.userThrow = function (column) {

var row = self.throw(column, "O");

if (fourInARow.checkThrowWinner(self.getDashboard(), column, row, "O")) {

self.winner("YOU WIN!!");

return;

}

if (!fourInARow.checkItemsFree(self.getDashboard())) {

self.winner("DRAWN");

return;

}

self.computerThrow();

}

//Method computerThrow. Called from userThrow

self.computerThrow = function () {

var computerThrow = fourInARow.nextComputerThrow(self.getDashboard());

var row = self.throw(computerThrow, "X");

if (fourInARow.checkThrowWinner(self.getDashboard(), computerThrow, row, "X")) {

self.winner("PC WIN");

}

if (!fourInARow.checkItemsFree(self.getDashboard())) {

self.winner("DRAWN");

return;

}

};

//Throw action. It sets the state of the dashboard, then KnockOut refresh DOM

self.throw = function (column, ficha) {

var dashboard = self.getDashboard();

var row = fourInARow.getLastRowFree(dashboard, column);

dashboard[row].cols[column] = ficha;

self.board(dashboard);

return (row);

};

//Throw enabled property to enable/disable column throw

self.throwEnabled = function (column) {

if (self.winner() != null) return (false);

for (var i = 0; i <= (self.getDashboard().length - 1) ; i++) {

if (self.getDashboard()[i].cols[column] == null) {

return (true);

}

}

return (false);

};

}

//Apply binding to KnockOut

ko.applyBindings(new AppViewModel());

/\*

Knockout ViewModel binding sample with array

To check KnockOut documentation visit: http://knockoutjs.com/documentation/introduction.html

Sample focuses on array binding,check documentation.

http://knockoutjs.com/documentation/foreach-binding.html

\*/

//The model object. File fourInARow.js

//This class defines the structure (board array) and implements the logic (AI) of the game

var fourInARow = new FourInARow();

//The View Model class

function AppViewModel() {

var self = this;

//Define observable properties

//The board: It's an array

self.board = ko.observableArray(fourInARow.newBoard());

//Label to winner text

self.winner = ko.observable();

//We use Ko.computed as a readonly property

self.playAgainVisible = ko.computed(function () { return (self.winner()!=null); }, self);

//self.board() it's a binding property

//To get the array model we need to map it by using utils arrayMap

self.getDashboard = function () {

var dashboard = ko.utils.arrayMap(self.board(), function (item) {

return ({

index: item.index

, cols: ko.utils.arrayMap(item.cols, function (item) {

return (item);

})

});

});

return (dashboard);

};

//Method new game. Clear board and starts new game

self.newGame = function () {

self.board(fourInARow.newBoard());

self.winner(null);

}

//Method userThrow. Called when user click the button of column

self.userThrow = function (column) {

var row = self.throw(column, "O");

if (fourInARow.checkThrowWinner(self.getDashboard(), column, row, "O")) {

self.winner("YOU WIN!!");

return;

}

if (!fourInARow.checkItemsFree(self.getDashboard())) {

self.winner("DRAWN");

return;

}

self.computerThrow();

}

//Method computerThrow. Called from userThrow

self.computerThrow = function () {

var computerThrow = fourInARow.nextComputerThrow(self.getDashboard());

var row = self.throw(computerThrow, "X");

if (fourInARow.checkThrowWinner(self.getDashboard(), computerThrow, row, "X")) {

self.winner("PC WIN");

}

if (!fourInARow.checkItemsFree(self.getDashboard())) {

self.winner("DRAWN");

return;

}

};

//Throw action. It sets the state of the dashboard, then KnockOut refresh DOM

self.throw = function (column, ficha) {

var dashboard = self.getDashboard();

var row = fourInARow.getLastRowFree(dashboard, column);

dashboard[row].cols[column] = ficha;

self.board(dashboard);

return (row);

};

//Throw enabled property to enable/disable column throw

self.throwEnabled = function (column) {

if (self.winner() != null) return (false);

for (var i = 0; i <= (self.getDashboard().length - 1) ; i++) {

if (self.getDashboard()[i].cols[column] == null) {

return (true);

}

}

return (false);

};

}

//Apply binding to KnockOut

ko.applyBindings(new AppViewModel());