**blackjack**

**code**

$(document).ready(function(){

console.log('jQuery loaded');

console.log('document ready');

initGame();

});

function initGame() {

console.log('initGame');

// ======= ======= ======= ======= ======= SCREENS ======= ======= ======= ======= =======

// ======= ======= ======= ======= ======= SCREENS ======= ======= ======= ======= =======

// ======= ======= ======= ======= ======= SCREENS ======= ======= ======= ======= =======

var player1\_scr = new Screen("player1", "player");

player1\_scr.borderH = { name:"borderH", type:"bg", iR:1,iC:0,iW:6,iH:1, merge:null,... };

player1\_scr.borderV = { ... };

...

var player2\_scr = new Screen("player2", "player");

var player3\_scr = new Screen("player3", "player");

var dealer\_scr = new Screen("dealer", "player");

var scoreboard\_scr = new Screen();

var gameScreen = new Screen("gameScreen", "game");

gameScreen.orbBtn = { name:"orbBtn", callback:"nextGameScreen", type:"btn", iR:5,iC:12,iW:3,iH:1, merge:"merge", class:"orbBtn", value:"START", tooltipOver:"start the game!", tooltipOut:"" };

...

// ======= ======= ======= ======= ======= OBJECTS ======= ======= ======= ======= =======

// ======= ======= ======= ======= ======= OBJECTS ======= ======= ======= ======= =======

// ======= ======= ======= ======= ======= OBJECTS ======= ======= ======= ======= =======

function Screen(name, type) {

console.log('Screen');

this.name = name;

this.type = type;

}

function Player(id, name) {

console.log('Player');

this.id = id;

this.name = name;

this.state = "inactive";

this.hand = null;

this.score = 0;

this.onesBet = 0;

this.fivesBet = 0;

this.tensBet = 0;

this.onesBank = 20;

this.fivesBank = 30;

this.tensBank = 50;

}

function Game(whichGame) {

console.log('Game');

this.name = whichGame;

this.deckArray = [];

this.deckPointsArray = [];

this.currentPlayer = 0;

this.currentScreen = null;

this.playerNamesArray = [];

this.playerObjectsArray = [player1, player2, player3, dealer];

this.screenNamesArray = ["splash", "nameEnter", "enterPlay", "playGame", "doTheMath"];

this.screenObjectsArray = [];

this.subScreenNamesArray = ["player1\_scr", "player2\_scr", "player3\_scr", "dealer\_scr", "scoreboard\_scr"];

this.subcreenObjectsArray = [player1\_scr, player2\_scr, player3\_scr, dealer\_scr, scoreboard\_scr];

this.onesBet = 0;

this.fivesBet = 0;

this.tensBet = 0;

}

function Display(whichDisplay) {

console.log('Display');

this.name = whichDisplay;

this.rowSpansArray = null;

this.playerStateItems = {

inactive: ["borderH","borderV","pName","pScore","pBank"],

active: ["borderH","borderV","pName","pScore","pBank"],

placeBets: ["borderH","borderV","pName","pScore","pBank","pBet","pBet\_1s","pBet\_5s","pBet\_10s","pBetOnes","pBetFives","pBetTens","chips1s","chips5s","chips10s"],

hitMeHoldMe: ["borderH","borderV","pName","pScore","pBank","pBet","pBet\_1s","pBet\_5s","pBet\_10s","pBetOnes","pBetFives","pBetTens","hitMeBtn","holdMeBtn"],

turnOver: ["borderH","borderV","pName","pScore","pBank","pBet","pBet\_1s","pBet\_5s","pBet\_10s","pBetOnes","pBetFives","pBetTens"],

handOver: ["borderH","borderV","pName","pScore","pBank"]

};

this.gameStateItems = {

splash: ["orbBtn","tooltips"],

nameEnter: ["enterBtn","playerName","tooltips"],

enterPlay: ["enterBtn","startBtn","playerName","tooltips"],

playGame: ["playBtn","tooltips"],

doTheMath: ["playAgainBtn","newGameBtn","tooltips"]

};

}

// ======= ======= ======= ======= ======= DISPLAY ======= ======= ======= ======= =======

// ======= ======= ======= ======= ======= DISPLAY ======= ======= ======= ======= =======

// ======= ======= ======= ======= ======= DISPLAY ======= ======= ======= ======= =======

// ======= ======= ======= nextGameScreen ======= ======= =======

Display.prototype.nextGameScreen = function() {

console.log("");

console.log("nextGameScreen");

console.log("== PREV gameScreen == " + game.currentScreen);

// identify where we are in the game (prev screen/next screen/selected screen)

// use current screen name to identify current screen index

var prevItemNames, nextItemNames, nextScreenIndex;

var REMprevItems = [];

var ADDnextItems = [];

// == game start initialize first screen ("splash")

if (game.currentScreen == null) {

game.currentScreen = "splash";

prevItemNames = [];

nextItemNames = display.gameStateItems[game.currentScreen];

// increment screen index to get next screen object

} else {

nextScreenIndex = game.screenNamesArray.indexOf(game.currentScreen) + 1;

nextGameScreenName = game.screenNamesArray[nextScreenIndex];

prevItemNames = display.gameStateItems[game.currentScreen];

nextItemNames = display.gameStateItems[game.screenNamesArray[nextScreenIndex]];

game.currentScreen = nextGameScreenName;

}

console.log("== NEXT gameScreen == " + game.currentScreen);

// == identify items to add or remove (by comparison of NAME strings)

var REMprevNames = $(prevItemNames).not(nextItemNames).get();

var ADDnextNames = $(nextItemNames).not(prevItemNames).get();

// == get screen item OBJECTS from lists of screen item NAMES

if (prevItemNames) {

for (var i = 0; i < prevItemNames.length; i++) {

prevItemName = prevItemNames[i];

var found = $.inArray(prevItemName, REMprevNames)

if (found > -1) {

prevItem = gameScreen[prevItemName];

REMprevItems.push(prevItem);

}

}

}

for (var i = 0; i < nextItemNames.length; i++) {

nextItemName = nextItemNames[i];

var found = $.inArray(nextItemName, ADDnextNames)

if (found > -1) {

nextItem = gameScreen[nextItemName];

ADDnextItems.push(nextItem);

}

}

// == send screen data to screen building functions

this.addRemoveScreenItems(REMprevItems, ADDnextItems);

}

// ======= ======= ======= nextSubscreen ======= ======= =======

Display.prototype.nextSubscreen = function(playerIndex) {

console.log("nextSubscreen");

var nextSubscreenItems = [];

currentPlayer = game.playerObjectsArray[playerIndex - 1];

currentPlayerScreen = game.subcreenObjectsArray[playerIndex - 1];

// == check for special case of dealer

if (playerIndex == 4) {

playerIndex = "D";

}

// == game start initialize first screen ("splash")

if (currentPlayer.state == null) {

currentPlayer.state = "inactive";

}

if (currentPlayer.name == "dealer") {

subscreenItemNames = ["borderH","borderV","pName","pScore"];

} else {

subscreenItemNames = this.playerStateItems[currentPlayer.state];

}

for (var i = 0; i < subscreenItemNames.length; i++) {

nextItemName = subscreenItemNames[i];

nextItemObject = currentPlayerScreen[nextItemName];

nextSubscreenItems.push(nextItemObject);

}

// == put items from current player state onto screen

for (var j = 0; j < nextSubscreenItems.length; j++) {

nextItem = nextSubscreenItems[j];

nextType = nextItem.type;

indexCell = display.modifyGridRegion(nextItem, playerIndex);

if (nextType == "input") {

newTextInput = "<input id='" + nextItem.name + "Input' class='" + nextIte...

$(indexCell).append(newTextInput);

$(newTextInput).attr("id", nextItem.name);

}

if ((nextType == "btn") || (nextType == "slider")) {

display.activateNextItem(nextItem, indexCell);

}

}

}

// ======= ======= ======= updateSubscreen ======= ======= =======

Display.prototype.updateSubscreen = function(playerIndex, nextPlayerState) {

console.log("");

console.log("updateSubscreen");

var prevItemNames, nextItemNames, nextScreenIndex;

var REMprevItems = [];

var ADDnextItems = [];

var whichPlayer = game.playerObjectsArray[playerIndex - 1];

var whichPlayerScreen = game.subcreenObjectsArray[playerIndex - 1];

var prevPlayerState = whichPlayer.state;

prevItemNames = this.playerStateItems[prevPlayerState];

whichPlayer.state = nextPlayerState;

nextItemNames = this.playerStateItems[nextPlayerState];

// == identify items to add or remove (by comparison of NAME strings)

var REMprevNames = $(prevItemNames).not(nextItemNames).get();

var ADDnextNames = $(nextItemNames).not(prevItemNames).get();

// == get screen item OBJECTS from lists of screen item NAMES

if (prevItemNames) {

for (var i = 0; i < prevItemNames.length; i++) {

prevItemName = prevItemNames[i];

var found = $.inArray(prevItemName, REMprevNames)

if (found > -1) {

prevItem = whichPlayerScreen[prevItemName];

REMprevItems.push(prevItem);

}

}

}

for (var i = 0; i < nextItemNames.length; i++) {

nextItemName = nextItemNames[i];

var found = $.inArray(nextItemName, ADDnextNames)

if (found > -1) {

nextItem = whichPlayerScreen[nextItemName];

ADDnextItems.push(nextItem);

}

}

// == send screen data to screen building functions

this.addRemoveScreenItems(REMprevItems, ADDnextItems, playerIndex);

}

// ======= ======= ======= addRemoveScreenItems ======= ======= =======

Display.prototype.addRemoveScreenItems = function(removeItemsArray, addItemsArray, playerIndex) {

console.log("addRemoveScreenItems");

var nextItem, nextType, indexCell, newTextInput;

// == remove prev items

if (removeItemsArray) {

for (var j = 0; j < removeItemsArray.length; j++) {

nextItem = removeItemsArray[j];

nextType = nextItem.type;

if ((nextType == "btn") || (nextType == "slider")) {

display.deActivateNextItem(nextItem, indexCell);

}

display.unModifyGridRegion(nextItem);

}

}

// == add new items and append child elements if any

for (var j = 0; j < addItemsArray.length; j++) {

nextItem = addItemsArray[j];

nextType = nextItem.type;

indexCell = display.modifyGridRegion(nextItem, playerIndex);

// append child elements (inputs, sliders)

if (nextType == "input") {

newTextInput = "<input id='" + nextItem.name + "Input' class='" + nextIt...

$(indexCell).append(newTextInput);

$(newTextInput).attr("id", nextItem.name);

}

if (nextType == "slider") {

var sliderLoc = $("#" + nextItem.name + "\_" + playerIndex).offset();

var locX = sliderLoc.left + 80;

var locY = sliderLoc.top + 15;

var sliderId = nextItem.name + "\_slider" + "\_" + playerIndex;

sliderString = "<div id='" + sliderId + "' class='slider " + nextItem.cl...

$(indexCell).append(sliderString);

$("#" + sliderId).css("left", locX);

$("#" + sliderId).css("top", locY);

}

// == activate interacive elements (buttons/sliders)

if ((nextType == "btn") || (nextType == "slider")) {

display.activateNextItem(nextItem, indexCell);

}

}

}

// ======= ======= ======= updatePlayerScreenData ======= ======= =======

Display.prototype.updatePlayerScreenData = function(whichPlayerScreen, whichPlayer) {

console.log("updatePlayerScreenData");

if (whichPlayer.id < 4) {

var totalBank = whichPlayer.onesBank + whichPlayer.fivesBank + whichPlayer.tensBank;

var totalBet = whichPlayer.onesBet + whichPlayer.fivesBet + whichPlayer.tensBet;

$("#pScore\_" + whichPlayer.id).text(whichPlayer.score);

$("#pBetOnes\_" + whichPlayer.id).text("$" + whichPlayer.onesBank);

$("#pBetFives\_" + whichPlayer.id).text("$" + whichPlayer.fivesBank);

$("#pBetTens\_" + whichPlayer.id).text("$" + whichPlayer.tensBank);

$("#pBet\_1s\_" + whichPlayer.id).text("$" + whichPlayer.onesBet);

$("#pBet\_5s\_" + whichPlayer.id).text("$" + whichPlayer.fivesBet);

$("#pBet\_10s\_" + whichPlayer.id).text("$" + whichPlayer.tensBet);

$("#pBank\_" + whichPlayer.id).text("$" + totalBank);

$("#pBet\_" + whichPlayer.id).text("$" + totalBet);

} else {

whichPlayer.id = "D";

$("#pScore\_" + whichPlayer.id).text(whichPlayer.score);

}

}

// ======= ======= ======= ======= ======= GRID ======= ======= ======= ======= =======

// ======= ======= ======= ======= ======= GRID ======= ======= ======= ======= =======

// ======= ======= ======= ======= ======= GRID ======= ======= ======= ======= =======

// ======= ======= ======= modifyGridRegion ======= ======= =======

Display.prototype.modifyGridRegion = function(whichItem, playerIndex, offsetR, offsetC) {

console.log("modifyGridRegion: " + whichItem.name);

if (playerIndex) {

playerIndex = "\_" + playerIndex;

} else {

playerIndex = "";

}

if (!offsetR) { offsetR = 0 };

if (!offsetC) { offsetC = 0 };

var tableRows = $("tr");

var regionType = whichItem.merge;

var loopCount = 0

// == remove cells from merge area (check row/colspans in each row)

for (var row = 0; row < whichItem.iH; row++) {

loopCount++;

nextRow = whichItem.iR + offsetR + row;

nextCol = whichItem.iC + offsetC;

nextRowObject = tableRows[nextRow];

colspans = this.checkColumnSpans(nextRowObject, nextRow, nextCol);

rowspans = this.checkRowSpans(nextRow, nextCol);

temp\_iC = nextCol - colspans - rowspans;

if (regionType == "merge") {

for (var col = 0; col < whichItem.iW; col++) {

if (((row == 0) && (col == 1))) {

totalColOffset = temp\_iC + col;

}

if (row > 0) {

totalColOffset = temp\_iC;

display.toggleRowspans(whichItem, offsetR, offsetC, "on")

}

// remove all but index cell in merge area

if ((row == 0) && (col == 0)) {

indexCell = $(nextRowObject).children()[temp\_iC];

$(indexCell).attr("colSpan", whichItem.iW);

$(indexCell).attr("rowSpan", whichItem.iH);

$(indexCell).addClass(whichItem.class);

if (whichItem.type != "input") {

$(indexCell).attr("id", whichItem.name + playerIndex);

}

} else {

nextCell = $(nextRowObject).children()[totalColOffset];

$(nextCell).remove();

}

}

// == modify item class via item class parameters

} else {

for (var col = 0; col < whichItem.iW; col++) {

nextCell = $(nextRowObject).children()[temp\_iC + col];

$(nextCell).removeClass();

$(nextCell).addClass(whichItem.class);

if ((row == 0) && (col == 0)) {

indexCell = $(nextRowObject).children()[temp\_iC];

if (whichItem.image != null) {

newImage = $(new Image()).attr('src', "images/" + whi...

$(newImage).attr("id", whichItem.name);

} else {

$(indexCell).attr("id", whichItem.name);

}

$(indexCell).attr("id", whichItem.name + playerIndex);

} else {

$(nextCell).attr("id", (nextRow) + "-" + (nextCol + col));

}

}

}

}

return indexCell;

}

// ======= ======= ======= unModifyGridRegion ======= ======= =======

Display.prototype.unModifyGridRegion = function(whichItem, offsetR, offsetC) {

console.log("unModifyGridRegion: " + whichItem.name);

if (!offsetR) { offsetR = 0 };

if (!offsetC) { offsetC = 0 };

var tableRows = $("tr");

var regionType = whichItem.merge;

console.log(" regionType: " + regionType);

// == remove cells from merge area (check row/colspans in each row)

for (var row = 0; row < whichItem.iH; row++) {

nextRow = whichItem.iR + offsetR + row;

nextCol = whichItem.iC + offsetC;

nextRowObject = tableRows[nextRow];

colspans = this.checkColumnSpans(nextRowObject, nextRow, nextCol);

rowspans = this.checkRowSpans(nextRow, nextCol);

temp\_iC = nextCol - colspans - rowspans;

if (regionType == "merge") {

for (var col = 0; col < (whichItem.iW); col++) {

if ((row == 0) && (col == 0)) {

indexCell = $(nextRowObject).children()[temp\_iC];

$(indexCell).remove();

}

indexRowCell = $(nextRowObject).children()[temp\_iC - 1];

for (var col = 0; col < whichItem.iW; col++) {

var newCell = document.createElement("td");

$(indexRowCell).after(newCell);

$(newCell).addClass("cell");

$(newCell).attr("id", (nextRow) + "-" + (nextCol + col));

}

if (row > 0) {

display.toggleRowspans(whichItem, offsetR, offsetC, "off")

}

}

} else {

for (var col = 0; col < whichItem.iW; col++) {

nextCell = $(nextRowObject).children()[temp\_iC + col];

if (whichItem.type == "slider") {

$(nextCell).empty();

} else {

if (whichItem.type == "btn") {

$(nextCell).empty();

}

$(nextCell).removeClass();

$(nextCell).addClass("cell");

$(nextCell).attr("id", (nextRow) + "-" + (temp\_iC + col));

}

}

}

}

}

// ======= ======= ======= activateNextItem ======= ======= =======

Display.prototype.activateNextItem = function(whichItem, indexCell) {

console.log("activateNextItem");

var self = this;

var indexElement, tooltip;

var whichAction = whichItem.callback;

// ======= tooltips =======

$(indexCell).off("mouseenter").on("mouseenter", function(event){

// console.log("-- mouseenter");

indexElement = event.target;

tooltip = display.getTooltip(indexElement, "over");

});

$(indexCell).off("mouseout").on("mouseout", function(event){

// console.log("-- mouseout");

indexElement = event.target;

tooltip = display.getTooltip(indexElement, "out");

});

// ======= general =======

switch(whichAction) {

case "nextGameScreen":

$(indexCell).off("click").on("click", function(){

console.log("");

console.log("-- -- -- nextGameScreen -- -- -- ");

self.nextGameScreen();

$("#tooltips").text("enter player name and click ENTER");

});

break;

case "saveNewPlayer":

$(indexCell).off("click").on("click", function(){

console.log("");

console.log("-- -- -- saveNewPlayer -- -- -- ");

game.saveNewPlayer();

if (game.playerNamesArray.length < 3) {

$("#tooltips").text("enter another player name or click START to begin");

}

});

break;

case "startGame":

$(indexCell).off("click").on("click", function(){

console.log("");

console.log("-- -- -- startGame -- -- -- ");

game.startGame();

$("#tooltips").text("");

});

break;

case "mngBets":

$(indexCell).off("mousedown").on("mousedown", function(){

console.log("-- -- -- -- -- mngBets -- -- -- -- -- ");

display.moveSlider(event);

});

break;

case "playGame":

$(indexCell).off("click").on("click", function(){

console.log("");

console.log("-- -- -- playGame -- -- -- ");

game.playGame();

$("#tooltips").text("");

});

break;

case "hitMe":

$(indexCell).off("click").on("click", function(){

console.log("");

console.log("-- -- -- hitMe -- -- -- ");

game.hitMe();

});

break;

case "holdMe":

$(indexCell).off("click").on("click", function(){

console.log("");

console.log("-- -- -- holdMe -- -- -- ");

game.holdMe();

});

break;

}

}

// ======= ======= ======= deActivateNextItem ======= ======= =======

Display.prototype.deActivateNextItem = function(whichItem, indexCell) {

// console.log("deActivateNextItem: " + whichItem.name);

}

// ======= ======= ======= checkRowSpans ======= ======= =======

Display.prototype.checkRowSpans = function(whichRow, whichCol) {

// console.log("checkRowSpans");

var rowspans = 0;

var indexRow = 0;

for (var col = 0; col < 18; col++) {

if (col < whichCol) {

rowspanSpanObject = this.rowSpansArray[whichRow][col];

if (rowspanSpanObject.rspan == true) {

rowspans++;

}

}

}

return rowspans;

}

// ======= ======= ======= checkColumnSpans ======= ======= =======

Display.prototype.checkColumnSpans = function(whichRowObject, whichRow, whichCol) {

// console.log("checkColumnSpans");

var colspans = 0;

var indexCol = 0;

for (var col = 0; col < $(whichRowObject).children().length; col++) {

nextColumnObject = $(whichRowObject).children()[col];

nextColspan = $(nextColumnObject).attr('colSpan');

nextColId = $(nextColumnObject).attr('id');

if ((nextColspan > 1) && (col < (whichCol - colspans))) {

colspans += nextColspan - 1;

}

}

return colspans;

}

// ======= ======= ======= toggleRowspans ======= ======= =======

Display.prototype.toggleRowspans = function(whichItem, offsetR, offsetC, onOff) {

// console.log("toggleRowspans");

if (whichItem.iH > 1) {

for (var row = 0; row < whichItem.iH; row++) {

for (var col = 0; col < whichItem.iW; col++) {

if (row != 0) {

tableRow = whichItem.iR + offsetR + row;

tableCol = whichItem.iC + offsetC + col;

if (onOff == "on") {

this.rowSpansArray[tableRow][tableCol].rspan = true;

} else {

this.rowSpansArray[tableRow][tableCol].rspan = false;

}

}

}

}

}

}

// ======= ======= ======= reportRowspans ======= ======= =======

Display.prototype.reportRowspans = function(whichRow) {

console.log("reportRowspans ------- ------- row: " + whichRow);

targetRow = 7;

if (whichRow == targetRow) {

for (var row = 0; row < this.tableRowspansArray.length; row++) {

nextRow = this.tableRowspansArray[row];

if (row == whichRow) {

console.log(" nextRow.length: " + nextRow.length);

for (var col = 0; col < nextRow.length; col++) {

nextCol = nextRow[col];

if (nextCol.rspan == true) {

console.log(" rowspan TRUE: " + row + "/" + col);

} else {

console.log(" rowspan FALSE: " + row + "/" + col);

}

}

}

}

}

}

// ======= ======= ======= initRowSpans ======= ======= =======

Display.prototype.initRowSpans = function() {

console.log("initRowSpans");

var nextRow, nextCol, tableCol, cellRC;

var rowSpansArray = [];

var tableRows = $(".row");

for (var row = 0; row < tableRows.length; row++) {

nextRow = tableRows[row];

nextRowArray = [];

tableCols = $(nextRow).children(".cell");

for (var col = 0; col < tableCols.length; col++) {

nextCol = tableCols[col];

$(nextCol).attr("id", row + "-" + col);

cellRC = { R:row, C:col, rspan:false };

nextRowArray.push(cellRC);

}

rowSpansArray.push(nextRowArray);

}

this.rowSpansArray = rowSpansArray;

}

// ======= ======= ======= getTooltip ======= ======= =======

Display.prototype.getTooltip = function(whichElement, overOut) {

console.log("getTooltip");

// $("#tooltips").text(nextTooltip);

}

// ======= ======= ======= moveSlider ======= ======= =======

Display.prototype.moveSlider = function(event) {

console.log("-- moveSlider --");

var dX, dY;

var maxY = 100;

var minY = 100;

var updateChips;

var triggerFlag = true;

var whichSlider = event.target;

// ======= selected slider object ======= ======= ======= ======= =======

var slider = {

slider\_id: whichSlider.id,

slider\_element: whichSlider,

player\_chips: whichSlider.className.split(/\s+/)[1],

player\_index: parseInt(whichSlider.id.charAt(whichSlider.id.length-1) - 1),

start\_mouseX: 0,

start\_elementX: 0,

drag\_elementX: 0,

zLevel: 3,

self: this,

// ======= MOUSE\_DOWN ======= MOUSE\_DOWN ======= MOUSE\_DOWN ======= MOUSE\_DOWN =======

initSlider: function (newEvent) {

event.preventDefault();

var evt = newEvent || window.event;

this.start\_mouseX = newEvent.clientX;

this.start\_elementX = this.slider\_element.offsetLeft;

removeEventSimple(window.document, 'mousedown', display.moveSlider);

addEventSimple(window.document, 'mousemove', slider.dragSlider);

addEventSimple(window.document, 'mouseup', slider.dropSlider);

},

// ======= MOUSE\_MOVE ======= MOUSE\_MOVE ======= MOUSE\_MOVE ======= MOUSE\_MOVE =======

dragSlider: function (newEvent) {

// console.log('dragSlider');

var evt = newEvent || window.event;

dX = parseInt(evt.clientX) - parseInt(slider.start\_mouseX);

if (dX < -20) {

dX = -20;

}

if (dX > 20) {

dX = 20;

}

slider.updateSlider(dX);

if (triggerFlag == true) {

slider.updateSvgs(dX);

triggerFlag = false;

}

},

updateSlider: function (dX) {

// console.log('updateSlider');

slider.slider\_element.style.left = slider.start\_elementX + dX + 'px';

slider.drag\_elementX = slider.start\_elementX + dX;

},

updateSvgs: function (dX) {

// console.log('updateSvgs');

var currentChips = slider.player\_chips;

var player\_object = game.playerObjectsArray[this.player\_index];

var player\_screen = game.subcreenObjectsArray[this.player\_index];

if (dX > 0) {

updateChips = setInterval(function() {

console.log(" dX+: " + dX);

game.placeBet(currentChips, player\_object);

display.updatePlayerScreenData(player\_screen, player\_object);

}, 300);

} else {

updateChips = setInterval(function() {

console.log(" dX-: " + dX);

game.returnBet(currentChips, player\_object);

display.updatePlayerScreenData(player\_screen, player\_object);

}, 300);

}

},

// ======= MOUSE\_UP ======= MOUSE\_UP ======= MOUSE\_UP ======= MOUSE\_UP =======

dropSlider: function(newEvent) {

// console.log('dropSlider');

clearInterval(updateChips);

slider.slider\_element.style.left = slider.start\_elementX + "px";

removeEventSimple(window.document, 'mouseup', slider.dropSlider);

removeEventSimple(window.document, 'mousemove', slider.dragSlider);

}

}

// ======= events ======= ======= ======= ======= ======= ======= =======

function addEventSimple(obj,evt,fn) {

// console.log('addEventSimple');

if (obj.addEventListener)

obj.addEventListener(evt,fn,false);

else if (obj.attachEvent)

obj.attachEvent('on'+evt,fn);

}

function removeEventSimple(obj,evt,fn) {

// console.log('removeEventSimple');

if (obj.removeEventListener)

obj.removeEventListener(evt,fn,false);

else if (obj.detachEvent)

obj.detachEvent('on'+evt,fn);

}

slider.initSlider(event);

return slider;

}

// ======= ======= ======= ======= ======= GAME ======= ======= ======= ======= =======

// ======= ======= ======= ======= ======= GAME ======= ======= ======= ======= =======

// ======= ======= ======= ======= ======= GAME ======= ======= ======= ======= =======

// ======= ======= ======= saveNewPlayer ======= ======= =======

Game.prototype.saveNewPlayer = function() {

console.log("saveNewPlayer");

var playerName, playerCount, playerState, newPlayer, playerNameCell;

playerName = $('#playerNameInput').val();

if (playerName) {

playerCount = this.playerNamesArray.length;

if (playerCount < 3) {

// == get pre-made player object and set player name on it

newPlayer = this.playerObjectsArray[playerCount];

newPlayer.name = playerName;

// == player names array tracks actual players in current game

this.playerNamesArray.push(newPlayer.name);

playerCount = this.playerNamesArray.length;

display.nextSubscreen(playerCount);

playerNameCell = "#pName\_" + newPlayer.id;

$(playerNameCell).text(newPlayer.name);

$("#tooltips").text("Enter another name or press 'start'");

}

if (playerCount == 1) {

// == activates play game button if no more players to enter

display.nextGameScreen();

}

if (playerCount == 3) {

// == max of 3 players; advance to game start

$("#tooltips").text("Max of 3 players. Start game!");

game.startGame();

}

$("#playerNameInput").val("");

} else {

$("#tooltips").text("Enter name or press 'start'");

}

}

// ======= ======= ======= startGame ======= ======= =======

Game.prototype.startGame = function() {

console.log("startGame");

display.nextSubscreen(4); // create dealer screen elements

playerNameCell = "#pName\_D";

$(playerNameCell).text("dealer");

display.nextGameScreen();

for (var i = 0; i < this.playerNamesArray.length; i++) {

nextPlayer = this.playerObjectsArray[i];

nextPlayerScreen = this.subcreenObjectsArray[i];

nextPlayerId = nextPlayer.id;

display.updateSubscreen(nextPlayerId, "placeBets");

}

this.currentPlayer = game.playerObjectsArray[0];

game.dealCards();

}

// ======= ======= ======= dealCards ======= ======= =======

Game.prototype.dealCards = function(indexCell, whichAction) {

console.log("");

console.log("dealCards");

var winnersArray = [];

var dealerScreen = this.subcreenObjectsArray[3];

var nextPlayer, nextSuit, nextValue, nextPoints, cardPointsArray, nextCard, nextPoints;

// ======= initialize deck

var suitArray = ['C','D','H','S'];

var valueArray = ['A','2','3','4','5','6','7','8','9','10','J','Q','K'];

var pointsArray = [11, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10];

var tableString, nextValue, nextPoints, cardPoints, nextCard, nextPlayer;

for (var i = 0; i < suitArray.length; i++) {

nextSuit = suitArray[i];

for (var j = 0; j < valueArray.length; j++) {

nextValue = valueArray[j];

nextPoints = pointsArray[j];

this.deckArray.push(nextValue + nextSuit);

this.deckPointsArray.push(nextPoints);

}

}

// ======= clear previous player hands

for (var i = 0; i < (this.playerNamesArray.length); i++) {

nextPlayer = this.playerObjectsArray[i];

nextPlayer.hand = [];

}

// ======= deal cards to each player and dealer

for (var i = 0; i < (this.playerNamesArray.length); i++) {

nextPlayer = this.playerObjectsArray[i];

nextPlayerScreen = this.subcreenObjectsArray[i];

// ======= getNextCard

for (var j = 0; j < 2; j++) {

cardPointsArray = game.getNextCard(); // get card from deck; shrink deck

nextCard = cardPointsArray[0];

nextPoints = cardPointsArray[1];

nextPlayer.hand.push(nextCard);

nextPlayer.score = nextPlayer.score + nextPoints; // calculate player score

this.displayNextCard(nextPlayer, nextPlayerScreen);

}

// ======= if Ace card and > 21 (2 aces)

if (nextPlayer.score > 21) {

console.log(' 2 aces: ' + nextPlayer.score);

for (var k = 0; k < nextPlayer.hand.length; k++) {

nextCard = nextPlayer.hand[k];

// ======= change Ace value to 1

if (nextCard.indexOf("A") > 0) {

nextPlayer.score = nextPlayer.score - 10;

console.log(' found ace: ' + nextPlayer.score);

break;

}

}

}

// ======= instant winner

if (nextPlayer.score == 21) {

winnersArray.push(nextPlayer);

}

display.updatePlayerScreenData(nextPlayerScreen, nextPlayer);

$("#tooltips").text("Click PLAY button to place bets");

}

// ======= deal to dealer

dealer.hand = []

for (var j = 0; j < 2; j++) {

nextCard = cardPointsArray[0];

nextPoints = cardPointsArray[1];

dealer.hand.push(nextCard);

dealer.score = dealer.score + nextPoints;

this.displayNextCard(dealer, dealerScreen);

}

display.updatePlayerScreenData(dealerScreen, dealer);

this.flipCards();

}

// ======= ======= ======= getNextCard ======= ======= =======

Game.prototype.getNextCard = function() {

// console.log("getNextCard");

var cardIndex = parseInt(Math.random() \* this.deckArray.length);

var nextCard = this.deckArray[cardIndex];

var nextPoints = this.deckPointsArray[cardIndex];

this.deckArray.splice(cardIndex, 1);

this.deckPointsArray.splice(cardIndex, 1);

return [nextCard, nextPoints];

}

// ======= ======= ======= displayNextCard ======= ======= =======

Game.prototype.displayNextCard = function(whichPlayer, whichSubscreen) {

console.log("displayNextCard");

var whichMerge, cardDivString;

var playerIndex = whichPlayer.id;

// == get player card object for positioning

whichCardObject = whichSubscreen.pCards;

whichClass = whichCardObject.class;

var whichName = whichCardObject.name;

var cardCount = whichPlayer.hand.length;

var cardValue = whichPlayer.hand[cardCount - 1];

var whichSuit = cardValue.substr(cardValue.length - 1);

var whichValue = cardValue.substring(0, cardValue.length - 1);

switch(whichSuit) {

case "C":

imageString = "<img src='images/clubs.png' alt='clubs'>";

break;

case "D":

imageString = "<img src='images/diamonds.png' alt='diamonds'>";

break;

case "H":

imageString = "<img src='images/hearts.png' alt='hearts'>";

break;

case "S":

imageString = "<img src='images/spades.png' alt='spades'>";

break;

}

if (whichPlayer.name == "dealer") {

offsetC = cardCount - 1;

} else {

offsetC = -(cardCount - 1);

}

offsetR = 0;

indexCell = display.modifyGridRegion(whichCardObject, playerIndex, offsetR, offsetC)

cardDivString = "<div class='flip-container'>";

cardDivString += "<div id='" + cardValue + "' class='cardFlip'><div class='front " + whichClass + "'><p class='cardText'>&nbsp;</p></div>";

cardDivString += "<div class='back " + whichClass + "'>" + imageString + "<p class='cardText'>" + whichValue + "</p></div></div></div>";

$(indexCell).append(cardDivString);

}

// ======= ======= ======= flipCards ======= ======= =======

Game.prototype.flipCards = function() {

console.log("flipCards");

self = this;

interval = 400;

cardIndex = -1;

cardCount = 0;

playerIndex = 0;

nextPlayer = this.playerObjectsArray[0];

flipCards = setInterval(function() {

cardCount++;

if (cardCount > 20) {

stopFlips();

}

if (interval > 400) {

stopFlips();

}

cardIndex++;

if (cardIndex > nextPlayer.hand.length - 1) {

if (nextPlayer.name == "dealer") {

interval = 1000;

} else {

playerIndex++;

if (playerIndex > self.playerNamesArray.length - 1) {

nextPlayer = dealer;

} else {

nextPlayer = self.playerObjectsArray[playerIndex];

}

}

cardIndex = 0;

}

nextCard = nextPlayer.hand[cardIndex];

$("#" + nextCard).addClass('flipper');

}, interval);

function stopFlips() {

clearInterval(flipCards);

}

}

// ======= ======= ======= placeBet ======= ======= =======

Game.prototype.placeBet = function(whichBet, whichPlayer, whichPlayerScreen) {

console.log("placeBet");

var onesBank = whichPlayer.onesBank;

var fivesBank = whichPlayer.fivesBank;

var tensBank = whichPlayer.tensBank;

var onesBet = whichPlayer.onesBet;

var fivesBet = whichPlayer.fivesBet;

var tensBet = whichPlayer.tensBet;

var limitFlag = false;

switch(whichBet) {

case "ones":

onesBank = onesBank - 1;

if (onesBank < 0) {

onesBank = 0;

limitFlag = true;

} else {

onesBet = onesBet + 1;

}

break;

case "fives":

fivesBank = fivesBank - 5;

if (fivesBank < 0) {

fivesBank = 0;

limitFlag = true;

} else {

fivesBet = fivesBet + 5;

}

break;

case "tens":

tensBank = tensBank - 10;

if (tensBank < 0) {

tensBank = 0;

limitFlag = true;

} else {

tensBet = tensBet + 10;

}

break;

}

if (limitFlag == false) {

whichPlayer.onesBank = onesBank;

whichPlayer.fivesBank = fivesBank;

whichPlayer.tensBank = tensBank;

whichPlayer.onesBet = onesBet;

whichPlayer.fivesBet = fivesBet;

whichPlayer.tensBet = tensBet;

} else {

$("#tooltips").text("Oops you're out of money!");

}

}

// ======= ======= ======= returnBet ======= ======= =======

Game.prototype.returnBet = function(whichBet, whichPlayer, whichPlayerScreen) {

console.log("returnBet");

var onesBank = whichPlayer.onesBank;

var fivesBank = whichPlayer.fivesBank;

var tensBank = whichPlayer.tensBank;

var onesBet = whichPlayer.onesBet;

var fivesBet = whichPlayer.fivesBet;

var tensBet = whichPlayer.tensBet;

var limitFlag = false;

switch(whichBet) {

case "ones":

onesBet = onesBet - 1;

if (onesBet < 0) {

onesBet = 0;

limitFlag = true;

} else {

onesBank = onesBank + 1;

}

break;

case "fives":

fivesBet = fivesBet - 5;

if (fivesBet < 0) {

fivesBet = 0;

limitFlag = true;

} else {

fivesBank = fivesBank + 5;

}

break;

case "tens":

tensBet = tensBet - 10;

if (tensBet < 0) {

tensBet = 0;

limitFlag = true;

} else {

tensBank = tensBank + 10;

}

break;

}

if (limitFlag == false) {

whichPlayer.onesBank = onesBank;

whichPlayer.fivesBank = fivesBank;

whichPlayer.tensBank = tensBank;

whichPlayer.onesBet = onesBet;

whichPlayer.fivesBet = fivesBet;

whichPlayer.tensBet = tensBet;

} else {

$("#tooltips").text("Total bet is returned");

}

}

// ======= ======= ======= playGame ======= ======= =======

Game.prototype.playGame = function() {

console.log("playGame");

this.currentPlayer = game.playerObjectsArray[0];

for (var i = 0; i < this.playerNamesArray.length; i++) {

nextPlayer = this.playerObjectsArray[i];

if (nextPlayer.name == this.currentPlayer.name) {

display.updateSubscreen(nextPlayer.id, "hitMeHoldMe");

} else {

display.updateSubscreen(nextPlayer.id, "turnOver");

}

}

}

// ======= ======= ======= hitMe ======= ======= =======

Game.prototype.hitMe = function() {

console.log("hitMe");

var nextPlayer = this.currentPlayer;

var nextPlayerScreen = this.subcreenObjectsArray[nextPlayer.id - 1];

var cardPoints = this.getNextCard(); // get card from deck; shrink deck

var nextCard = cardPoints[0];

var nextPoints = cardPoints[1];

console.log(' nextCard: ' + nextCard);

nextPlayer.hand.push(nextCard);

nextPlayer.score = nextPlayer.score + nextPoints;

this.displayNextCard(nextPlayer, nextPlayerScreen);

$("#pScore\_" + nextPlayer.id).text(nextPlayer.score);

// ======= check for Aces and adjust score

if (nextPlayer.score > 21) {

for (var i = 0; i < nextPlayer.hand.length; i++) {

nextCard = nextPlayer.hand[i];

// ======= change A value to 1 if > 21 score

if (nextCard.indexOf("A") > 0) {

$("#tooltips").text("Your're okay with ace value = 1");

nextPlayer.score = nextPlayer.score - 10;

break;

}

}

// ======= score still high after adjustment

if (nextPlayer.score > 21) {

$("#tooltips").text("Bummer... you're over 21!");

this.turnOver();

}

}

flipCardsP = setTimeout(function(){

nextCard = nextPlayer.hand[nextPlayer.hand.length - 1];

$("#" + nextCard).addClass('flipper');

}, 200);

}

// ======= ======= ======= holdMe ======= ======= =======

Game.prototype.holdMe = function() {

console.log("holdMe");

$("#tooltips").text("Next player turn");

this.turnOver();

}

// ======= ======= ======= turnOver ======= ======= =======

Game.prototype.turnOver = function() {

console.log("turnOver");

console.log(" PREV player: " + this.currentPlayer.name);

var currentPlayerIndex = this.currentPlayer.id;

var currentPlayer = this.currentPlayer;

var nextPlayer;

if (currentPlayerIndex < (this.playerNamesArray.length)) {

display.updateSubscreen(currentPlayerIndex, "turnOver");

nextPlayer = this.playerObjectsArray[currentPlayerIndex];

this.currentPlayer = nextPlayer;

console.log(" NEXT player: " + this.currentPlayer.name);

display.updateSubscreen(currentPlayerIndex + 1, "hitMeHoldMe");

$("#tooltips").text(nextPlayer.name + "'s turn");

} else {

display.updateSubscreen(currentPlayerIndex, "turnOver");

this.currentPlayer = this.dealer;

$("#tooltips").text("dealer's turn");

this.hitDealer();

}

}

// ======= ======= ======= hitDealer ======= ======= =======

Game.prototype.hitDealer = function() {

console.log("");

console.log("hitDealer");

self = this;

// == hit dealer again or end hand

if (dealer.score < 18) {

console.log(" dealer.hand.length1: " + dealer.hand.length);

cardPointsArray = game.getNextCard();

nextCard = cardPointsArray[0];

nextPoints = cardPointsArray[1];

dealer.hand.push(nextCard);

console.log(" dealer.hand.length1: " + dealer.hand.length);

dealer.score = dealer.score + nextPoints;

this.displayNextCard(dealer, dealer\_scr);

$("#pScore\_D").text(nextPlayer.score);

flipCards = setTimeout(function(){

nextCard = dealer.hand[dealer.hand.length - 1];

$("#" + nextCard).addClass('flipper');

if (dealer.score < 18) {

self.hitDealer();

} else {

display.nextGameScreen();

game.doTheMath();

$("#tooltips").text("");

}

}, 800);

} else {

display.nextGameScreen();

game.doTheMath();

$("#tooltips").text("");

}

}

// ======= ======= ======= doTheMath ======= ======= =======

Game.prototype.doTheMath = function() {

console.log("doTheMath");

var nextPlayer, nextName, winLossLabel;

var dealerScore = game.dealer.score;

var playerWinLoss = 0;

var playerWinLossString = 'RESULTS!!\nDealer score: ' + dealerScore + '\n\n';

// =======

for (var i = 0; i < (game.playerNamesArray.length); i++) {

nextPlayer = game.playerObjectsArray[i];

nextPlayerScreen = game.subscreenObjectsArray[i];

nextName = nextPlayer.name;

console.log(" nextName: " + nextName);

// ======= calculate win/loss results

playerWinLoss = nextPlayer.onesBet + nextPlayer.fivesBet + nextPlayer.tensBet;

console.log(" playerWinLoss: " + playerWinLoss);

// ======= calculate wins/losses for players

if ((nextPlayer.score > dealerScore) && (nextPlayer.score < 22)) {

winLossLabel = ' and won $';

nextPlayer.onesBank += (nextPlayer.onesBet \* 2);

nextPlayer.fivesBank += (nextPlayer.fivesBet \* 2);

nextPlayer.tensBank += (nextPlayer.tensBet \* 2);

nextPlayer.totalBank = nextPlayer.totalBank + (playerWinLoss \* 2);

} else if ((dealerScore > 21) && (nextPlayer.score < 22)) {

winLossLabel = ' and won $';

nextPlayer.onesBank += (nextPlayer.onesBet \* 2);

nextPlayer.fivesBank += (nextPlayer.fivesBet \* 2);

nextPlayer.tensBank += (nextPlayer.tensBet \* 2);

nextPlayer.totalBank = nextPlayer.totalBank + (playerWinLoss \* 2);

} else if (((nextPlayer.score > 21) || (nextPlayer.score < dealerScore)) && (deale...

winLossLabel = ' and lost $';

} else {

winLossLabel = ' tie game' + '\n';

nextPlayer.onesBank += nextPlayer.onesBet;

nextPlayer.fivesBank += nextPlayer.fivesBet;

nextPlayer.tensBank += nextPlayer.tensBet;

nextPlayer.totalBank = nextPlayer.totalBank + playerWinLoss;

playerWinLoss = ' no wins/losses' + '\n';

}

playerWinLossString += nextName + "'s score: " + nextPlayer.score + winLos...

nextPlayer.score = 0;

nextPlayer.onesBet = 0;

nextPlayer.fivesBet = 0;

nextPlayer.tensBet = 0;

display.updatePlayerScreenData(nextPlayerScreen, nextPlayer);

}

dealer.score = 0;

flipCardsP = setTimeout(function(){

alert(playerWinLossString);

}, 1000);

}

// ======= ======= ======= ======= ======= initialize ======= ======= ======= ======= =======

// ======= ======= ======= ======= ======= initialize ======= ======= ======= ======= =======

// ======= ======= ======= ======= ======= initialize ======= ======= ======= ======= =======

var player1 = new Player(1, "player1");

var player2 = new Player(2, "player2");

var player3 = new Player(3, "player3");

var dealer = new Player(4, "dealer");

var game = new Game("game1");

var display = new Display("display1");

display.initRowSpans();

display.nextGameScreen();

}