**blackjack**

**code**

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

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

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

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

Display.prototype.nextSubscreen = function(playerIndex) {

console.log("nextSubscreen");

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

// == player screen items include player index as suffix (e.g. "\_1")

// == makePlayerItems adds current player suffix to generic item names

currentPlayerScreen = game.subcreenObjectsArray[playerIndex - 1];

console.log(" currentPlayerScreen.name: " + currentPlayerScreen.name);

// == check for special case of dealer

if (playerIndex == 4) {

playerIndex = "D";

}

subscreenItems = this.playerStateItems["inactive"];

var nextItemType, items, nextItem, nextItemName;

var tempItemsNext = [];

var tempNamesNext = [];

itemTypesArray = ["bg", "btn", "slider", "text", "input", "image"];

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

nextItemType = itemTypesArray[i];

if (currentPlayerScreen[nextItemType]) {

items = Array.isArray(currentPlayerScreen[nextItemType]);

if (items) {

for (var j = 0; j < currentPlayerScreen[nextItemType].length; j++) {

nextItem = currentPlayerScreen[nextItemType][j];

nextItemName = nextItem.name;

// console.log(" nextItemName: " + nextItemName);

var found = $.inArray(nextItemName, subscreenItems) > -1;

if (found) {

tempItemsNext.push(nextItem);

tempNamesNext.push(nextItemName);

}

}

} else {

nextItem = currentPlayerScreen[nextItemType];

nextItemName = nextItem.name;

var found = $.inArray(nextItemName, subscreenItems) > -1;

if (found) {

tempItemsNext.push(nextItem);

tempNamesNext.push(nextItemName);

}

}

}

}

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

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

nextItem = tempItemsNext[j];

nextType = nextItem.type;

indexCell = display.modifyGridRegion(nextItem, playerIndex);

if (nextType == "input") {

newTextInput = "<input id='" + nextItem.name + "Input' class='" + nextItem.class + "' type='text' value='Tom'>"

$(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");

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

var prevScreen, nextScreen;

var REMprevItems = [];

var ADDnextItems = [];

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

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

var prevPlayerState = whichPlayer.state;

prevPlayerItems = this.playerStateItems[prevPlayerState];

whichPlayer.state = nextPlayerState;

nextPlayerItems = this.playerStateItems[nextPlayerState];

var REMprevNames = $(prevPlayerItems).not(nextPlayerItems).get();

var ADDnextNames = $(nextPlayerItems).not(prevPlayerItems).get();

itemTypesArray = ["bg", "btn", "slider", "text", "input", "image"];

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

nextItemType = itemTypesArray[i];

if (whichPlayerScreen[nextItemType]) {

items = Array.isArray(whichPlayerScreen[nextItemType]);

if (items) {

for (var j = 0; j < whichPlayerScreen[nextItemType].length; j++) {

nextItem = whichPlayerScreen[nextItemType][j];

nextItemName = nextItem.name;

var found = $.inArray(nextItemName, REMprevNames) > -1;

if (found) {

REMprevItems.push(nextItem);

}

}

} else {

nextItem = whichPlayerScreen[nextItemType];

nextItemName = nextItem.name;

var found = $.inArray(nextItemName, REMprevNames) > -1;

if (found) {

REMprevItems.push(nextItem);

}

}

}

}

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

nextItemType = itemTypesArray[i];

if (whichPlayerScreen[nextItemType]) {

items = Array.isArray(whichPlayerScreen[nextItemType]);

if (items) {

for (var j = 0; j < whichPlayerScreen[nextItemType].length; j++) {

nextItem = whichPlayerScreen[nextItemType][j];

nextItemName = nextItem.name;

var found = $.inArray(nextItemName, ADDnextNames) > -1;

if (found) {

ADDnextItems.push(nextItem);

}

}

} else {

nextItem = whichPlayerScreen[nextItemType];

nextItemName = nextItem.name;

var found = $.inArray(nextItemName, ADDnextNames) > -1;

if (found) {

ADDnextItems.push(nextItem);

}

}

}

}

this.addRemoveScreenItems(REMprevItems, ADDnextItems, playerIndex);

}

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

Display.prototype.nextGameScreen = function() {

console.log("");

console.log("nextGameScreen");

var prevScreen, nextScreen;

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

if (game.currentScreen == null) {

game.currentScreen = splash;

prevScreen = { name: "start"};

nextScreenIndex = 0;

// == get prev and next screen objects

} else {

prevScreen = game.currentScreen;

prevScreenName = prevScreen.name;

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

nextName = game.screenObjectsArray[i].name;

if (nextName == prevScreenName) {

nextScreenIndex = i + 1;

break

}

}

}

nextScreen = game.screenObjectsArray[nextScreenIndex];

game.currentScreen = nextScreen;

console.log("== " + prevScreen.name + " ==");

console.log("== " + nextScreen.name + " ==");

// == identify items to remove, add or keep

var changeItemsArray = this.sortPrevNextItems(prevScreen, nextScreen);

var removeItemsArray = changeItemsArray[0]; // delete these items

var addItemsArray = changeItemsArray[1]; // add these items

// == send screen data to screen building functions

this.addRemoveScreenItems(removeItemsArray, addItemsArray);

}

// ======= ======= ======= sortPrevNextItems ======= ======= =======

Display.prototype.sortPrevNextItems = function(prevScreen, nextScreen) {

console.log("sortPrevNextItems");

var REMprevItems = [];

var ADDnextItems = [];

var tempNamesPrev = [];

var tempNamesNext = [];

var tempItemsPrev = [];

var tempItemsNext = [];

// create lists of screen item NAMES for prev and next screen items

itemTypesArray = ["bg", "btn", "slider", "text", "input", "image"];

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

nextItemType = itemTypesArray[i];

if (prevScreen) {

if (prevScreen[nextItemType]) {

items = Array.isArray(prevScreen[nextItemType]);

if (items) {

for (var j = 0; j < prevScreen[nextItemType].length; j++) {

nextItem = prevScreen[nextItemType][j];

nextItemName = nextItem.name;

tempItemsPrev.push(nextItem);

tempNamesPrev.push(nextItemName);

}

} else {

nextItem = prevScreen[nextItemType];

nextItemName = nextItem.name;

tempItemsPrev.push(nextItem);

tempNamesPrev.push(nextItemName);

}

}

}

if (nextScreen[nextItemType]) {

items = Array.isArray(nextScreen[nextItemType]);

if (items) {

for (var j = 0; j < nextScreen[nextItemType].length; j++) {

nextItem = nextScreen[nextItemType][j];

nextItemName = nextItem.name;

tempItemsNext.push(nextItem);

tempNamesNext.push(nextItemName);

}

} else {

nextItem = nextScreen[nextItemType];

nextItemName = nextItem.name;

tempItemsNext.push(nextItem);

tempNamesNext.push(nextItemName);

}

}

}

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

REMprevNames = $(tempNamesPrev).not(tempNamesNext).get();

ADDnextNames = $(tempNamesNext).not(tempNamesPrev).get();

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

if (tempItemsPrev) {

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

nextItem = tempItemsPrev[i];

nextItemName = nextItem.name;

var found = $.inArray(nextItemName, REMprevNames)

if (found > -1) {

REMprevItems.push(nextItem);

}

}

}

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

nextItem = tempItemsNext[i];

nextItemName = nextItem.name;

var found = $.inArray(nextItemName, ADDnextNames)

if (found > -1) {

ADDnextItems.push(nextItem);

}

}

return [REMprevItems, ADDnextItems];

}

// ======= ======= ======= 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='" + nextItem.class + "' type='text' value='Tom'>";

$(indexCell).append(newTextInput);

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

}

if (nextType == "slider") {

var sliderLoc = $("#" + nextItem.name + "\_" + playerIndex).offset(); // location of grid cell

var locX = sliderLoc.left + 80;

var locY = sliderLoc.top + 15;

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

sliderString = "<div id='" + sliderId + "' class='slider " + nextItem.class + "'>&nbsp;</div>";

$(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);

}

}

}

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

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

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

if (playerIndex) {

playerIndex = "\_" + playerIndex;

}

if (!offsetR) { offsetR = 0 };

if (!offsetC) { offsetC = 0 };

var tableRows = $("tr");

var regionType = whichItem.merge;

// console.log(" whichItem.merge: " + 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 = $(new Image()).attr('src', "images/" + whichItem.image).appendTo($(indexCell));

$(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;

// == 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;

// console.log(" temp\_iC: " + temp\_iC);

if (regionType == "merge") {

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

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

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

$(indexCell).remove();

// console.log(" $(indexCell).attr('id'): " + $(indexCell).attr('id'));

}

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 {

$(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;

}

}

// ======= ======= ======= 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");

// == record rowspan elements

if (whichItem.iH > 1) {

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

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

// rowspans not recorded for first row of multi-row area

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);

// == display rowspan status for targeted row

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);

}

}

}

}

}

}

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

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

console.log("getTooltip");

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

}

// ======= ======= ======= 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;

}

// ======= ======= ======= 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) {

// console.log(" initSlider");

event.preventDefault();

var evt = newEvent || window.event;

this.start\_mouseX = newEvent.clientX;

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

console.log(" this.start\_elementX: " + this.start\_elementX);

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

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

addEventSimple(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);

updatePlayerScreenText();

}, 300);

} else {

updateChips = setInterval(function() {

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

game.returnBet(currentChips, player\_object);

updatePlayerScreenText();

}, 300);

}

function updatePlayerScreenText() {

console.log('updatePlayerScreenText');

// $("#" + player\_screen.slider[0].name + "\_" + player\_object.id).text("$" + player\_object.onesBank);

// $("#" + player\_screen.slider[1].name + "\_" + player\_object.id).text("$" + player\_object.fivesBank);

// $("#" + player\_screen.slider[2].name + "\_" + player\_object.id).text("$" + player\_object.tensBank);

$("#" + player\_screen.text[4].name + "\_" + player\_object.id).text("$" + player\_object.onesBet);

$("#" + player\_screen.text[5].name + "\_" + player\_object.id).text("$" + player\_object.fivesBet);

$("#" + player\_screen.text[6].name + "\_" + player\_object.id).text("$" + player\_object.tensBet);

var totalBet = player\_object.onesBet + player\_object.fivesBet + player\_object.tensBet;

var totalBank = player\_object.onesBank + player\_object.fivesBank + player\_object.tensBank;

$("#" + player\_screen.text[1].name + "\_" + player\_object.id).text("$" + totalBank);

$("#" + player\_screen.text[3].name + "\_" + player\_object.id).text("$" + totalBet);

}

},

// ======= 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(document, 'mouseup', slider.dropSlider);

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

}

}

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

function addEventSimple(obj,evt,fn) {

console.log('addEventSimple');

console.log(' obj: ' + obj.id);

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;

}

// ======= ======= ======= updatePlayerScoreText ======= ======= =======

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

console.log("updatePlayerScoreText");

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

var playerScoreCell = "#pScore\_D";

} else {

var playerScoreCell = "#pScore\_" + whichPlayer.id;

}

$(playerScoreCell).text(whichPlayer.score);

var playerScreenText = whichPlayerScreen.text;

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

nextText = playerScreenText.name;

if (nextText == "pName") {

nextText.value = whichPlayer.name;

}

if (nextText == "pScore") {

nextText.value = 0;

}

}

}

// ======= ======= ======= updatePlayerScreenText ======= ======= =======

// Display.prototype.updatePlayerScreenText = function(whichPlayer, whichPlayerScreen) {

// console.log("updatePlayerScreenText");

//

...

// }

// ======= ======= ======= updatePlayerBetText ======= ======= =======

Display.prototype.updatePlayerBetText = function(whichPlayer) {

console.log("updatePlayerBetText");

$("#" + whichPlayer.textParams.pBet\_1s.name).text("$0");

$("#" + whichPlayer.textParams.pBet\_5s.name).text("$0");

$("#" + whichPlayer.textParams.pBet\_10s.name).text("$0");

$("#" + whichPlayer.textParams.pBet.name).text("$0");

$("#" + whichPlayer.textParams.pBank.name).text("$" + whichPlayer.totalBank);

}

// ======= ======= ======= updatePlayerNames ======= ======= =======

Display.prototype.updatePlayerNames = function(whichPlayer, playerId) {

console.log("updatePlayerNames");

var playerScreen = game.subcreenObjectsArray[playerId - 1];

if (playerId == 4) {

playerId = "D";

} else {

whichPlayer.name = $('#playerNameInput').val();

}

var playerNameCell = "#pName\_" + playerId;

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

playerScreenText = playerScreen.text;

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

nextText = playerScreenText.name;

if (nextText == "pName") {

nextText.value = whichPlayer.name;

}

if (nextText == "pScore") {

nextText.value = 0;

}

}

}

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

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

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

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

Game.prototype.startGame = function() {

console.log("startGame");

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

display.updatePlayerNames(dealer, 4);

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();

}

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

Game.prototype.saveNewPlayer = function() {

console.log("saveNewPlayer");

var playerName, playerCount, playerState, newPlayer;

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

if (playerName) {

playerCount = this.playerNamesArray.length;

if (playerCount < 3) {

newPlayer = this.playerObjectsArray[playerCount];

newPlayer.name = playerName;

this.playerNamesArray.push(newPlayer.name);

playerCount = this.playerNamesArray.length;

display.nextSubscreen(playerCount);

display.updatePlayerNames(newPlayer, playerCount);

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

}

if (playerCount == 1) {

display.nextGameScreen();

}

if (playerCount == 3) {

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

game.startGame();

}

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

} else {

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

}

}

// ======= ======= ======= 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.updatePlayerScoreText(nextPlayerScreen, nextPlayer);

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

}

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

dealer.hand = []

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

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

nextCard = cardPointsArray[0];

nextPoints = cardPointsArray[1];

dealer.hand.push(nextCard);

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

this.displayNextCard(dealer, dealerScreen);

}

display.updatePlayerScoreText(dealerScreen, dealer);

this.flipCards();

}

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

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

// console.log("displayNextCard");

var whichMerge, cardDivString;

var playerIndex = whichPlayer.id;

// == get player card object for positioning

var subscreenTextObjects = whichSubscreen.text;

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

nextTextObject = subscreenTextObjects[i];

nextTextObjectName = subscreenTextObjects[i].name;

nextTextObjectClass = subscreenTextObjects[i].class;

if (nextTextObjectName == "pCards") {

whichCardObject = nextTextObject;

whichClass = nextTextObjectClass;

break;

}

}

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);

}

// ======= ======= ======= 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];

}

// ======= ======= ======= 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");

console.log(" whichPlayer: " + whichPlayer.name);

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, "inactive");

}

}

}

}

// ======= ======= ======= ======= ======= 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();

// display.getNextsubscreen();

}