**blackjack**

**grid code**

// ======= ======= ======= clearAllCardStacks ======= ======= =======

Sequencer.prototype.clearAllCardstacks = function() {

console.log("clearAllCardstacks");

var tableRows = $(".row");

if (game.playerNamesArray.length > 0) {

// == clear previous hand for players and dealer

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

nextPlayer = game.playerObjectsArray[i];

this.clearPlayerCardstack(nextPlayer);

}

// == clear dealer cards

if (game.dealer.hand.length > 0) {

this.clearPlayerCardstack(game.dealer);

}

}

}

// ======= ======= ======= clearPlayerCardstack ======= ======= =======

Sequencer.prototype.clearPlayerCardstack = function(nextPlayer) {

console.log("clearPlayerCardstack");

var tableRows = $(".row");

var cardCount = nextPlayer.hand.length;

var whichCardObject = nextPlayer.textParams.pCards;

var cardsRow = whichCardObject.iR;

var cardsRowObject = tableRows[cardsRow];

if (nextPlayer == game.dealer) {

var cardsCol = whichCardObject.iC;

} else {

var cardsCol = whichCardObject.iC - cardCount + 1;

}

colspans = display.checkColumnSpans(cardsRowObject, cardsCol);

rowspans = display.checkRowSpans(cardsRow, cardsCol);

var cardsCol = cardsCol - (colspans + rowspans);

// == identify card cells in table row and remove

for (var card = 0; card < cardCount; card++) {

$(tableRows[cardsRow]).children()[cardsCol].remove();

}

// == add new single cells for each row/column of card

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

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

var indexCell = $(tableRows[cardsRow + row]).children()[cardsCol - 1 + col];

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

$(indexCell).after(newCell);

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

$(newCell).attr("id", (cardsRow) + "-" + (cardsCol + col));

}

}

// ======= initialize values on player object

if (nextPlayer != game.dealer) {

nextPlayer.onesBet = 0;

nextPlayer.fivesBet = 0;

nextPlayer.tensBet = 0;

game.updateBetButtonText(nextPlayer);

}

nextPlayer.hand = [];

nextPlayer.score = 0;

game.updatePlayerScoreText(nextPlayer);

}

// ======= ======= ======= mergeRegion ======= ======= =======

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

console.log("mergeRegion");

console.log(" ======= ======= ======= item: " + whichItem.name);

if (!offsetR) { offsetR = 0 };

if (!offsetC) { offsetC = 0 };

var indexRow, indexCell, indexRowObject, rowspanSpanObject, colspans, rowspans;

// == record rowspan elements

if (whichItem.iH > 1) {

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

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

if (row != 0) {

tableRow = whichItem.iR + offsetR + row;

tableCol = whichItem.iC + offsetC + col;

rowspanSpanObject = this.tableRowspansArray[tableRow][tableCol];

rowspanSpanObject.rspan = true;

}

}

}

}

// == get index cell location (check row/colspans in index row)

var tableRows = $("tr");

indexRow = whichItem.iR + offsetR;

indexCol = whichItem.iC + offsetC;

indexRowObject = tableRows[indexRow];

colspans = this.checkColumnSpans(indexRowObject, indexCol);

rowspans = this.checkRowSpans(indexRow, indexCol);

totalColOffset = indexCol - colspans - rowspans;

indexCell = $(indexRowObject).children()[indexCol - colspans - rowspans];

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

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

nextRowObject = tableRows[whichItem.iR + offsetR + row];

nextRow = whichItem.iR + offsetR + row;

nextCol = whichItem.iC + offsetC;

colspans = this.checkColumnSpans(nextRowObject, nextCol);

rowspans = this.checkRowSpans(nextRow, nextCol);

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

totalColOffset = nextCol + col - colspans - rowspans;

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

// == remove all but index cell in merge area

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

$(nextCell).remove();

}

}

}

// == set row/colspans on index cell to fill space

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

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

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

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

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

}

return indexCell;

}

// ======= ======= ======= unMergeRegion ======= ======= =======

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

console.log("unMergeRegion");

console.log(" ======= ======= ======= item: " + whichItem.name);

if (!offsetR) { offsetR = 0 };

if (!offsetC) { offsetC = 0 };

var indexRow, indexCell, indexRowObject, rowspanSpanObject, colspans, rowspans;

// == remove rowspan elements from record

if (whichItem.iH > 1) {

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

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

rowspanSpanObject = this.tableRowspansArray[whichItem.iR + row][whichItem.iC + col];

rowspanSpanObject.rspan = false;

}

}

}

// == get index cell location (check row/colspans in index row)

var tableRows = $("tr");

indexRow = whichItem.iR + offsetR;

indexCol = whichItem.iC + offsetC;

indexRowObject = tableRows[indexRow];

colspans = this.checkColumnSpans(indexRowObject, indexCol);

rowspans = this.checkRowSpans(indexRow, indexCol);

indexCell = $(indexRowObject).children()[indexCol - colspans - rowspans];

// == remove merged index cell

$(indexCell).remove();

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

nextRowObject = tableRows[indexRow + row];

colspans = this.checkColumnSpans(nextRowObject, indexCol);

rowspans = this.checkRowSpans(row, indexCol);

// == set index to cell adjacent to (left of) removed cell

indexRowCell = $(nextRowObject).children()[indexCol - colspans - rowspans - 1];

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

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

$(indexRowCell).after(newCell);

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

$(newCell).attr("id", (indexRow + row) + "-" + (indexCol + col));

}

}

}

// ======= ======= ======= selectTableCells ======= ======= =======

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

console.log("selectTableCells");

console.log(" ======= ======= ======= item: " + whichItem.name);

if (!offsetR) { offsetR = 0 };

if (!offsetC) { offsetC = 0 };

var indexRow, indexCell, indexRowObject, rowspanSpanObject, colspans, rowspans;

var tableRows = $("tr");

indexRow = whichItem.iR + offsetR;

indexCol = whichItem.iC + offsetC;

indexRowObject = tableRows[indexRow];

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

nextRowObject = tableRows[indexRow + row];

colspans = this.checkColumnSpans(nextRowObject, indexCol);

rowspans = this.checkRowSpans(indexRow + row, indexCol);

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

nextCell = $(nextRowObject).children()[indexCol + col - colspans - rowspans];

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

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

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

indexCell = $(nextRowObject).children()[indexCol - colspans - rowspans];

}

}

}

return indexCell;

}

// ======= ======= ======= deselectTableCells ======= ======= =======

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

console.log("deselectTableCells");

console.log(" ======= ======= ======= item: " + whichItem.name);

if (!offsetR) { offsetR = 0 };

if (!offsetC) { offsetC = 0 };

var indexRow, indexCell, nextCell, indexRowObject, colspans, rowspans, rowCell, colCell;

// == get index cell location (check row/colspans in index row)

var tableRows = $("tr");

indexRow = whichItem.iR + offsetR;

indexCol = whichItem.iC + offsetC;

indexRowObject = tableRows[indexRow];

colspans = this.checkColumnSpans(indexRowObject, indexCol);

rowspans = this.checkRowSpans(indexRow, indexCol);

indexCell = $(indexRowObject).children()[indexCol - colspans - rowspans];

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

nextRowObject = tableRows[indexRow + row];

colspans = this.checkColumnSpans(nextRowObject, indexCol);

rowspans = this.checkRowSpans(row, indexCol);

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

nextCell = $(nextRowObject).children()[indexCol + col - colspans - rowspans];

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

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

$(nextCell).text("");

rowCell = whichItem.iR + row;

colCell = whichItem.iC + col;

}

}

$(indexCell).attr("id", (rowCell + "-" + colCell));

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

$(indexCell).empty();

return indexCell;

}

// ======= ======= ======= 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.tableRowspansArray[whichRow][col];

if (rowspanSpanObject.rspan == true) {

rowspans++;

}

}

}

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

return rowspans;

}

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

Display.prototype.checkColumnSpans = function(whichRowObject, 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');

console.log(" id/colspan " + nextColId + "/" + + nextColspan);

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

colspans += nextColspan - 1;

}

}

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

return colspans;

}