**blackjack**

**grid code**

// ======= ======= ======= Sequencer params ======= ======= =======

Sequencer.prototype.sequencerParams = function(whichState) {

// console.log("seqencerParams");

var player = game.playerObjectsArray[0];

var playerIndex = 0;

// ======= states =======

switch(whichState) {

case "splash":

var splash = {

name: "splash",

bg: null,

btn: [game.btnParams.orbBtn],

text: [game.textParams.tooltips],

input: null,

image: null

}

return splash;

break;

case "login":

var login = {

name: "login",

bg: null,

btn: [game.btnParams.enterBtn],

text: [game.textParams.tooltips],

input: [game.inputParams.playerName],

image: null

}

return login;

break;

case "saveStart":

var saveStart = {

name: "saveStart",

bg: null,

btn: [game.btnParams.enterBtn, game.btnParams.startBtn],

text: [game.textParams.tooltips],

input: [game.inputParams.playerName],

image: null

}

return saveStart;

break;

case "deal":

var deal = {

name: "deal",

bg: null,

btn: [game.gameParams("btn").dealBtn],

text: [game.gameParams("text").tooltips],

input: null,

image: null

}

return deal;

break;

case "bet":

var bet = {

name: "bet",

bg: null,

btn: [game.btnParams.playGameBtn, player.sliderParams.betOnesBtn, player.sliderParams.betFivesBtn, player.sliderParams.betTensBtn],

text: [game.gameParams("text").tooltips],

input: null,

image: null

}

return bet;

break;

case "playGame":

var playGame = {

name: "playGame",

bg: null,

btn: [player.sliderParams.betOnesBtn, player.sliderParams.betFivesBtn, player.sliderParams.betTensBtn, player.btnParams.hitMeBtn, player.btnParams.holdMeBtn],

text: [game.gameParams("text").tooltips],

input: null,

image: null

}

return playGame;

break;

case "hitMe":

var hitMe = {

name: "hitMe",

bg: null,

btn: [player.sliderParams.betOnesBtn, player.sliderParams.betFivesBtn, player.sliderParams.betTensBtn, player.btnParams.hitMeBtn, player.btnParams.holdMeBtn],

text: [game.gameParams("text").tooltips],

input: null,

image: null

}

return hitMe;

break;

case "holdMe":

var holdMe = {

name: "holdMe",

bg: null,

btn: [player.sliderParams.betOnesBtn, player.sliderParams.betFivesBtn, player.sliderParams.betTensBtn, player.btnParams.hitMeBtn, player.btnParams.holdMeBtn],

text: [game.gameParams("text").tooltips],

input: null,

image: null

}

return holdMe;

break;

case "turnOver":

var turnOver = {

name: "turnOver",

bg: null,

btn: [player.sliderParams.betOnesBtn, player.sliderParams.betFivesBtn, player.sliderParams.betTensBtn, player.btnParams.hitMeBtn, player.btnParams.holdMeBtn],

text: [game.gameParams("text").tooltips],

input: null,

image: null

}

return turnOver;

break;

case "hitDealer":

var hitDealer = {

name: "hitDealer",

bg: null,

btn: [game.gameParams("btn").playAgainBtn, game.gameParams("btn").newGameBtn, player.sliderParams.betOnesBtn, player.sliderParams.betFivesBtn, player.sliderParams.betTensBtn],

text: [game.gameParams("text").tooltips],

input: null,

image: null

}

return hitDealer;

break;

}

};

// ======= ======= ======= activateButton ======= ======= =======

Sequencer.prototype.activateButton = function(indexCell, whichAction) {

console.log("ACTIVATE\_button");

self = this;

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

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

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

whichIndexCell = event.target;

targetTooltip = display.getTargetTooltip(whichIndexCell, "over");

});

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

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

whichIndexCell = event.target;

targetTooltip = display.getTargetTooltip(whichIndexCell, "out");

});

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

switch(whichAction) {

case "nextGameState":

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

console.log("");

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

self.nextGameState();

$("#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 "deal":

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

console.log("");

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

game.deal();

});

break;

case "playGame":

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

console.log("");

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

game.playGame();

});

break;

case "mngBets":

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

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

display.activateScreenSlider(event);

});

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;

case "playAgain":

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

console.log("");

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

game.playAgain();

});

break;

case "newGame":

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

console.log("");

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

game.newGame();

});

break;

}

}

// ======= ======= ======= deActivateButton ======= ======= =======

Sequencer.prototype.deActivateButton = function(indexCell, whichAction) {

console.log("DEACTIVATE\_button");

// ======= event listeners =======

$(indexCell).off("mouseenter", null);

$(indexCell).off("mouseout", null);

$(indexCell).off("click", null);

}

// ======= ======= ======= Player ======= ======= =======

function Player(name, id) {

console.log('Player');

this.id = id;

this.name = name;

this.hand = null;

this.score = 0;

this.onesBank = 20;

this.fivesBank = 30;

this.tensBank = 50;

this.totalBank = 100;

this.onesBet = 0;

this.fivesBet = 0;

this.tensBet = 0;

this.bgParams = this.playerParams(id, "bg");

this.btnParams = this.playerParams(id, "btn");

this.textParams = this.playerParams(id, "text");

this.inputParams = this.playerParams(id, "input");

this.imageParams = this.playerParams(id, "image");

this.sliderParams = this.playerParams(id, "slider");

}

// ======= ======= ======= Game Object ======= ======= =======

function Game(whichGame) {

console.log('Game');

this.name = whichGame;

this.dealer = null;

this.deckArray = [];

this.deckPointsArray = [];

this.currentPlayer = null;

this.playerNamesArray = [];

this.playerObjectsArray = null;

this.bgParams = this.gameParams("bg");

this.btnParams = this.gameParams("btn");

this.textParams = this.gameParams("text");

this.inputParams = this.gameParams("input");

this.imageParams = this.gameParams("image");

this.prevBgs = [];

this.prevBtns = [];

this.prevTexts = [];

this.prevInputs = [];

this.prevImages = [];

this.onesBet = 0;

this.fivesBet = 0;

this.tensBet = 0;

this.display = null;

this.message = null;

}

// ======= ======= ======= Sequencer ======= ======= =======

function Sequencer() {

console.log('Sequencer');

this.name = "Sequencer";

this.currentGameState = "hitDealer";

this.gameStatesArray = ["splash", "login", "saveStart", "deal", "bet", "playGame", "turnOver", "hitDealer"];

}

// ======= ======= ======= Display ======= ======= =======

function Display(whichDisplay) {

console.log('Display');

this.name = whichDisplay;

this.tableCellsArray = null;

this.tableRowspansArray = null;

}

// ======= ======= ======= ======= ======= ======= SEQUENCER ======= ======= ======= ======= ======= =======

// ======= ======= ======= ======= ======= ======= SEQUENCER ======= ======= ======= ======= ======= =======

// ======= ======= ======= ======= ======= ======= SEQUENCER ======= ======= ======= ======= ======= =======

// ======= ======= ======= loadStartGameState ======= ======= =======

Sequencer.prototype.loadStartGameState = function(prevNext) {

console.log("loadStartGameState");

var nextItemTypes, nextItem, nextType;

var currentGameState = this.currentGameState;

var sequencerParams = this.sequencerParams(currentGameState);

var itemTypesArray = [sequencerParams.bg, sequencerParams.btn, sequencerParams.text, sequencerParams.input, sequencerParams.image]

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

nextItemTypes = itemTypesArray[j];

if (nextItemTypes != null) {

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

nextItem = nextItemTypes[i];

nextType = nextItem.type;

display.modifyGridRegion(nextItem, prevNext);

}

}

}

}

// ======= ======= ======= 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 colIndex, colspans, rowspans, totalSpanOffset, indexCell, rowSpanGridOffset, indexRowCell;

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

var cardCount = nextPlayer.hand.length;

var playerCardObject = nextPlayer.textParams.pCards; //pCards: { player:1,name:"pCards\_1",type:"text",iR:2,iC:5,iW:1,iH:2,merge:"merge",c... }

var cards\_iR = playerCardObject.iR;

var cards\_iC = playerCardObject.iC;

var tableRows = $("tr"); // table rows

var indexRowCollection = tableRows[cards\_iR]; // row columns (collection)

if (nextPlayer == game.dealer) {

colIndex = playerCardObject.iC;

} else {

colIndex = playerCardObject.iC - cardCount + 1;

}

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

colspans = display.checkColumnSpans(indexRowCollection, cards\_iR, colIndex);

rowspans = display.checkRowSpans(cards\_iR, colIndex);

totalSpanOffset = colIndex - colspans - rowspans;

indexCell = $(indexRowCollection).children()[totalSpanOffset]; // <td> element

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

display.toggleRowspans(playerCardObject, 0, -i, "off");

} else {

display.toggleRowspans(playerCardObject, 0, i, "off");

}

$(indexCell).remove();

}

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

var tableRows = $("tr"); // table rows

var indexRowCollection = tableRows[cards\_iR]; // row columns (collection)

if (nextPlayer == game.dealer) {

colIndex = playerCardObject.iC;

} else {

colIndex = playerCardObject.iC - cardCount + 1;

}

// ======= replace cells in card slots

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

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

nextRowCollection = tableRows[cards\_iR + row];

colspans = display.checkColumnSpans(nextRowCollection, cards\_iR, colIndex);

rowspans = display.checkRowSpans(cards\_iR, colIndex);

totalSpanOffset = colIndex - colspans - rowspans;

indexRowCell = $(nextRowCollection).children()[totalSpanOffset + col - 1];

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

$(indexRowCell).after(newCell);

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

$(newCell).attr("id", (cards\_iR) + "-" + (totalSpanOffset + col));

}

}

if (nextPlayer != game.dealer) {

game.updateBetButtonText(nextPlayer);

game.updatePlayerScoreText(nextPlayer);

nextPlayer.onesBet = 0;

nextPlayer.fivesBet = 0;

nextPlayer.tensBet = 0;

}

nextPlayer.hand = [];

nextPlayer.score = 0;

}

// ======= ======= ======= nextGameState ======= ======= =======

Sequencer.prototype.nextGameState = function(whichState) {

console.log("");

console.log("== nextGameState ==");

var currentGameState, sequencerParams, addItemsArray, removeItemsArray;

var playAgainFlag = false;

// == start of game (splash screen)

if (whichState == "splash") {

nextGameState = "splash";

console.log("== " + nextGameState + " ==");

this.currentGameState = nextGameState;

sequencerParams = this.sequencerParams(nextGameState);

prevItemTypesArray = [];

nextItemTypesArray = [sequencerParams.bg, sequencerParams.btn, sequencerParams.text, sequencerParams.input, sequencerParams.image]

} else {

// == set prev params items for current state

prevGameState = this.currentGameState;

console.log("== " + prevGameState + " ==");

sequencerParams = this.sequencerParams(prevGameState);

var prevItemTypesArray = [sequencerParams.bg, sequencerParams.btn, sequencerParams.text, sequencerParams.input, sequencerParams.image]

// == return to deal state if new game requested

var prevGameStateIndex = this.gameStatesArray.indexOf(prevGameState);

if (prevGameStateIndex == (this.gameStatesArray.length - 1)) {

var dealStateIndex = this.gameStatesArray.indexOf("bet");

nextGameState = this.gameStatesArray[dealStateIndex];

game.currentPlayer = game.playerObjectsArray[0];

playAgainFlag = true;

} else {

nextGameState = this.gameStatesArray[prevGameStateIndex + 1];

}

// == advance to next game state (set in game object)

this.currentGameState = nextGameState;

console.log("== " + nextGameState + " ==");

var sequencerParams = this.sequencerParams(nextGameState);

var nextItemTypesArray = [sequencerParams.bg, sequencerParams.btn, sequencerParams.text, sequencerParams.input, sequencerParams.image]

}

// == identify items to delete/keep/add

var changeItemsArray = this.swapPrevNextParams(nextItemTypesArray, prevItemTypesArray);

var removeItemsArray = changeItemsArray[2]; // delete these items

var addItemsArray = changeItemsArray[3]; // add these items

// == remove prev items/add next items

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

nextItem = removeItemsArray[j];

nextType = nextItem.type;

display.modifyGridRegion(nextItem, "prev");

}

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

nextItem = addItemsArray[j];

nextType = nextItem.type;

display.modifyGridRegion(nextItem, "next");

}

if (playAgainFlag == true) {

sequencer.activateBetSliders();

playAgainFlag == false;

game.deal();

}

}

// ======= ======= ======= activateBetSliders ======= ======= =======

Sequencer.prototype.activateBetSliders = function(nextItemsParamsObject, prevItemsParamsObject, itemType) {

console.log("activateBetSliders");

var sliderDiv;

var tableRows = $("tr");

var firstPlayer = game.playerObjectsArray[0];

var firstPlayerSliders = [firstPlayer.sliderParams.betOnesBtn, firstPlayer.sliderParams.betFivesBtn, firstPlayer.sliderParams.betTensBtn];

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

nextItem = firstPlayerSliders[j];

sliderDiv = display.makeSlider(nextItem);

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

sequencer.activateButton(sliderDiv, nextItem.callback)

}

}

// ======= ======= ======= swapPrevNextParams ======= ======= =======

Sequencer.prototype.swapPrevNextParams = function(nextItemsParamsObject, prevItemsParamsObject, itemType) {

// console.log("swapPrevNextParams");

var REMprevItems = [];

var ADDnextItems = [];

var tempNamesOld = [];

var tempNamesNew = [];

nextNewArray = nextItemsParamsObject;

nextOldArray = prevItemsParamsObject;

// == get items names for array processing

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

nextItemArray = nextOldArray[i];

if (nextItemArray) {

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

nextItem = nextItemArray[j];

nextItemName = nextItem.name;

tempNamesOld.push(nextItemName);

}

}

}

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

nextItemArray = nextNewArray[i];

if (nextItemArray) {

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

nextItem = nextItemArray[j];

nextItemName = nextItem.name;

tempNamesNew.push(nextItemName);

}

}

}

// == find array differences

ADDnextNames = $(tempNamesNew).not(tempNamesOld).get();

REMprevNames = $(tempNamesOld).not(tempNamesNew).get();

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

nextItemArray = nextOldArray[i];

if (nextItemArray) {

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

nextItem = nextItemArray[j];

nextItemName = nextItem.name;

var found = $.inArray(nextItemName, REMprevNames)

if (found > -1) {

REMprevItems.push(nextItem);

}

}

}

}

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

nextItemArray = nextNewArray[i];

if (nextItemArray) {

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

nextItem = nextItemArray[j];

nextItemName = nextItem.name;

var found = $.inArray(nextItemName, ADDnextNames)

if (found > -1) {

ADDnextItems.push(nextItem);

}

}

}

}

return [REMprevNames, ADDnextNames, REMprevItems, ADDnextItems];

}

// ======= ======= ======= nextPlayerTurn ======= ======= =======

Sequencer.prototype.nextPlayerTurn = function() {

console.log("");

console.log("== nextPlayerTurn ==");

console.log("== PREV: " + game.currentPlayer.name + " ==");

var indexCell;

var tableRows = $("tr");

// == deactivate/remove prev player buttons

var prevPlayer = game.currentPlayer;

var prevPlayerIndex = prevPlayer.id;

display.modifyGridRegion(prevPlayer.btnParams.hitMeBtn, "prev");

display.modifyGridRegion(prevPlayer.btnParams.holdMeBtn, "prev");

// == activate dealer if last player turn over

if (prevPlayerIndex != game.playerNamesArray.length - 1) {

// == activate next player buttons

var nextPlayer = game.playerObjectsArray[prevPlayerIndex + 1];

console.log("== NEXT: " + nextPlayer.name + " ==");

display.modifyGridRegion(nextPlayer.btnParams.hitMeBtn, "next");

display.modifyGridRegion(nextPlayer.btnParams.holdMeBtn, "next");

}

}

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

Sequencer.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];

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)) && (dealerScore < 22)) {

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 + winLossLabel + playerWinLoss + '\n\n';

nextPlayer.score = 0;

nextPlayer.onesBet = 0;

nextPlayer.fivesBet = 0;

nextPlayer.tensBet = 0;

game.updateBetButtonText(nextPlayer);

game.updatePlayerBetText(nextPlayer);

}

dealer.score = 0;

flipCardsP = setTimeout(function(){

alert(playerWinLossString);

}, 1000);

}

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

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

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

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

Game.prototype.startGame = function() {

console.log("startGame");

this.displayDealer();

this.updatePlayerNames(this.dealer, 4);

this.currentPlayer = this.playerObjectsArray[0];

sequencer.nextGameState();

}

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

Game.prototype.playGame = function() {

console.log("playGame");

var nextPlayerSliders, nextItem, indexRowObject, colspans, rowspans, totalColOffset, indexCell;

var tableRows = $("tr");

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

nextPlayer = this.playerObjectsArray[i];

nextPlayerSliders = [nextPlayer.sliderParams.betOnesBtn, nextPlayer.sliderParams.betFivesBtn, nextPlayer.sliderParams.betTensBtn];

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

nextItem = nextPlayerSliders[j];

indexRowObject = tableRows[nextItem.iR];

colspans = display.checkColumnSpans(indexRowObject, nextItem.iR, nextItem.iC);

rowspans = display.checkRowSpans(nextItem.iR, nextItem.iC);

totalColOffset = nextItem.iC - colspans - rowspans;

indexCell = $(indexRowObject).children()[totalColOffset];

display.unMakeSlider(nextItem);

sequencer.deActivateButton(indexCell, nextItem.callback)

}

}

this.currentPlayer = this.playerObjectsArray[0];

sequencer.nextGameState();

$("#tooltips").text(this.currentPlayer.name + " can hit or hold");

}

// ======= ======= ======= playAgain ======= ======= =======

Game.prototype.playAgain = function() {

console.log("playAgain");

game.updatePlayerScoreText(nextPlayer);

sequencer.clearAllCardstacks();

sequencer.nextGameState("bet");

}

// ======= ======= ======= newGame ======= ======= =======

Game.prototype.newGame = function() {

console.log("newGame");

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

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

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

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

sequencer.clearAllCardstacks();

display.housekeeping();

display.initGridElements();

game = null;

display = null;

player1 = null;

player2 = null;

player3 = null;

dealer = null;

sequencer = null;

game = new Game();

display = new Display("gameDisplay");

player1 = new Player(null, 0);

player2 = new Player(null, 1);

player3 = new Player(null, 2);

dealer = new Player("dealer", 3);

game.playerObjectsArray = [player1, player2, player3];

game.dealer = dealer;

sequencer = new Sequencer();

display.housekeeping();

display.initGridElements();

sequencer.nextGameState("splash");

}

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

Game.prototype.saveNewPlayer = function() {

console.log("saveNewPlayer");

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

if (playerName) {

var playerCount = this.playerNamesArray.length;

if (playerCount < 3) {

newPlayer = this.playerObjectsArray[playerCount];

this.playerNamesArray.push(newPlayer.name);

var playerCount = this.playerNamesArray.length;

this.displayPlayer(newPlayer);

this.updatePlayerNames(newPlayer, playerCount);

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

}

if (playerCount == 1) {

sequencer.nextGameState();

}

if (playerCount == 3) {

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

this.displayDealer();

this.updatePlayerNames(this.dealer, 4);

this.currentPlayer = game.playerObjectsArray[0];

sequencer.nextGameState();

$("#tooltips").text("click DEAL to deal cards");

}

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

} else {

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

}

}

// ======= ======= ======= displayPlayer ======= ======= =======

Game.prototype.displayPlayer = function() {

console.log("displayPlayer");

// == display player components (unique for each player)

var playerCount = game.playerNamesArray.length;

var currentPlayer = game.playerObjectsArray[playerCount - 1];

var playerParamsArray = [currentPlayer.bgParams.borderH, currentPlayer.bgParams.borderV, currentPlayer.textParams.pName, currentPlayer.textParams.pScore, currentPlayer.textParams.pBank];

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

nextItem = playerParamsArray[j];

if (nextItem != null) {

display.modifyGridRegion(nextItem, "next");

if (nextType == "slider") {

display.makeSlider(nextItem);

}

// == player buttons not active yet (display amounts only)

if ((nextItem.type == "btn") || (nextItem.type == "slider") || (nextItem.type == "2wayBtn")) {

var indexCell = display.tableCellsArray[nextItem.iR][nextItem.iC];

sequencer.deActivateButton(indexCell, "click")

}

}

}

currentPlayer.textParams.pName.value = this.name;

currentPlayer.textParams.pScore.value = 0;

currentPlayer.textParams.pBank.value = currentPlayer.totalBank;

}

// ======= ======= ======= displayDealer ======= ======= =======

Game.prototype.displayDealer = function() {

// console.log("displayDealer");

var playerParamsArray = [dealer.bgParams.borderH, dealer.bgParams.borderV, dealer.textParams.pName, dealer.textParams.pScore];

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

nextItem = playerParamsArray[j];

if (nextItem) {

display.modifyGridRegion(nextItem, "next");

}

}

dealer.textParams.pName.value = this.name;

dealer.textParams.pScore.value = 0;

}

// ======= ======= ======= deal ======= ======= =======

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

console.log("deal");

// ======= initialize deck

// var suitArray = ['&clubs; ','&diams; ','&hearts; ','&spades; '];

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++) {

var nextPlayer = this.playerObjectsArray[i];

nextPlayer.hand = [];

}

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

var winnersArray = [];

var nextPlayer;

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

nextPlayer = this.playerObjectsArray[i];

console.log(' nextPlayer.name: ' + nextPlayer.name);

// ======= 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);

}

// ======= 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);

}

this.updatePlayerScoreText(nextPlayer);

var playerParamsArray = [nextPlayer.sliderParams.betOnesBtn, nextPlayer.sliderParams.betFivesBtn, nextPlayer.sliderParams.betTensBtn, nextPlayer.textParams.pBet\_1s, nextPlayer.textParams.pBet\_5s, nextPlayer.textParams.pBet\_10s, nextPlayer.textParams.pBet];

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

nextItem = playerParamsArray[j];

if (nextItem != null) {

display.modifyGridRegion(nextItem, "next");

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

sliderDiv = display.makeSlider(nextItem);

sequencer.activateButton(sliderDiv, nextItem.callback)

}

}

}

$("#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);

}

this.updatePlayerScoreText(dealer);

// ======= set default player (unless winner)

if (winnersArray.length > 0) {

// calculateWinner();

} else {

console.log(" \*\* to playGame");

this.activePlayer = 1;

sequencer.nextGameState();

}

this.flipCards();

}

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

Game.prototype.hitMe = function() {

console.log("hitMe");

var nextPlayer = this.currentPlayer;

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);

this.updatePlayerScoreText(nextPlayer);

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

}

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

Game.prototype.hitDealer = function() {

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(); // get card from deck; shrink deck

nextCard = cardPointsArray[0];

nextPoints = cardPointsArray[1];

dealer.hand.push(nextCard);

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

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

this.displayNextCard(dealer); // display new card

this.updatePlayerScoreText(dealer); // display dealer score

flipCards = setTimeout(function(){

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

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

if (dealer.score < 18) {

self.hitDealer();

} else {

sequencer.nextGameState();

sequencer.doTheMath();

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

}

}, 800);

} else {

sequencer.nextGameState();

sequencer.doTheMath();

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

}

}

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

console.log("displayNextCard");

var whichMerge, cardDivString;

var whichCardObject = whichPlayer.textParams.pCards;

var whichName = whichCardObject.name;

var whichClass = whichPlayer.textParams.pName.class;

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);

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

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;

if (whichCardObject.merge == "merge") {

indexCell = display.modifyGridAreas(whichCardObject, offsetR, offsetC, "merge")

} else {

indexCell = display.modifyGridAreas(whichCardObject, 0, 0, "UNMERGE")

}

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 = self.dealer;

} else {

nextPlayer = self.playerObjectsArray[playerIndex];

}

}

cardIndex = 0;

}

nextCard = nextPlayer.hand[cardIndex];

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

}, interval);

function stopFlips() {

clearInterval(flipCards);

}

}

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

Game.prototype.turnOver = function() {

console.log("turnOver");

console.log(" this.currentPlayer: " + this.currentPlayer);

var currentPlayerIndex = this.currentPlayer.id;

var currentPlayer = this.currentPlayer;

if (currentPlayerIndex < (this.playerNamesArray.length - 1)) {

sequencer.nextPlayerTurn();

var nextPlayer = this.playerObjectsArray[currentPlayerIndex + 1];

this.currentPlayer = nextPlayer;

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

} else {

sequencer.nextPlayerTurn();

this.currentPlayer = this.dealer;

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

sequencer.nextGameState("hitDealer");

this.hitDealer();

}

}

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

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

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;

whichPlayer.totalBank = onesBank + fivesBank + tensBank;

game.updateBetButtonText(whichPlayer);

} else {

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

}

}

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

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

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;

whichPlayer.totalBank = onesBank + fivesBank + tensBank;

game.updateBetButtonText(whichPlayer);

} else {

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

}

}

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

Game.prototype.updatePlayerScoreText = function(whichPlayer) {

console.log("updatePlayerScoreText");

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

var playerScoreCell = "#pScore";

} else {

var playerScoreCell = "#pScore\_" + (whichPlayer.id + 1);

}

whichPlayer.textParams.pScore.value = whichPlayer.score;

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

}

// ======= ======= ======= updateBetButtonText ======= ======= =======

Game.prototype.updateBetButtonText = function(whichPlayer) {

console.log("updateBetButtonText");

$("#" + whichPlayer.sliderParams.betOnesBtn.name).text("$" + whichPlayer.onesBank);

$("#" + whichPlayer.sliderParams.betFivesBtn.name).text("$" + whichPlayer.fivesBank);

$("#" + whichPlayer.sliderParams.betTensBtn.name).text("$" + whichPlayer.tensBank);

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

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

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

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

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

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

}

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

Game.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 ======= ======= =======

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

// console.log("updatePlayerNames");

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

var playerNameCell = "#pName";

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

} else {

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

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

whichPlayer.textParams.pName.value = whichPlayer.name;

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

}

whichPlayer.textParams.pScore.value = 0;

}

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

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

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

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

Display.prototype.modifyGridRegion = function(whichItem, prevNext) {

// console.log("modifyGridRegion");

var whichMerge, buttonActivate;

if (prevNext == "prev") {

if (whichItem.merge == "merge") {

whichMerge = "unmerge";

} else {

whichMerge = "restore";

}

buttonActivate = false;

} else {

if (whichItem.merge == "merge") {

whichMerge = "merge";

} else {

whichMerge = null;

}

buttonActivate = true;

}

var whichType = whichItem.type;

var whichValue = whichItem.value;

var whichName = whichItem.name;

var textType = whichName.substring(0, 5);

if (textType == "pBank") {

whichValue = "$" + whichValue;

}

if (whichMerge == "merge") {

indexCell = display.modifyGridAreas(whichItem, 0, 0, "merge")

// indexCell = this.mergeRegion(whichItem);

} else if (whichMerge == "unmerge") {

indexCell = display.modifyGridAreas(whichItem, 0, 0, "UNMERGE")

// indexCell = this.unMergeRegion(whichItem);

} else if (whichMerge == "restore") {

indexCell = display.modifyGridAreas(whichItem, 0, 0, "deselect")

// indexCell = this.deselectTableCells(whichItem);

} else {

indexCell = display.modifyGridAreas(whichItem, 0, 0, "select")

// indexCell = this.selectTableCells(whichItem);

}

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

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

// ======= data connection =======

switch(whichType) {

case "bg":

break;

case "btn":

if (buttonActivate == true) {

if (whichItem.image == null) {

$(indexCell).text(whichValue);

}

sequencer.activateButton(indexCell, whichItem.callback)

} else {

sequencer.deActivateButton(indexCell, whichItem.callback)

}

break;

case "slider":

var sliderCellId = "#" + whichItem.name;

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

var sliderDiv = display.makeSlider(whichItem);

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

console.log(" ACTIVATE: " + $(sliderDiv).attr('id'));

sequencer.activateButton(sliderDiv, whichItem.callback)

case "text":

$(indexCell).text(whichValue);

break;

case "input":

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

$(indexCell).append(newTextInput);

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

case "image":

break;

}

}

// ======= ======= ======= modifyGridAreas ======= ======= =======

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

console.log("modifyGridAreas");

console.log(" ======= PROCESS ======= " + whichProcess);

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

if (!offsetR) { offsetR = 0 };

if (!offsetC) { offsetC = 0 };

var tableRows, indexRow, indexCol, indexRowObject, colspans, rowspans, indexColOffset, indexCell;

var nextRowObject, nextRow, totalSpanOffset, totalColOffset, nextCell;

// == record rowspan elements

if ((whichProcess == "merge") && (whichItem.iH > 1)) {

this.toggleRowspans(whichItem, offsetR, offsetC, "on");

} else if ((whichProcess == "UNMERGE") && (whichItem.iH > 1)) {

this.toggleRowspans(whichItem, offsetR, offsetC, "off");

}

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

tableRows = $("tr");

indexRow = whichItem.iR + offsetR;

indexCol = whichItem.iC + offsetC;

indexRowObject = tableRows[indexRow];

colspans = this.checkColumnSpans(indexRowObject, indexRow, indexCol);

rowspans = this.checkRowSpans(indexRow, indexCol);

indexColOffset = indexCol - colspans - rowspans;

indexCell = $(indexRowObject).children()[indexColOffset];

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

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

nextRow = indexRow + row;

nextRowObject = tableRows[nextRow];

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

rowspans = this.checkRowSpans(nextRow, indexCol);

totalSpanOffset = indexCol - colspans - rowspans;

if (whichProcess == "merge") {

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

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

totalColOffset = parseInt(totalSpanOffset + col);

}

if (row > 0) {

totalColOffset = parseInt(totalSpanOffset);

}

if (!((row == 0) && (col == 0))) { // remove all but index cell in merge area

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

$(nextCell).remove();

}

}

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

if (row == (whichItem.iH - 1)) {

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

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

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

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

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

}

}

} else if (whichProcess == "UNMERGE") {

$(indexCell).remove();

indexRowCell = $(nextRowObject).children()[totalSpanOffset - 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));

}

} else if (whichProcess == "select") {

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

nextCell = $(nextRowObject).children()[totalSpanOffset + col];

$(nextCell).removeClass();

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

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

indexCell = $(nextRowObject).children()[totalSpanOffset];

if ((whichItem.image != null) && (whichItem.type != "2wayBtn")) {

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

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

} else {

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

}

} else {

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

}

}

} else if (whichProcess == "deselect") {

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

nextCell = $(nextRowObject).children()[totalSpanOffset + col];

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

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

$(nextCell).text("");

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

indexCell = $(nextRowObject).children()[totalSpanOffset];

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

$(indexCell).empty();

} else {

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

}

}

}

}

return indexCell;

}

// ======= ======= ======= makeSlider ======= ======= =======

Display.prototype.makeSlider = function(whichItem) {

console.log("makeSlider");

var sliderLoc = $("#" + whichItem.name).offset(); // location of grid cell

var locX = sliderLoc.left + 100;

var locY = sliderLoc.top + 15;

var sliderId = "slider\_" + whichItem.name;

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

$("body").append(sliderString);

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

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

var sliderDiv = $("#" + sliderId);

return sliderDiv;

}

// ======= ======= ======= unMakeSlider ======= ======= =======

Display.prototype.unMakeSlider = function(whichItem) {

console.log("unMakeSlider");

var sliderId = "#slider\_" + whichItem.name;

$(sliderId).remove();

}

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

}

}

}

}

}

}

// ======= ======= ======= 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.tableRowspansArray[tableRow][tableCol].rspan = true;

// console.log(" rowspan SET: " + tableRow + "/" + tableCol);

} else {

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

// console.log(" rowspan CLEARED: " + tableRow + "/" + tableCol);

}

}

}

}

}

}

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

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

// console.log("checkRowSpans");

// console.log(" which\_R/C: " + whichRow + "/" + whichCol);

var rowspans = 0;

var indexRow = 0;

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

if (col < whichCol) {

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

// console.log(" rowspan\_R/C: " + rowspanSpanObject.R + "/" + rowspanSpanObject.C);

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;

}

// ======= ======= ======= getTargetTooltip ======= ======= =======

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

console.log("getTargetTooltip");

console.log(" whichElement: "+ whichElement.id);

for (var key in game.btnParams) {

nextId = game.btnParams[key].name;

if (nextId == whichElement.id) {

if (overOut == "over") {

nextTooltip = game.btnParams[key].tooltipOver;

} else {

nextTooltip = game.btnParams[key].tooltipOut;

}

break;

}

}

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

}

// ======= ======= ======= initGridElements ======= ======= =======

Display.prototype.initGridElements = function() {

// console.log("initGridElements");

var tableCol, cellRCs;

var tableCellsArray = [];

var tableRowspansArray = [];

var tableRows = $(".row");

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

nextRow = tableRows[row];

nextRowArray = [];

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

tableCellsArray.push(tableCols);

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

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

nextRowArray.push(cellRC);

nextCell = tableCols[col];

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

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

}

tableRowspansArray.push(nextRowArray);

}

this.tableCellsArray = tableCellsArray;

this.tableRowspansArray = tableRowspansArray;

}

// ======= ======= ======= housekeeping ======= ======= =======

Display.prototype.housekeeping = function() {

console.log("housekeeping");

var tableCol, cellRCs;

var tableCellsArray = [];

var tableRowspansArray = [];

var tableRows = $(".row");

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

nextRowObject = tableRows[row];

nextRowArray = [];

tableCols = $(nextRowObject).children();

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

nextCell = tableCols[col];

nextCellRspan = $(nextCell).attr("rowSpan");

nextCellCspan = $(nextCell).attr("colSpan");

if ((nextCellCspan > 1) || (nextCellRspan > 1)) {

for (var fillRow = 0; fillRow < nextCellRspan; fillRow++) {

nextRow = row + fillRow;

fillRowObject = tableRows[nextRow];

fillCols = $(fillRowObject).children();

fillCell = fillCols[col];

colspans = this.checkColumnSpans(fillRowObject, nextRow, col);

rowspans = this.checkRowSpans(nextRow, col);

if (fillRow == 0) {

$(fillCell).remove();

}

indexRowCell = fillCols[col - 1];

for (var fillCol = 0; fillCol < nextCellCspan; fillCol++) {

colTotal = col + fillCol;

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

$(indexRowCell).after(newCell);

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

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

$(newCell).attr("class", "cell");

}

}

} else {

$(nextCell).text("");

$(nextCell).removeClass();

$(nextCell).attr("id", row + "-" + col); // "holdMeBtn", row + "-" + col

$(nextCell).attr("class", "cell"); // "green\_grid"

}

}

tableCellsArray.push(tableCols);

tableRowspansArray.push(nextRowArray);

}

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

nextBtn = game.btnParams[i];

sequencer.deActivateButton(nextBtn);

}

this.tableCellsArray = tableCellsArray;

this.tableRowspansArray = tableRowspansArray;

}

// ======= ======= ======= activateScreenSlider ======= ======= =======

Display.prototype.activateScreenSlider = function(event) {

console.log("-- activateScreenSlider");

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;

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

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');

currentChips = slider.player\_chips;

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

if (dX > 0) {

updateChips = setInterval(function() {

game.placeBet(currentChips, player\_object);

}, 300);

} else {

updateChips = setInterval(function() {

game.returnBet(currentChips, 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(document, 'mouseup', slider.dropSlider);

removeEventSimple(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;

}

// ======= ======= ======= init ======= ======= =======

var game = new Game();

var display = new Display("gameDisplay");

var player1 = new Player(null, 0);

var player2 = new Player(null, 1);

var player3 = new Player(null, 2);

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

game.playerObjectsArray = [player1, player2, player3];

game.dealer = dealer;

var sequencer = new Sequencer();

display.housekeeping();

display.initGridElements();

sequencer.nextGameState("splash");

}