**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":

bet = {

name: "bet",

bg: null,

btn: [game.btnParams.retOnesBtn, game.btnParams.retFivesBtn, game.btnParams.retTensBtn, game.btnParams.playGameBtn, player.btnParams.betOnesBtn, player.btnParams.betFivesBtn, player.btnParams.betTensBtn],

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

input: null,

image: null

}

return bet;

break;

case "playGame":

playGame = {

name: "playGame",

bg: null,

btn: [game.btnParams.retOnesBtn, game.btnParams.retFivesBtn, game.btnParams.retTensBtn, player.btnParams.betOnesBtn, player.btnParams.betFivesBtn, player.btnParams.betTensBtn, player.btnParams.hitMeBtn, player.btnParams.holdMeBtn],

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

input: null,

image: null

}

return playGame;

break;

case "hitMe":

hitMe = {

name: "hitMe",

bg: null,

btn: [player.btnParams.hitMeBtn, player.btnParams.holdMeBtn],

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

input: null,

image: null

}

return hitMe;

break;

case "holdMe":

holdMe = {

name: "holdMe",

bg: null,

btn: [player.btnParams.hitMeBtn, player.btnParams.holdMeBtn],

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

input: null,

image: null

}

return holdMe;

break;

case "turnOver":

turnOver = {

name: "turnOver",

bg: null,

btn: [game.btnParams.retOnesBtn, game.btnParams.retFivesBtn, game.btnParams.retTensBtn, player.btnParams.betOnesBtn, player.btnParams.betFivesBtn, player.btnParams.betTensBtn, player.btnParams.hitMeBtn, player.btnParams.holdMeBtn],

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

input: null,

image: null

}

return turnOver;

break;

case "hitDealer":

hitDealer = {

name: "hitDealer",

bg: null,

btn: null,

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

input: null,

image: null

}

return hitDealer;

break;

case "doTheMath":

doTheMath = {

name: "doTheMath",

bg: null,

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

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

input: null,

image: null

}

return doTheMath;

break;

}

};

// ======= ======= ======= 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 = this.onesBank + this.fivesBank + this.tensBank;

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");

}

// ======= ======= ======= 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.tableCellsArray = 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 = "doTheMath";

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

}

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

function Display(whichDisplay) {

console.log('Display');

this.name = whichDisplay;

this.tableCellsArray = null;

}

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

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

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

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

Sequencer.prototype.loadStartGameState = function(prevNext) {

console.log("loadStartGameState");

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];

display.modifyGridRegion(nextItem, prevNext);

}

}

}

}

// ======= ======= ======= clearPrevGameState ======= ======= =======

Sequencer.prototype.clearPrevGameState = function() {

console.log("clearPrevGameState");

var tableRows = $(".row");

if (game.playerNamesArray.length > 0) {

var nextPlayer, cardCount, offsetC, whichCardObject, nextItem, indexCell;

var offsetR = 0;

// == clear previous hand for players and dealer

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

nextPlayer = game.playerObjectsArray[i];

cardCount = nextPlayer.hand.length;

whichCardObject = nextPlayer.textParams.pCards;

cardsRow = whichCardObject.iR;

cardsCol = whichCardObject.iC - cardCount + 1;

cardsCells = tableRows[cardsRow];

// == identify first card cell in table row and remove

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

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

}

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

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

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

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

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

$(indexCell).after(newCell);

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

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

}

}

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

nextPlayer.onesBet = 0;

nextPlayer.fivesBet = 0;

nextPlayer.tensBet = 0;

nextPlayer.hand = [];

nextPlayer.score = 0;

game.updateBetButtonText(nextPlayer);

game.updatePlayerScoreText(nextPlayer);

}

// == clear dealer cards

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

cardCount = game.dealer.hand.length;

whichCardObject = game.dealer.textParams.pCards;

cardsRow = whichCardObject.iR;

cardsCol = whichCardObject.iC;

cardsCells = tableRows[cardsRow];

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

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

}

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

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

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

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

$(indexCell).after(newCell);

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

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

}

}

}

game.dealer.hand = [];

game.dealer.score = 0;

game.updatePlayerScoreText(game.dealer);

}

}

// ======= ======= ======= 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';

// =======

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.pBank = nextPlayer.pBank + playerWinLoss;

nextPlayer.pBet\_1s = nextPlayer.pBet\_1s + nextPlayer.onesBet;

nextPlayer.pBet\_5s = nextPlayer.pBet\_5s + nextPlayer.fivesBet;

nextPlayer.pBet\_10s = nextPlayer.pBet\_10s + nextPlayer.tensBet;

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

winLossLabel = ' and won $';

nextPlayer.pBank = nextPlayer.pBank + playerWinLoss;

nextPlayer.pBet\_1s = nextPlayer.pBet\_1s + nextPlayer.onesBet;

nextPlayer.pBet\_5s = nextPlayer.pBet\_5s + nextPlayer.fivesBet;

nextPlayer.pBet\_10s = nextPlayer.pBet\_10s + nextPlayer.tensBet;

} else if (dealerScore < 22) {

winLossLabel = ' and lost $';

nextPlayer.pBank = nextPlayer.pBank - playerWinLoss;

nextPlayer.pBet\_1s = nextPlayer.pBet\_1s - nextPlayer.onesBet;

nextPlayer.pBet\_5s = nextPlayer.pBet\_5s - nextPlayer.fivesBet;

nextPlayer.pBet\_10s = nextPlayer.pBet\_10s - nextPlayer.tensBet;

} else {

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

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

}

playerWinLossString += nextName + "'s score: " + nextPlayer.score + winLossLabel + playerWinLoss + '\n\n';

}

alert(playerWinLossString);

}

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

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

console.log("activateButton");

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

console.log(" action: " + whichAction);

self = this;

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

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

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

whichIndexCell = event.target;

display.tooltips(whichIndexCell, "on");

});

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

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

whichIndexCell = event.target;

display.tooltips(whichIndexCell, "off");

});

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

switch(whichAction) {

case "updateGameGrid":

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

console.log("");

console.log("-- updateGameGrid");

self.updateGameGrid();

});

break;

case "saveNewPlayer":

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

console.log("");

console.log("-- saveNewPlayer");

game.saveNewPlayer();

});

break;

case "startGame":

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

console.log("");

console.log("-- startGame");

game.startGame();

});

break;

case "deal":

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

console.log("");

console.log("-- deal");

game.deal();

});

break;

case "betOne":

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

console.log("");

console.log("-- betOne");

game.placeBet("ones");

});

break;

case "betFive":

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

console.log("");

console.log("-- betFive");

game.placeBet("fives");

});

break;

case "betTen":

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

console.log("");

console.log("-- betTen");

game.placeBet("tens");

});

break;

case "retOne":

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

console.log("");

console.log("-- retOne");

game.returnBet("ones");

});

break;

case "retFive":

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

console.log("");

console.log("-- retFive");

game.returnBet("fives");

});

break;

case "retTen":

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

console.log("");

console.log("-- retTen");

game.returnBet("tens");

});

break;

case "playGame":

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

console.log("");

console.log("-- playGame");

game.playGame();

});

break;

case "hitMe":

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

console.log("");

console.log("-- hitMe");

game.hitMe();

});

break;

case "holdMe":

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

console.log("");

console.log("-- holdMe");

game.holdMe();

});

break;

case "newGame":

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

console.log("");

console.log("-- newGame");

game.newGame();

});

break;

}

}

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

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

console.log("deActivateButton");

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

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

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

}

// ======= ======= ======= updatePlayerGrid ======= ======= =======

Sequencer.prototype.updatePlayerGrid = function() {

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

// == get prev player info

var indexCell;

var prevPlayer = game.currentPlayer;

var prevPlayerHitHold = [prevPlayer.btnParams.hitMeBtn, prevPlayer.btnParams.holdMeBtn];

var prevPlayerBetBtns = [prevPlayer.btnParams.betOnesBtn, prevPlayer.btnParams.betFivesBtn, prevPlayer.btnParams.betTensBtn];

var prevPlayerIndex = prevPlayer.id;

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

// == remove prev player hitMe/holdMe and deactivate bet buttons

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

nextItem = prevPlayerHitHold[i];

display.modifyGridRegion(nextItem, "prev");

}

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

nextItem = prevPlayerBetBtns[j];

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

sequencer.deActivateButton(indexCell, nextItem.callback)

}

// == activate dealer if last player turn over

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

// == get next player info

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

var nextPlayerHitHold = [nextPlayer.btnParams.hitMeBtn, nextPlayer.btnParams.holdMeBtn];

var nextPlayerBetBtns = [nextPlayer.btnParams.betOnesBtn, nextPlayer.btnParams.betFivesBtn, nextPlayer.btnParams.betTensBtn];

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

// == add next player hitMe/holdMe and activate bet buttons

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

nextItem = nextPlayerHitHold[j];

display.modifyGridRegion(nextItem, "next");

}

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

nextItem = nextPlayerBetBtns[j];

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

sequencer.activateButton(indexCell, nextItem.callback)

}

}

}

// ======= ======= ======= updateGameGrid ======= ======= =======

Sequencer.prototype.updateGameGrid = function(whichState) {

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

// == get current state

var currentGameState = this.currentGameState;

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

var sequencerParams = this.sequencerParams(currentGameState);

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

var currentGameStateIndex = this.gameStatesArray.indexOf(this.currentGameState);

// == return to start game

if ((currentGameStateIndex == this.gameStatesArray.length - 1) && (!whichState)) {

currentGameState = this.gameStatesArray[0];

this.currentGameState = currentGameState;

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

// this.doTheMath();

this.clearPrevGameState();

this.loadStartGameState("next");

} else {

// == maintain state for player changes

if (whichState) {

currentGameState = whichState;

this.currentGameState = whichState;

} else {

// == advance to next game state

currentGameState = this.gameStatesArray[currentGameStateIndex + 1];

this.currentGameState = currentGameState;

}

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

var sequencerParams = this.sequencerParams(currentGameState);

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

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

var changeItemsArray = this.processItemArrays(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];

display.modifyGridRegion(nextItem, "prev");

}

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

nextItem = addItemsArray[j];

display.modifyGridRegion(nextItem, "next");

}

}

}

// ======= ======= ======= processItemArrays ======= ======= =======

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

// console.log("processItemArrays");

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];

}

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

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

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

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

Game.prototype.newGame = function() {

console.log("newGame");

sequencer.clearPrevGameState();

sequencer.updateGameGrid("deal");

}

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

Game.prototype.startGame = function() {

console.log("startGame");

this.loadDealer();

this.updatePlayerNames(this.dealer, 4);

this.currentPlayer = this.playerObjectsArray[0];

sequencer.updateGameGrid();

}

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

Game.prototype.playGame = function() {

console.log("playGame");

this.currentPlayer = this.playerObjectsArray[0];

sequencer.updateGameGrid();

}

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

Game.prototype.saveNewPlayer = function() {

console.log("saveNewPlayer");

var playerCount = this.playerNamesArray.length;

if (playerCount < 3) {

newPlayer = this.playerObjectsArray[playerCount];

this.playerNamesArray.push(newPlayer.name);

var playerCount = this.playerNamesArray.length;

this.loadNewPlayer(newPlayer);

this.updatePlayerNames(newPlayer, playerCount);

}

if (playerCount == 1) {

sequencer.updateGameGrid();

}

if (playerCount == 3) {

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

this.loadDealer();

this.updatePlayerNames(this.dealer, 4);

this.currentPlayer = game.playerObjectsArray[0];

sequencer.updateGameGrid();

}

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

}

// ======= ======= ======= loadNewPlayer ======= ======= =======

Game.prototype.loadNewPlayer = function() {

console.log("loadNewPlayer");

var playerCount = game.playerNamesArray.length;

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

var playerParamsArray = [currentPlayer.bgParams.borderH, currentPlayer.bgParams.borderV, currentPlayer.btnParams.betOnesBtn, currentPlayer.btnParams.betFivesBtn, currentPlayer.btnParams.betTensBtn, 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");

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

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

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;

}

// ======= ======= ======= loadDealer ======= ======= =======

Game.prototype.loadDealer = function() {

// console.log("loadDealer");

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;

}

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

Game.prototype.placeBet = function(whichBet) {

console.log("placeBet");

var onesBank = game.currentPlayer.onesBank;

var fivesBank = game.currentPlayer.fivesBank;

var tensBank = game.currentPlayer.tensBank;

var onesBet = game.currentPlayer.onesBet;

var fivesBet = game.currentPlayer.fivesBet;

var tensBet = game.currentPlayer.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) {

game.currentPlayer.onesBank = onesBank;

game.currentPlayer.fivesBank = fivesBank;

game.currentPlayer.tensBank = tensBank;

game.currentPlayer.onesBet = onesBet;

game.currentPlayer.fivesBet = fivesBet;

game.currentPlayer.tensBet = tensBet;

game.currentPlayer.totalBank = game.currentPlayer.totalBank - (onesBet + fivesBet + tensBet);

game.updateBetButtonText(game.currentPlayer);

} else {

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

}

}

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

Game.prototype.returnBet = function(whichBet) {

// console.log("returnBet");

var onesBank = game.currentPlayer.onesBank;

var fivesBank = game.currentPlayer.fivesBank;

var tensBank = game.currentPlayer.tensBank;

var onesBet = game.currentPlayer.onesBet;

var fivesBet = game.currentPlayer.fivesBet;

var tensBet = game.currentPlayer.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) {

game.currentPlayer.onesBank = onesBank;

game.currentPlayer.fivesBank = fivesBank;

game.currentPlayer.tensBank = tensBank;

game.currentPlayer.onesBet = onesBet;

game.currentPlayer.fivesBet = fivesBet;

game.currentPlayer.tensBet = tensBet;

game.currentPlayer.totalBank = game.currentPlayer.totalBank + (onesBet + fivesBet + tensBet);

game.updateBetButtonText(game.currentPlayer);

} else {

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

}

}

// ======= ======= ======= 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.dealNextCard(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!");

nextPlayer.totalBank = nextPlayer.totalBank - (nextPlayer.onesBank + nextPlayer.fivesBank + nextPlayer.tensBank);

this.turnOver();

}

}

}

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

Game.prototype.holdMe = function() {

console.log("holdMe");

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

this.turnOver();

}

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

Game.prototype.hitDealer = function() {

console.log("hitDealer");

// == hit dealer again or end hand

if (dealer.score < 18) {

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 dealer score

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

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

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

// == pause between dealer cards

// setTimeout(function() {

// console.log("dealer pause");

// }, 2000);

if (dealer.score < 18) {

this.hitDealer();

} else {

sequencer.updateGameGrid();

sequencer.doTheMath();

}

} else {

sequencer.updateGameGrid();

sequencer.doTheMath();

}

}

// ======= ======= ======= 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.updatePlayerGrid();

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

this.currentPlayer = nextPlayer;

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

} else {

sequencer.updatePlayerGrid();

this.currentPlayer = this.dealer;

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

sequencer.updateGameGrid("hitDealer");

this.hitDealer();

}

}

// ======= ======= ======= 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 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.dealNextCard(nextPlayer);

}

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

if (nextPlayer.score > 21) {

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;

break;

}

}

}

// ======= instant winner

if (nextPlayer.score == 21) {

winnersArray.push(nextPlayer);

}

this.updatePlayerScoreText(nextPlayer);

var playerParamsArray = [nextPlayer.textParams.pBet\_1s, nextPlayer.textParams.pBet\_5s, nextPlayer.textParams.pBet\_10s];

// var playerParamsArray = [nextPlayer.textParams.pBet\_10s];

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

nextItem = playerParamsArray[j];

// console.log(" nextItem.name: " + nextItem.name);

if (nextItem != null) {

display.modifyGridRegion(nextItem, "next");

}

}

display.tooltips("place bets", "on");

}

// ======= 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.dealNextCard(dealer);

}

this.updatePlayerScoreText(dealer);

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

if (winnersArray.length > 0) {

// calculateWinner();

} else {

this.activePlayer = 1;

sequencer.updateGameGrid();

}

}

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

}

// ======= ======= ======= dealNextCard ======= ======= =======

Game.prototype.dealNextCard = function(whichPlayer) {

console.log("dealNextCard");

var whichMerge, cardDivString;

var whichCardObject = whichPlayer.textParams.pCards;

var whichName = whichCardObject.name;

var whichClass = whichCardObject.class;

var cardCount = whichPlayer.hand.length;

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

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

offsetC = cardCount - 1;

} else {

offsetC = -(cardCount - 1);

}

offsetR = 0;

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

indexCell = display.mergeRegion(whichCardObject, offsetR, offsetC);

} else {

indexCell = display.unMergeRegion(whichCardObject);

}

cardDivString = "<div class='flip-container " + whichClass + "' ontouchstart='this.classList.toggle('hover');'>";

cardDivString += "<div class='flipper'><div class='front'><p class='cardText'>" + cardValue + "</p></div>";

cardDivString += "<div class='back'><p class='cardText'>" + cardValue + "</p></div></div></div>";

$(indexCell).append(cardDivString);

}

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

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

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

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

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

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

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

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

// whichPlayer.totalBank = whichPlayer.totalBank - (whichPlayer.onesBet + whichPlayer.fivesBet + whichPlayer.tensBet);

$("#" + whichPlayer.textParams.pBank.name).text("$" + 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 = this.mergeRegion(whichItem);

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

indexCell = this.unMergeRegion(whichItem);

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

indexCell = this.deselectTableCells(whichItem);

} else {

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) {

$(indexCell).text(whichValue);

sequencer.activateButton(indexCell, whichItem.callback)

// console.log(" whichItem.callback: " + whichItem.callback);

} else {

sequencer.deActivateButton(indexCell, whichItem.callback)

}

break;

case "text":

$(indexCell).text(whichValue);

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

}

}

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

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

console.log("mergeRegion");

if (!offsetR) { offsetR = 0 };

if (!offsetC) { offsetC = 0 };

var offsetR = parseInt(offsetR);

var offsetC = parseInt(offsetC);

var indexCell = this.tableCellsArray[whichItem.iR + offsetR][whichItem.iC + offsetC];

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

console.log(" whichItem.iR/C: " + whichItem.iR + "/" + whichItem.iC);

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

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

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

nextCell = this.tableCellsArray[whichItem.iR + offsetR + j][whichItem.iC + offsetC + i];

if (!((i == 0) && (j == 0))) {

$(nextCell).remove();

}

}

}

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

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

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

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

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

}

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

return indexCell;

}

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

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

console.log("unMergeRegion");

if (!offsetR) { offsetR = 0 };

if (!offsetC) { offsetC = 0 };

var offsetR = parseInt(offsetR);

var offsetC = parseInt(offsetC);

var indexR = parseInt(whichItem.iR);

var indexC = parseInt(whichItem.iC);

var rowCell = indexR + offsetR;

var colCell = indexC + offsetC;

var cellString = rowCell + "/" + colCell;

var indexCell = this.tableCellsArray[rowCell][colCell];

$(indexCell).attr("id", cellString);

$(indexCell).attr("colSpan", 1);

$(indexCell).attr("rowSpan", 1);

$(indexCell).text("");

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

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

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

if (i != 0) {

// var tableRows = $(".row");

// nextRow = tableRows[indexR + offsetR];

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

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

$(indexCell).after(newCell);

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

$(newCell).attr("id", (rowCell+ j) + "-" + (colCell + i));

}

if ((j > 0) && (i == 0)) {

indexCell = this.tableCellsArray[rowCell+ j][colCell - 1];

$(indexCell).addClass("ones");

// var tableRows = $(".row");

// nextRow = tableRows[indexR + offsetR];

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

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

$(indexCell).after(newCell);

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

$(newCell).attr("id", (rowCell+ j) + "-" + (colCell + i));

}

}

}

// == remove content (e.g. input element) if required

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

if ($(indexCell).children().length > 0) {

console.log(" $(indexCell).children().length: " + $(indexCell).children().length);

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

$(indexCell).after(newCell);

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

$(indexCell).remove();

}

// return newCell;

}

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

Display.prototype.selectTableCells = function(whichItem) {

// console.log("selectTableCells");

var indexCell = this.tableCellsArray[whichItem.iR][whichItem.iC];

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

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

nextCell = this.tableCellsArray[whichItem.iR + j][whichItem.iC + i];

// $(nextCell).attr("colSpan", 1);

// $(nextCell).attr("rowSpan", 1);

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

}

}

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

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

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

return indexCell;

}

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

Display.prototype.deselectTableCells = function(whichItem) {

console.log("deselectTableCells");

var indexCell = this.tableCellsArray[whichItem.iR][whichItem.iC];

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

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

var rowCell = whichItem.iR + j;

var colCell = whichItem.iC + i;

var rowStrg = rowCell.toString();

var colStrg = colCell.toString();

nextCell = this.tableCellsArray[rowCell][colCell];

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

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

$(indexCell).attr("id", (rowStrg + "-" + colStrg));

}

}

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

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

$(indexCell).empty();

return indexCell;

}

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

Display.prototype.tooltips = function(whichItem, onOff) {

// console.log("tooltips");

var nextTooltip;

if (whichItem !== null && typeof whichItem === 'object') {

if (whichItem.tooltip) {

nextTooltip = whichItem.tooltip;

} else if (whichItem.value) {

nextTooltip = whichItem.value;

} else {

nextTooltip = whichItem.id;

}

} else {

nextTooltip = whichItem;

}

var tooltipIndexCell = this.tableCellsArray[9][12];

if (onOff == "on") {

$(tooltipIndexCell).text(nextTooltip);

} else {

$(tooltipIndexCell).text("");

}

}

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

Display.prototype.initGridElements = function() {

// console.log("initGridElements");

var tableCols;

var tableCellsArray = [];

var tableRows = $(".row");

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

nextRow = tableRows[i];

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

tableCellsArray.push(tableCols);

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

nextCell = tableCols[j];

$(nextCell).attr("id", i + "-" + j);

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

}

}

this.tableCellsArray = tableCellsArray;

}

// ======= ======= ======= 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.initGridElements();

sequencer.updateGameGrid();

}