using UnityEngine;

using System.Collections;

using System.Collections.Generic;

/// <summary> ##################################

///

/// NOTICE :

/// This script is a setup function to customize the gameBoard looks during gameplay.

///

/// DO NOT TOUCH UNLESS REQUIRED

///

/// </summary> ##################################

public class JSFBoardLayout : MonoBehaviour {

public JSFGameManager gm;

public bool randomOnStart = false;

public bool useSelector = false;

public bool showHexGrid = true;

public bool hidePanel1 = false;

public bool hidePanel2 = false;

public bool hidePanel3 = false;

public JSFPanelDefinition[] panelScripts; // panel reference scripts to be used

public JSFPieceDefinition[] pieceScripts; // piece reference scripts to be used

// these are max values for a panel type during randomize

public JSFPanelLimit[] randomPanelLimit;

[System.Serializable]

public class JSFPanelLimit {

[HideInInspector] public string name;

public int randomLimit;

}

// these are the counter for the max panels above

public int[] randomPanelCount;

// these are the texture array for representing the panels

public TextureArray[] panelEditVisuals;

// these are the texture array for representing the pieces

public TextureArray[] pieceEditVisuals;

[System.Serializable]

public class TextureArray {

[HideInInspector] public string name;

public Texture texture;

}

public int[] panelArray; // the PanelType[] converted to be and int reference so that it is serialisable

public int[] pStrength; // the strength of the panel assigned

public int[] pieceArray; // the piece type to be assigned during gameplay.

public int[] colorArray; // the manual skin to assign ( semi randomized )

// these textures are for inspector visuals only - does not effect gameplay

// paired and referenced by "BoardSetup" GUI script

public Vector2 scrollPos; // for the scrollbar

public Vector2 scrollPos2; // for the scrollbar

public Vector2 scrollPos3; // for the scrollbar

// for post manual color pre-start match

bool[,] isManual;

// weighted colors

public List<JSFWeightedLayout> colorWeight;

public JSFWeightedLayout displayedWeight;

[System.Serializable]

public class JSFWeightedLayout {

[HideInInspector] public string name;

public bool useWeights = false;

[Range(0,100)]

public List<int> weights = new List<int>(9);

}

int totalWeight = 0; // variable to hold the total weights

int selected = 0; // a variable to store the selected random range for weights

int addedWeight = 0; // a variable to store the cumulative added weight for calculations

// called by GameManager for panel setup during pre-game init

public void setupGamePanels(){

if(randomOnStart){

randomize();

}

// code below sets up the layout as per shown in the inspector

int count = 0;

for(int y = gm.boardHeight -1 ; y >= 0 ; y--){

for(int x = 0; x < gm.boardWidth; x++){

int num = panelArray[count];

// create the panel and set the type by JSFPanelDefinition selected

gm.board[x,y].panel = new JSFBoardPanel ( panelScripts[num], pStrength[count]-1, gm.board[x,y] );

count++;

}

}

}

public void setupGamePieces(){

// code below sets up the pieces as per shown in the inspector

// color randomization

int randomColor = Random.Range(0,9);

// save the manual color reference

isManual = new bool[gm.boardWidth,gm.boardHeight]; // size of the board

int count = 0;

for(int y = gm.boardHeight -1 ; y >= 0 ; y--){

for(int x = 0; x < gm.boardWidth; x++){

// init default value

isManual[x,y] = false;

// set the piece type first

if(pieceArray[count] != 0 ){

gm.board[x,y].setSpecialPiece( pieceScripts[pieceArray[count] ] );

isManual[x,y] = true; // manual override is true

}

// then set the color (if defined...)

if(colorArray[count] != 0 && gm.board[x,y].isFilled && !gm.board[x,y].piece.pd.isSpecial){

gm.board[x,y].piece.slotNum = (colorArray[count] + randomColor) % gm.NumOfActiveType ;

isManual[x,y] = true; // manual override is true

} else if(gm.board[x,y].isFilled && !gm.board[x,y].piece.pd.isSpecial && colorWeight[count].useWeights){ // weights distribution functionality

// run once weighted calculation...

totalWeight = 0; // reset the value first...

for(int z = 0; z < gm.NumOfActiveType; z++){ // adds all available skin based on active type

if(z < colorWeight[count].weights.Count ){ // ensure we have allocated weights and add to the list

totalWeight += colorWeight[count].weights[z];

}

}

selected = Random.Range(1,totalWeight+1); // the selected weight by random

addedWeight = 0; // resets the value first...

for(int z = 0; z < colorWeight[count].weights.Count; z++){

addedWeight+= colorWeight[count].weights[z];

if(colorWeight[count].weights[z] > 0 && addedWeight > selected){

gm.board[x,y].piece.slotNum = z; // found the skin we want to use based on the selected weight

break;

}

}

isManual[x,y] = true; // manual override is true

}

count++;

}

}

}

// cycles through each panel type based on the "Panel Definition" scripts. any changes there will reflect here.

public void togglePanel(int position,int val){

panelArray[position] = (panelArray[position] + val) % panelScripts.Length;

if(panelArray[position] < 0){

panelArray[position] = panelScripts.Length-1; // loop backwards

}

setDefaultStrength(position);

}

// set panel directly

public void setPanel(int position,int val){

panelArray[position] = val;

setDefaultStrength(position);

}

// cycles through each piece type based on the "Piece Definition" scripts. any changes there will reflect here.

public void togglePiece(int position,int val){

pieceArray[position] = (pieceArray[position] + val) % pieceScripts.Length;

if(pieceArray[position] < 0){

pieceArray[position] = pieceScripts.Length-1; // loop backwards

}

}

// set piece directly

public void setPiece(int position,int val){

pieceArray[position] = val;

}

// cycles through each piece type based on the "Piece Definition" scripts. any changes there will reflect here.

public void toggleColor(int position,int val){

colorArray[position] = (colorArray[position] + val) % (gm.NumOfActiveType + 1);

if(colorArray[position] < 0){

colorArray[position] = gm.NumOfActiveType; // loop backwards

}

}

void setDefaultStrength(int position){

for(int x = 0; x < panelScripts.Length; x++){ // search the array

if(panelScripts[x] == panelScripts[panelArray[position]] ){ // if found the correct array

pStrength[position] = panelScripts[x].defaultStrength; // return the associated default strength

}

}

}

// just a simple function to reset everything!

public void resetMe(){

int count = 0;

for(int x = 0; x < gm.boardWidth; x++){

for(int y = 0; y < gm.boardHeight; y++){

panelArray[count] = 0;

pieceArray[count] = 0;

colorArray[count] = 0;

colorWeight = new List<JSFWeightedLayout>();

setDefaultStrength(count);

count++;

}

}

}

// just a simple function to reset all pieces to BASIC type

public void resetPieceOnly(){

int count = 0;

for(int x = 0; x < gm.boardWidth; x++){

for(int y = 0; y < gm.boardHeight; y++){

pieceArray[count] = 0;

count++;

}

}

}

// just a simple function to reset all piece color type to random

public void resetColorOnly(){

int count = 0;

for(int x = 0; x < gm.boardWidth; x++){

for(int y = 0; y < gm.boardHeight; y++){

colorArray[count] = 0;

colorWeight = new List<JSFWeightedLayout>();

count++;

}

}

}

// just a simple function to reset all panels to basic

public void resetPanelOnly(){

int count = 0;

for(int x = 0; x < gm.boardWidth; x++){

for(int y = 0; y < gm.boardHeight; y++){

panelArray[count] = 0;

setDefaultStrength(count);

count++;

}

}

}

// just a simple function to click all panels

public void clickAll(int val){

int count = 0;

for(int x = 0; x < gm.boardWidth; x++){

for(int y = 0; y < gm.boardHeight; y++){

togglePanel(count,val);

count++;

}

}

}

// just a simple function to randomize all panels

public void randomize(){

int count = 0;

// reset to initial count of panels

for(int x = 0; x < randomPanelCount.Length; x++){

randomPanelCount[x] = 0;

}

for(int x = 0; x < gm.boardWidth; x++){

for(int y = 0; y < gm.boardHeight; y++){

panelArray[count] = generateNumber(); // generate and assigns a random number

setDefaultStrength(count);

count++;

}

}

}

// an internal function to generate a number but also keep within the max limits

// of the panels defined.

int generateNumber(){

int generated = Random.Range(0, panelScripts.Length);

if(generated > 0){

if(Random.Range(0, 2) == 0){ // 1/2 chance to make a special panel

if( randomPanelCount[generated] < randomPanelLimit[generated].randomLimit ){

randomPanelCount[generated]++;

return generated;

}

}

}

return 0; // if nothing happens above, return default panel

}

}