public class JSFBoardLayoutEditor : Editor {

\*\*\* other unchanged variables \*\*\*

SerializedProperty randomPanelLimit;

SerializedProperty randomPanelCount;

SerializedProperty panelEditVisuals;

SerializedProperty pieceEditVisuals;

SerializedProperty colorWeight;

public PanelDefinition[] scripts;

public PieceDefinition[] pieces;

public JSFPanelDefinition[] scripts;

public JSFPieceDefinition[] pieces;

public void initMe(){

script = new SerializedObject(target);

bl = ((JSFBoardLayout) target);

if(bl.gm == null){

bl.gm = bl.GetComponent<JSFGameManager>(); // assign gm ref if needed

}

// scripts to use

scripts = bl.gm.panelManager.GetComponents<JSFPanelDefinition>();

pieces = bl.gm.pieceManager.GetComponents<JSFPieceDefinition>();

// visual textures

panelEditVisuals = script.FindProperty("panelEditVisuals");

pieceEditVisuals = script.FindProperty("pieceEditVisuals");

// weights

colorWeight = script.FindProperty("colorWeight");

// random on start boolean

randomOnStart = script.FindProperty("randomOnStart");

// show hex grid arrangement

showHexGrid = script.FindProperty("showHexGrid");

// use Selector boolean

useSelector = script.FindProperty("useSelector");

// max panels during randomization

randomPanelLimit = script.FindProperty("randomPanelLimit");

randomPanelCount = script.FindProperty("randomPanelCount");

// gui usage booleans

hidePanel1 = script.FindProperty("hidePanel1");

hidePanel2 = script.FindProperty("hidePanel2");

hidePanel3 = script.FindProperty("hidePanel3");

// board GUI setups

panelArray = script.FindProperty("panelArray"); // for the button arrays

pStrength = script.FindProperty("pStrength"); // for the strength fields

pieceArray = script.FindProperty("pieceArray"); // for the piece types

colorArray = script.FindProperty("colorArray"); // for the piece color type

scrollPos = script.FindProperty("scrollPos"); // for the scrollbar to conpensate for big boards

scrollPos2 = script.FindProperty("scrollPos2"); // for the scrollbar to conpensate for big boards

scrollPos3 = script.FindProperty("scrollPos3"); // for the scrollbar to conpensate for big boards

}

public override void OnInspectorGUI () {

initMe(); // initialize the required serialized stuff

script.Update();

setRequiredValues(); // set the variables with the correct value - called after script.Update()

drawLayoutTable(); // shows the custom tables

script.ApplyModifiedProperties();

EditorUtility.SetDirty(bl); // refresh the changes

}

void setRequiredValues(){

if(scripts.Length > 0 ){

bl.panelScripts = scripts;

// auto adjust array sizes for all arrays according to the number of available scripts

panelEditVisuals.arraySize = randomPanelLimit.arraySize = randomPanelCount.arraySize = bl.panelScripts.Length;

pieceEditVisuals.arraySize = bl.pieceScripts.Length;

script.ApplyModifiedProperties();

script.Update();

} else {

Debug.LogError("No panels found... go to PanelsManager and add your panels!");

}

if(pieces.Length > 0 ){

bl.pieceScripts = pieces;

} else {

Debug.LogError("No Piece-type found... go to PiecesManager and add your pieces!");

}

colorArray.arraySize = colorWeight.arraySize = pieceArray.arraySize = panelArray.arraySize = pStrength.arraySize = bl.gm.boardWidth \* bl.gm.boardHeight;

script.ApplyModifiedProperties();

script.Update();

}

void drawLayoutTable()

{

if( GUILayout.Button( "Launch Window", GUILayout.Width(250) ) ){

EditorWindow.GetWindow (typeof (JSFBoardLayoutWindow),false, "Board Setup");

}

if( GUILayout.Button( "Show/Hide Panel 1", GUILayout.Width(250) ) ){

hidePanel1.boolValue = !hidePanel1.boolValue;

}

if(!hidePanel1.boolValue){

drawPanel1();

}

if( GUILayout.Button( "Show/Hide Panel 2", GUILayout.Width(250) ) ){

hidePanel2.boolValue = !hidePanel2.boolValue;

}

if(!hidePanel2.boolValue){

drawPanel2();

}

if( GUILayout.Button( "Show/Hide NOTES", GUILayout.Width(250) ) ){

hidePanel3.boolValue = !hidePanel3.boolValue;

}

if(!hidePanel3.boolValue){

drawPanel3();

}

// custom inspector auto naming for ease of use.

for(int x = 0; x < panelEditVisuals.arraySize;x++){

bl.panelEditVisuals[x].name = scripts[x].GetType().Name + "'s";

panelEditVisuals.GetArrayElementAtIndex(x).isExpanded = true;

}

for(int x = 0; x < pieceEditVisuals.arraySize;x++){

bl.pieceEditVisuals[x].name = pieces[x].GetType().Name + "'s";

pieceEditVisuals.GetArrayElementAtIndex(x).isExpanded = true;

}

for(int x = 0; x < randomPanelLimit.arraySize;x++){

bl.randomPanelLimit[x].name = scripts[x].GetType().Name + "'s";

randomPanelLimit.GetArrayElementAtIndex(x).isExpanded = true;

}

EditorGUILayout.LabelField("\n\*If enabled, will generate a random board" +

"\nwhenever a new game starts disregarding the below layout.", GUILayout.Height(45));

EditorGUILayout.BeginHorizontal();

EditorGUILayout.PropertyField(randomOnStart, new GUILayoutOption[] {GUILayout.MinHeight(0),GUILayout.Width(200) });

EditorGUILayout.PropertyField(showHexGrid, new GUIContent("Hex Layout in Hex Mode"), GUILayout.MinWidth(0));

EditorGUILayout.EndHorizontal();

EditorGUILayout.BeginHorizontal();

EditorGUILayout.PropertyField(useSelector, new GUILayoutOption[] {GUILayout.Height(30),GUILayout.Width(200) });

EditorGUILayout.LabelField("<-- hold 'ctrl-btn' to alternate between the modes during clicks", GUILayout.MinWidth(0));

EditorGUILayout.EndHorizontal();

if(panelArray != null && !Application.isPlaying){

// selector switching for easy selector usage via ctrl-btn to alternate between itself

if(Event.current.control == true){

selectorSwitch = !bl.useSelector;

} else {

selectorSwitch = bl.useSelector;

}

GUILayoutOption[] scrollParams = {GUILayout.MinHeight(200), GUILayout.MaxHeight(888)};

scrollPos.vector2Value = EditorGUILayout.BeginScrollView(scrollPos.vector2Value,scrollParams);

int count = 0;

GUILayoutOption[] layoutParams = {GUILayout.Width(35),GUILayout.Height(35)};

EditorGUILayout.BeginHorizontal();

for(int x = 0; x < Mathf.Min( bl.gm.boardWidth, 20); x++){

EditorGUILayout.BeginVertical();

// hexagon displacement

if(x%2 == 1 && bl.showHexGrid && bl.gm.boardType == JSFBoardType.Hexagon){

EditorGUILayout.LabelField("", new GUILayoutOption[] {GUILayout.Height(25),GUILayout.Width(0) });

}

for(int y = 0; y < Mathf.Min( bl.gm.boardHeight, 20) ; y++){

count = x + (bl.gm.boardWidth\*y);

EditorGUILayout.BeginHorizontal();

// the strength field

pStrength.GetArrayElementAtIndex(count).intValue =

EditorGUILayout.IntField(pStrength.GetArrayElementAtIndex(count).intValue,

new GUILayoutOption[] {GUILayout.Width(30),GUILayout.Height(18)});

// color selection

GUI.backgroundColor = bColor[ bl.colorArray[count] ]; // render the color to the gui button

if(GUILayout.Button( "", new GUILayoutOption[] {GUILayout.ExpandWidth(false)} ) ){

if (Event.current.button == 0)

bl.toggleColor(count,1);

else bl.toggleColor(count,-1);

}

GUI.backgroundColor = Color.white; // back to normal color for other GUIs

// weights selection

if(bl.colorArray[count] == 0){ // show weights option if no color grouping...

if(bl.colorWeight[count].useWeights){

GUI.backgroundColor = Color.green;

}

if(GUILayout.Button( "w", new GUILayoutOption[] {GUILayout.Width(22),GUILayout.Height(18)} ) ){

EditorWindow.GetWindow (typeof (JSFBoardLayoutWeightsWindow),false, "Set Weights");

JSFBoardLayoutWeightsWindow.setDisplayType(count);

}

}

GUI.backgroundColor = Color.white; // back to normal color for other GUIs

EditorGUILayout.EndHorizontal();

EditorGUILayout.BeginHorizontal();

// panels selection

int num = panelArray.GetArrayElementAtIndex(count).intValue;

if(num > bl.panelEditVisuals.Length-1 ) { // script out of bounds, auto-fix!

bl.togglePanel(count,-1);

num = bl.panelArray[count];

}

if(bl.panelEditVisuals[num] != null && bl.panelEditVisuals[num].texture != null){ // assigned texture version

if(GUILayout.Button( bl.panelEditVisuals[num].texture,GUI.skin.box, layoutParams ) ){

if(selectorSwitch){

EditorWindow.GetWindow (typeof (JSFBoardLayoutChooserWindow),false, "Choose Type");

JSFBoardLayoutChooserWindow.setDisplayType(count,false);

} else{

if (Event.current.button == 0)

bl.togglePanel(count,1);

else bl.togglePanel(count,-1);

}

}

} else { // script name version

if(GUILayout.Button( bl.panelScripts[num].GetType().Name.Substring(0,

Mathf.Min (5,bl.panelScripts[num].name.Length)).ToString().Replace("Panel",""),

new GUILayoutOption[] {GUILayout.ExpandWidth(false)} ) ){

if(selectorSwitch){

EditorWindow.GetWindow (typeof (JSFBoardLayoutChooserWindow),false, "Choose Type");

JSFBoardLayoutChooserWindow.setDisplayType(count,false);

} else{

if (Event.current.button == 0)

bl.togglePanel(count,1);

else bl.togglePanel(count,-1);

}

}

}

// pieces selection

int type = pieceArray.GetArrayElementAtIndex(count).intValue;

if(type > bl.pieceScripts.Length-1 ) { // script out of bounds, auto-fix!

bl.togglePiece(count,-1);

type = bl.pieceArray[count];

}

if(bl.pieceEditVisuals[type] != null && bl.pieceEditVisuals[type].texture != null){ // assigned texture version

if(GUILayout.Button( bl.pieceEditVisuals[type].texture, GUI.skin.label, layoutParams ) ){

if(selectorSwitch){

EditorWindow.GetWindow (typeof (JSFBoardLayoutChooserWindow),false, "Choose Type");

JSFBoardLayoutChooserWindow.setDisplayType(count,true);

} else{

if (Event.current.button == 0)

bl.togglePiece(count,1);

else bl.togglePiece(count,-1);

}

}

} else if(bl.pieceScripts[type] != null){ // assigned texture version

if(GUILayout.Button( pieces[type].GetType().ToString().Replace("Piece",""),

new GUILayoutOption[] {GUILayout.ExpandWidth(false)} ) ){

if(selectorSwitch){

EditorWindow.GetWindow (typeof (JSFBoardLayoutChooserWindow),false, "Choose Type");

JSFBoardLayoutChooserWindow.setDisplayType(count,true);

} else{

if (Event.current.button == 0)

bl.togglePiece(count,1);

else bl.togglePiece(count,-1);

}

}

}

EditorGUILayout.EndHorizontal();

EditorGUILayout.Space();

}

EditorGUILayout.EndVertical();

}

EditorGUILayout.EndHorizontal();

EditorGUILayout.Space(); EditorGUILayout.Space(); EditorGUILayout.Space();

EditorGUILayout.EndScrollView();

}

GUILayoutOption[] scrollParams2 = {GUILayout.MinHeight(90)};

scrollPos2.vector2Value = EditorGUILayout.BeginScrollView(scrollPos2.vector2Value,scrollParams2);

// the bottom buttons for easy board modifications

EditorGUILayout.BeginHorizontal();

GUILayoutOption[] layoutParams2 = {GUILayout.Width(80),GUILayout.Height(30)};

if( GUILayout.Button( "Reset All", layoutParams2 ) ){

bl.resetMe();

}

if( GUILayout.Button( "Click All", layoutParams2 ) ){

if (Event.current.button == 0)

bl.clickAll(1);

else bl.clickAll(-1);

}

if( GUILayout.Button( "Randomize!", layoutParams2 ) ){

bl.randomize();

}

EditorGUILayout.EndHorizontal();

EditorGUILayout.BeginHorizontal();

layoutParams2 = new GUILayoutOption[] {GUILayout.Width(90),GUILayout.Height(30)};

if( GUILayout.Button( "Reset Pieces", layoutParams2 ) ){

bl.resetPieceOnly();

}

if( GUILayout.Button( "Reset Color", layoutParams2 ) ){

bl.resetColorOnly();

}

if( GUILayout.Button( "Reset Panels", layoutParams2 ) ){

bl.resetPanelOnly();

}

EditorGUILayout.EndHorizontal();

EditorGUILayout.EndScrollView();

}

void drawPanel1(){

// scroll view

GUILayoutOption[] scrollParams = {GUILayout.MinHeight(200), GUILayout.MaxHeight(1000)};

scrollPos3.vector2Value = EditorGUILayout.BeginScrollView(scrollPos3.vector2Value,scrollParams);

EditorGUILayout.LabelField("These textures below is just for visuals on the\n" +

"layout board and has no effect on the game itself", GUILayout.Height(30));

EditorGUILayout.PropertyField(panelEditVisuals, true);

EditorGUILayout.PropertyField(pieceEditVisuals, true);

EditorGUILayout.EndScrollView(); // scroll view

}

void drawPanel2(){

EditorGUILayout.LabelField("\n\*Properties below are the max amount of the panels " +

"\ngenerated during \"randomize!\"/\'Random on Start\"", GUILayout.Height(45));

EditorGUILayout.PropertyField(randomPanelLimit, true);

}

void drawPanel3(){

EditorGUILayout.LabelField("\n\*Click the buttons below to cycle through each panel type.." +

"\nNOTE: layout set below does not work when \"Random on Start\" is enabled~!" +

"\nNOTE 2: The numbers represents the strength of the panel (hits it take before destroyed)" +

"\n \*\*\* 0 means destroyed; 1 = takes one hit / Empty & Basic cannot be destroyed." +

"\nNOTE 3 : panel looks will follow the panel skin defined in their respective PanelDefinition script." +

"\n \*\*\* e.g. Rock strength 1 will use rock skin array element 0; and so on... " +

"\n \*\*\*\* if strength > prefab array size, it will use the last element defined." +

"\n \*\*\*\*\* e.g., Rock strength 10; array size 5; will use element 4 for strength 10 until strength 5" +

"\n NOTE 4 : Board color can be manually assigned and/or specified... " +

"\n \*\*\* The color order only specifies the grouping, not the actual skin used." +

"\n \*\*\*\* Non colored box would indicate random selected group. Color Weights can be assigned." +

"\n \*\*\*\*\* Improper assigning can lead to pre-start matches eventhough that option is selected."

, GUILayout.Height(190));

}

}