Lab 03 Report - Gipson

# Introduction

Designers need clean interfaces that are easy to interrupt and modify to help increase production speeds.

# Methods

# Conclusion

I learned how to create custom Drawers which are how data is displayed aswell as custom editors, which is how drawers are displays.

# Code

Below is the lab code with my modifications for section one.

using UnityEngine;

using System.Collections;

using UnityEditor;

/// <summary>

/// Author: Matt Gipson

/// Contact: Deadwynn@gmail.com

/// Domain: www.livingvalkyrie.com

///

/// Description: ObjectTypesDrawer

/// </summary>

[CustomPropertyDrawer(typeof (ObjectTypes))]

public class ObjectTypesDrawer : PropertyDrawer {

#region Fields

ObjectTypes thisObject;

float extraHeight = 55f;

int shouldSolidMove = 0;

#endregion

public override void OnGUI(Rect position, SerializedProperty property, GUIContent label) {

EditorGUI.BeginProperty(position, label, property);

#region get properties

SerializedProperty objectType = property.FindPropertyRelative("type");

SerializedProperty breakablePoints = property.FindPropertyRelative("breakablePoints");

SerializedProperty solidMoving = property.FindPropertyRelative("solidMoving");

SerializedProperty solidStart = property.FindPropertyRelative("solidStart");

SerializedProperty solidEnd = property.FindPropertyRelative("solidEnd");

SerializedProperty damageType = property.FindPropertyRelative("damageType");

SerializedProperty damageAmount = property.FindPropertyRelative("damageAmount");

SerializedProperty healingType = property.FindPropertyRelative("healingType");

SerializedProperty healingPickupType = property.FindPropertyRelative("healingPickupType");

SerializedProperty healingAmount = property.FindPropertyRelative("healingAmount");

#endregion

Rect objectTypeDisplay = new Rect(position.x, position.y, position.width, 15f);

EditorGUI.PropertyField(objectTypeDisplay, objectType, GUIContent.none);

switch ((ObjectType) objectType.enumValueIndex) {

case ObjectType.BREAKABLE:

Rect breakableRect = new Rect(position.x, position.y + 17, position.width, 15f);

EditorGUI.PropertyField(breakableRect, breakablePoints);

break;

case ObjectType.DAMAGING:

float offset = position.x;

Rect damageTypeLabelRect = new Rect(offset, position.y + 17, 50f, 17f);

offset += 35;

EditorGUI.LabelField(damageTypeLabelRect, "Type");

Rect damageTypeRect = new Rect(offset, position.y + 17, position.width / 3, 17f);

offset += position.width / 3;

EditorGUI.PropertyField(damageTypeRect, damageType, GUIContent.none);

Rect damageAmountLabelRect = new Rect(offset, position.y + 17, 65f, 17f);

offset += 55;

EditorGUI.LabelField(damageAmountLabelRect, "Amount");

Rect damageAmountRect = new Rect(offset, position.y + 17, position.width / 3, 17f);

EditorGUI.PropertyField(damageAmountRect, damageAmount, GUIContent.none);

break;

case ObjectType.HEALING:

float offsetH = position.x;

Rect healingLabel = new Rect(offsetH, position.y + 17, position.width, 17f);

EditorGUI.LabelField(healingLabel, "this item will heal the player's ");

offsetH += 175f;

Rect healingTypeRect = new Rect(offsetH, position.y + 17, position.width / 3, 17f);

EditorGUI.PropertyField(healingTypeRect, healingType, GUIContent.none);

offsetH += position.width / 3;

Rect healingLabel2 = new Rect(offsetH, position.y + 17, position.width, 17f);

EditorGUI.LabelField(healingLabel2, "by ");

offsetH += 20f;

Rect healingAmountRect = new Rect(offsetH, position.y + 17, position.width / 5, 17f);

EditorGUI.PropertyField(healingAmountRect, healingAmount, GUIContent.none);

offsetH = position.x;

Rect healingLabel3 = new Rect(offsetH, position.y + 34, position.width, 17f);

EditorGUI.LabelField(healingLabel3, "if it is ");

offsetH += 40f;

Rect healingPickupRect = new Rect(offsetH, position.y + 34, position.width / 5, 17f);

EditorGUI.PropertyField(healingPickupRect, healingPickupType, GUIContent.none);

offsetH += position.width / 5;

Rect healingLabel4 = new Rect( offsetH, position.y + 34, position.width, 17f );

EditorGUI.LabelField( healingLabel4, "with." );

break;

case ObjectType.PASSABLE:

Rect passableRect = new Rect(position.x, position.y + 17, position.width, 17f);

EditorGUI.LabelField(passableRect, "Passable objects have no parameters");

break;

case ObjectType.SOLID:

Rect shouldMove = new Rect(position.x, position.y + 17, position.width, 17f);

int index = solidMoving.boolValue ? 0 : 1;

string[] options = new string[] {"Yes", "No"};

index = EditorGUI.Popup(shouldMove, "Should it move?", index, options);

solidMoving.boolValue = (index > 0) ? false : true;

if (solidMoving.boolValue) {

float offsetS = position.x;

Rect startRect = new Rect(offsetS, position.y + 34, position.width / 2, 17f);

offsetS += position.width / 2;

EditorGUI.LabelField(startRect, "Start Point");

startRect = new Rect(offsetS, position.y + 34, position.width / 2, 17f);

offsetS += position.width / 2;

EditorGUI.LabelField(startRect, "End Point");

offsetS = position.x;

startRect = new Rect(offsetS, position.y + 50, position.width / 2, 17f);

offsetS += position.width / 2;

EditorGUI.PropertyField(startRect, solidStart, GUIContent.none);

startRect = new Rect(offsetS, position.y + 50, position.width / 2, 17f);

offsetS += position.width / 2;

EditorGUI.PropertyField(startRect, solidEnd, GUIContent.none);

}

break;

}

EditorGUI.EndProperty();

}

public override float GetPropertyHeight(SerializedProperty property, GUIContent label) {

return base.GetPropertyHeight(property, label) + extraHeight;

}

}