

Balázs Dénes Kovács

Editor Tools Development for   
Creating Avatar Content

The Avatar Content Editor

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

06 July 2016

|  |  |  |
| --- | --- | --- |
| Author(s)  Title  Number of Pages  Date | Balázs Dénes Kovács  Unity Editor Tools Development  xx pages + x appendices  5 May 2016 | |
| Degree | Bachelor of Engineering | |
| Degree Programme | Information Technology | |
| Specialisation option | Software Engineering | |
| Instructor(s) | Juho Iso-Markku, Title (for example: Project Manager)  Antti Laiho, Title (for example: Principal Lecturer) | |
|  | | |
| Keywords | |  |

**Contents**

[1 Introduction 1](#_Toc455551589)

[2 Custom Game Engine Tools 1](#_Toc455551590)

[2.1 History of Custom Tools in Game Development 1](#_Toc455551591)

[2.2 Avatar Customization 2](#_Toc455551592)

[2.3 Unity Editor Scripting 4](#_Toc455551593)

[3 Creating Unity Editor Tools 5](#_Toc455551594)

[3.1 Custom Inspector 5](#_Toc455551595)

[3.2 GUI Skins and GUI Styles 7](#_Toc455551596)

[3.3 Scriptable Objects 7](#_Toc455551597)

[3.4 AssetPostprocessor Scripts 7](#_Toc455551598)

[4 The Avatar Content Editor 7](#_Toc455551599)

[4.1 Items and Products 7](#_Toc455551600)

[4.2 Components 7](#_Toc455551601)

[4.3 Tabs 8](#_Toc455551602)

[4.4 Browsers and Pickers 8](#_Toc455551603)

[4.5 Attributes 8](#_Toc455551604)

[4.6 Functionality 8](#_Toc455551605)

[5 Results 11](#_Toc455551606)

[6 Conclusion 11](#_Toc455551607)

[References 11](#_Toc455551608)

Appendices

Appendix 1. Title of the Appendix

Appendix 2. Title of the Appendix

# Introduction

Today more and more developers use Unity as the game engine for their games. There are over 4.5 million registered developers of Unity as of now [i]. As the number of Unity developers increase so does the amount of complex game projects. These projects require more than only a team of skilled programmers and artists to work together using only the platform Unity provides. However, in order to develop their games efficiently they need to rely on tools made for Unity for special purposes. These tools can be generic to an extent and thus coded by an external personnel or they can also be highly specialized to fit specific needs of the game project.

In this paper I am going to describe the development of The Avatar Content Creator (ACE) that was proposed and developed in Sulake Corporation OY for Project X. The ACE is a highly specialized tool to create content for the avatars of the game. The purpose of this project was to re-implement an already existing tool called the Cloth Editor to facilitate more and better functionalities. Furthermore, the new tool had to support a faster and more advanced art and balancing process. In addition to this, the communication with the data servers of the game had to be overhauled to enable direct editing of the database of items. Finally, the project aimed to increase productivity and improve the stability of the avatar content creation process.

# Custom Game Engine Tools

## History of Custom Tools in Game Development

In the beginning of the history of games development, games used to be made as singular entities. Developers had to make an effort to build their software from the ground up and optimize them for the hardware they were developing for. It was only in the early 1990s that the term game engine surfaced [ii].

In 1993, before the game called DOOM came out id Software introduced the term DOOM engine. This referred to a revolutionary way John Carmack, the lead programmer at id, organized components of their game [ii.] He made a modular separation between the creative assets and functionality in his code. There were distinct core elements of the engine such as the three-dimensional graphics rendering system, the collision detection system, and the audio system. Also the levels, art assets and the rules governing the game were well separated. [iii, 11]

Further on, developers realized the benefit of having their own game engines where they were able to create more games, mainly in the same genre, by swapping elements inside the engine. For instance, replacing weapons and enemies or other art assets. In the late 1990s game engines like id Tech 3 and its game Quake III Arena and the first Unreal Engine were designed with reusability in mind. Furthermore, this meant that anyone could make their modifications to the games made with those engines. Thus the mod communities were born. These groups either made small modifications to the game or such big ones that yielded a completely new game using toolkits provided by the developers of the games. [iii, 11]

Early proprietary game engines and the toolkits developed for them were the first instances of high level tools being used in game development. Being able to edit levels visually in context of the game engine or visualizing rules and behaviours in the game were among the many useful features game engine tools provided in the beginning. Nowadays, there are many more tools created by the original game engine developers and independent developers alike. These tools can be generic, such as a grid creation tool, and some of them are designed to deal with a specific task.

## Avatar Customization

The demand for the ability to customize the in-game representation of players in recent games is high. Many RPG and MMORPG games offer such functionalities. Players may alter the sex, ethnicity and age of their avatars. Furthermore, hairstyles and clothes as well as their colors can usually be changed. These are an important elements of games where players are not supposed to play an existing character with a set personality. This also engages them to become more involved in these games. Players feel more attachment to the characters which they are allowed to customize. This makes them feel that they are part of the game to some extent. [iv]

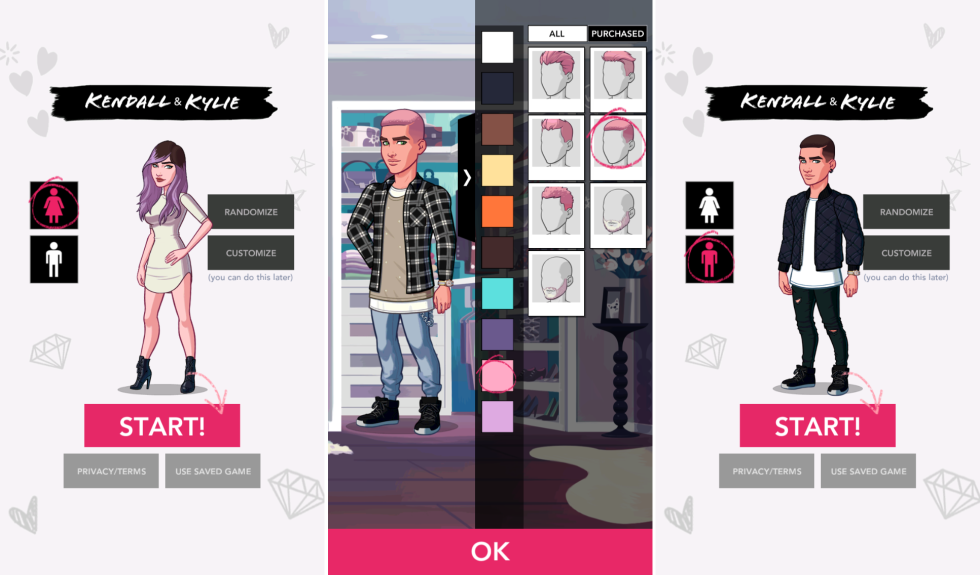
There are several examples of avatar customization between games. One of the best examples is the Mii editors made by Nintendo. It has many features through which players can make lookalikes of themselves, family members, celebrities or even cartoon characters.



1. Wii U Mii Studio [v]

The Mii Studio for the Wii U is one of the editors for Miis made by Nintendo as seen in Figure 1. Miis are a generic character many games on Nintendo consoles use since the release of the Wii in 2006. Players may race with their Miis in Mario Kart Wii or fight other Nintendo characters in the Super Smash Bros. franchise.

On the other hand, there are character customization features in games that do not perform well. For example, the customization screen of the game Kendall & Kylie. The characters do not change much no matter what the player picks to customize it. As seen on Figure 2 the offered colors for customization are not representing the same color on the items.



1. Screenshots of Kendall & Kylie

Additionally, the behaviour of these characters in the game are not representative of the actions or emotions of the players. The characters of this game are only customizable to a limited extent. Styles and colors only represent the narrow subculture the game is catering to and other cultures portrayed from the perspective of this audience. This essentially does not allow for the freedom of expression a customizable avatar should provide.

## Unity Editor Scripting

Editor scripting in Unity is a useful practice for large projects. It allows the developers to build tools and automation which are not implemented by the original developers of Unity. The purpose of this practice is to speed up, simplify and aid game creation processes. Custom tools made by editor scripting can range from custom inspectors to any form of internal utilities such as custom windows, wizards or useful scripts executed from the menus of the editor. [4, 2.]

Unity editor tools might range from generic purpose to highly specialized ones. Using editor scripts one may create custom inspectors. Unmodified inspector views show all the basic properties of the object selected in the hierarchy. Similarly, the inspector reveals all the public fields of the script and draw the default inspector for them. Once the scripted game object gets too many properties and becomes a burden to handle, making a custom inspector might be a good idea. With the help of the editor scripts developers can organize the fields of the objects into logical groups and even implement methods to handle the data from the inspector in a specific way. [4, 49.] Furthermore, the default inspector might be drawn if needed through editor scripts. A simple use case for editor scripts is to display the level of a character object based on a calculation, where the character script itself only stores information about experience points it gathered.

Additionally, the Unity editor features classes such as GUIStyle and GUISkin to allow further modification of the default look of the editor window elements. With the help of the GUIStyle class the font, the color, even the size of the elements can be changed among many other options. [4, 147-150.] Using the GUISkin class the developers may define new skin packages for the default editor window components [4, 156]. Applying a GUISkin rather than using GUIStyles with each element is generally a better approach.

Furthermore, saving data into scriptable objects persistently is also possible with editor scripts. Scriptable objects always exist in the project without being attached to a gameobject. This is why they are widely used for saving changes during play mode. Moreover, saving information into scriptable objects have some benefits over using XML or JSON files for the same purpose. Finally, allowing persistent changes to certain elements of the game gives liberty to game designers when they have to adjust certain values to make the game a better experience for players.

The Unity editor is an extensive set of tools to facilitate rapid game development across multiple platforms. The editor of Unity allows great creative and engineering freedom to developers and designers alike. The many tools of the Unity editor allow developers to visualize their work in the process and manipulate scenes with ease. With the use of editor scripts developers can modify the environment of Unity to benefit the developer team. Moreover, the custom editors are customizable not only with functionality, but with looks as well. Finally, scriptable objects provide a persistent way of storing important values of the game changed during testing and developing.

# Creating Unity Editor Tools

## Custom Inspector

A custom inspector is a view of a specifically modified script in the inspector view of the Unity editor. In any default inspector of a MonoBehaviour each public variable of the script is exposed in a corresponding field.

using UnityEngine;

public class InspectorExample : MonoBehaviour {

public int variableA = 100;

[SerializeField]

private int variableB = 200;

// This variable won't be exposed in the inspector

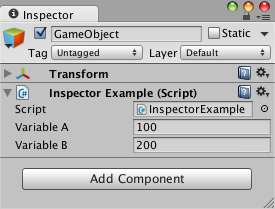
[HideInInspector]

public int variableC = 300;

}

1. Reprinted from [4]

Any value in the default inspector is changeable and are also serializable. Hiding an exposed public field is possible by attaching a HideInInspector attribute to it as seen in Listing 1 and the result in the inspector in Figure 3.



1. Reprinted from [4]

On the other hand, developers might not satisfied with the default inspector and variable hiding or the user friendliness of the resulting interface. They may implement other interfaces utilizing editor scripting.

In order to create a custom inspector for a class the CustomEditor attribute must be used on a class that inherits from the Editor class. The attribute also needs the type of script it is going to be used to create the custom inspector for as shown in Listing 2.

[CustomEditor(typeof(Level))]

public class LevelInspector : Editor {

}

1. Reprinted from [4]

This inspector class will cause the Unity editor to use the OnInspectorGUI function of the class whenever inspecting the script specified in the type of the CustomEditor attribute. A Class structure in the previously specified way has a variable called target. This variable stores the inspected element as a Unity object. Therefore, if developers want to access the script they need to cast this object to the appropriate type. [4]

Furthermore, in cases when the default inspector should be part of the custom inspector, one may add the DrawDefaultInspector function inside the OnInspectorGUI to draw it. The placement of this function is completely arbitrary and will only have an impact on the layout of the inspector. However, drawing custom elements in the inspector can be done by using the functions of the EditorGUI and the EditorGUILayout functions. Additionally, some more elements can be found in the GUILayout class as well. Classes with Layout in their names refer to elements that can be organized into a layout. Whereas the ones without the word have drawing functions that needs to have a Rect specifying their position and size on the display area. [4]

Layouts in the Editor can flow from top to bottom or from left to right. The inspector is drawn with a top to bottom layout. These layouts can be changed by the script and arranged like tables in HTML scripting. The functions from the EditorGUILayout class are used for a vertical layout are BeginVertical() and EndVertical(). Also for the horizontal flow BeginHorizontal() and EndHorizontal() are used.

Finally, organized into layouts are the fields and controls. Layout elements can be drawn in the custom inspector with the help of the GUILayout and the EditorGUILayout classes functions. There are several type specific fields developers can choose from. All of these fields have several overloads to accommodate different needs. For example, some fields need a label and some do not. The overloads of the field functions allow for both type of displays. Also these fields and controls after all the initial properties they take as arguments, they take more arguments in their overloads such as GUIStyle and GUILayoutOption-s. The properties allow to change the general look and feel of the field as well as the fixed and dynamic size of the GUI elements. [4]

## GUI Skins and GUI Styles

To be able to change the look and feel of the custom editor GUI developers need to utilize the classes GUIStyle and GUISkin. Before the 4.6 release of Unity these classes were solely used for game UI customization. As mentioned in the previous section GUI components accept an optional GUIStyle to override their default settings.

EditorGUILayout.LabelField("MyTitle", EditorStyles.boldLabel);

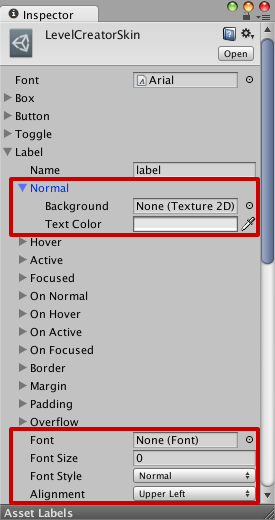
1. The usage of EditorStyles in GUI components Reprinted from [4]

A set of static styles used in the default editor of Unity can be found inside the EditorStyles class. Listing 3 illustrates the usage of such styles on a label field element.

Alternatively, developers may create custom GUIStyles to fit their needs. These styles are initialized with default values such as black font color. Although, the default values do not mirror the styles used in the Unity editor. Because of this, developers need to tweak several attributes to achieve the style they aim for.

Each GUIStyle defines specific states required by the Editor GUI. These states are stored inside the GUIStyle in GUIStyleState classes storing specialized values. The default state for each element is called normal. The several other states store information about the looks of the controls when the mouse is interacting with them in different ways.

Furthermore, GUIStyles can be organized into the GUISkin class which allows to customize the whole UI instead of separate elements. This class must be created as an asset. It is extending the ScriptableObject class.



1. GUISkin custom inspector. Reprinted from [1]

Additionally, the pros of using GUISkins is their reusability in other projects. Finally, these assets generate their own custom inspector to facilitate the graphical editing of GUIStyles as seen in Figure 4. [4]

## Scriptable Objects

Unity has a special object type called scriptable objects. Scripts inheriting from the ScriptableObject class do not need to be attached to an instance of a game object inside the scene in order to exist. This is due to these elements are being saved as assets in the project. Using scriptable object have the benefit of automated data handling and parsing in contrast to using plain text, XML or JSON formats. Furthermore, with the use of scriptable objects developers can create custom editors for game designers where they can test certain aspects of the game with different values and make the changes persist.

## AssetPostprocessor Scripts

# The Avatar Content Editor

## Items and Products

## Components

## Tabs

## Browsers and Pickers

## Attributes

## Functionality

In general

The Avatar Content Editor (ACE) is a tool that facilitates the editing and creation of avatar content such as clothes, set decorations, color, skin colors, clothing sets and gestures With the help of ACE you can modify data of the products and items listed. Furthermore, as Figure 2 illustrates the ACE interface is a custom inspector of a singleton class. This class interacts with a number of other manager classes in order to provide all its functionality.

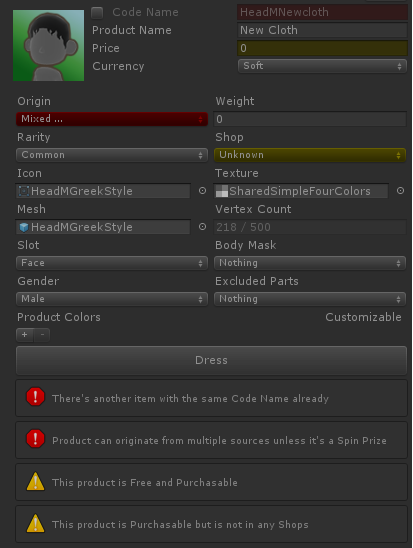


1. The basic look of the Avatar Content Editor

Clothes, colors, skin colors and clothing sets are stored on the servers of H4K, while set decorations and gestures are stored on the hard drive of the users. Because of this duality in storage methods the ACE does not make changes to any of the data until the users sync and save their changes. With the help of ACE users can also take screenshots of dressed up and animated avatars.

Validation

The ACE features a validation functionality which provides helpful feedback on suspicious or erroneous properties of certain items. Suspicious items can be products which are set to be purchasable, although they are also either free or are not assigned to be available in any shop. This would mean that the item is either for free or not available at all in-game. Fields with these type of warnings are colored yellow and show a description bellow the attributes of the items. These descriptions are marked with an exclamation point inside a yellow triangle as seen in Figure 3.



1. Example of Validation in the ACE [1]

Additionally, fields with errors are marked with red. This directs the attention of the user to the data fields where they must make changes. As seen on Figure 3. these fields also have a description bellow the attributes. [1]

Validation is a function called for each and every change on the attribute fields. There are many rules this process is based on. For instance, the code name of any given item must not match another. Also, another example, clothing items must have all of their meshes, textures and icons assigned to them.

Network Communication

# Results

# Conclusion

# References

x

|  |  |
| --- | --- |
| 1 | Tadres A. Extending Unity with Editor Scripting. Birmingham, UK: Packt Publishing Ltd.; 2015. |
| 2 | Navarro A, Pradilla JV, Rio O. Open Source 3D Game Engines for Serious Games Modeling. Alexandru C, editor. Rijeka, Croatia: InTech; 2012. |
| 3 | Henson C, Henson R. Unity 4.x Game Development by Example Beginner's Guide. Birmingham, UK: Packt Publishing Ltd; 2013. |
| 4 | Smith M, Queiroz C. Unity 4.x Cookbook. Birmingham, UK: Packt Publishing Ltd.; 2013. |
| 5. | Lowood H. Game Engines and Game History. Kinephanos. 2014 January. |

x

**Title of the Appendix**

Content of the appendix is placed here.

**Title of the Appendix**

Content of the appendix is placed here.