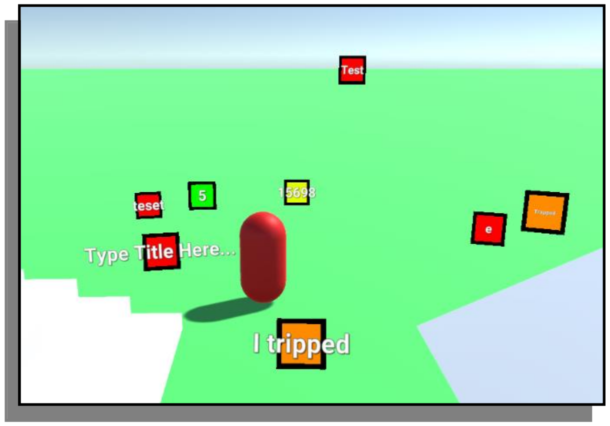
Advanced Technologies Report: How Effective is UI Toolkit at Making a Debugging System

Oscar Wilkinson
Department of Computer Science and Creative Technologies
University of the West of England
Coldharbour Lane
Bristol, UK
Oscar2.Wilkinson@live.uwe.ac.uk
21007294



Abstract

By utilizing the UI Toolkit, can it be a useful enough tool to improve the workflow within the Unity software compared to the in-built UI system?

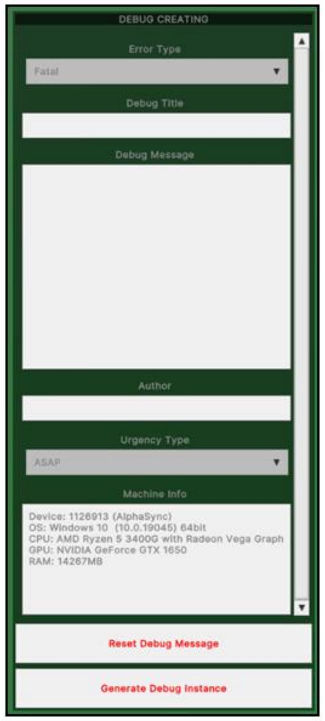
Author Keywords

Unity; UI; UI Toolkit; Debugging; Developer Tools

Introduction

All Unity projects will have access to the standard Unity UI tools without any package or asset importing and can be solely used to generate an entire UI system for a game. The same goes for UI Toolkit, so all developers have a choice to utilize the Toolkit or non-Toolkit UI tools to construct their project. The question lies in which set of resources is better for development workflow.

To help with this question, seeing the development process using UI Toolkit and how it provides its benefits and drawbacks is required. The system chosen to be developed with the UI Toolkit will be a debugging system that allows developers and testers to track and log errors within the game by any standard, ranging from small bugs and large-scale errors to level design choices.



Error Type: Fatal

Urgency Type: ASAP

Machine Info:

```
Device: 1126913 (AlphaSync)
OS: Windows 10  (10.0.19045) 64bit
CPU: AMD Ryzen 5 3400G with Radeon Vega Graph
GPU: NVIDIA GeForce GTX 1650
RAM: 14267MB
```

Figure 1: Creation UI Document

Background Research

UI Toolkit does not have much information regarding its features, resources, and tools compared to the standard Unity UI system, although there are two standout points of information. First being the documentation manual accessible online, informing about functions, classes, structures, and general application of the Toolkit (Unity, 2022). Second being a tutorial Unity project with example uses of systems created by the UI Toolkit package, which demonstrates how to access visual elements, how stylesheets are utilized, usage of functions, and assigning button callbacks (Unity, 2023).

Implementation

Tasks

No.	Description	Necessity
1	Generate debug instances containing information about the debug.	Required
2	Edit already existing debugs.	Required
3	View all debugs in UI.	Required
4	View all debugs physically within the scene.	Should be Implemented
5	Tab selection system, to show/hide different tools.	Should be implemented
6	Customize how the debugs are presented and interact with the system.	Could be implemented

Creating Debugs

Creating debugs requires knowing what should be created and how it should be created.

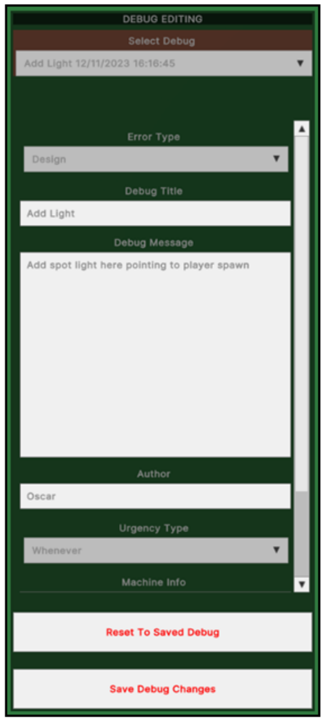
The 'what' is a struct that holds a variety of information the developers will find useful to resolve the bug. This includes:

- Type of debug (Fatal, Warning, Design…).
- Title for the debug.
- Message to describe what the debug is for.
- Date of debug creation.
- World space position of debug.
- Unity scene of debug origin.
- Urgency type (ASAP, Soon, Later…).
- Author of the debug.
- Machine information of source of debug

'How' it is created is through the UI Toolkit. UI Toolkit works by generating UI Documents, adding the component onto any active Game Object within a scene will make it display what is on the document.

These documents consist of visual elements that can range from a text field to a dropdown menu, and themes that can dictate what a collection of visual elements will look like and interact.

The creation of debugs involves inputting all the required information into the UI Document visual elements (shown in Figure 1) and pressing the button to generate a debug, saving all the information into the debug struct mentioned earlier.



DEBUG EDITING

Select Debug: Add Light 12/11/2023 16:16:45

Error Type: Design

Debug Title: Add Light

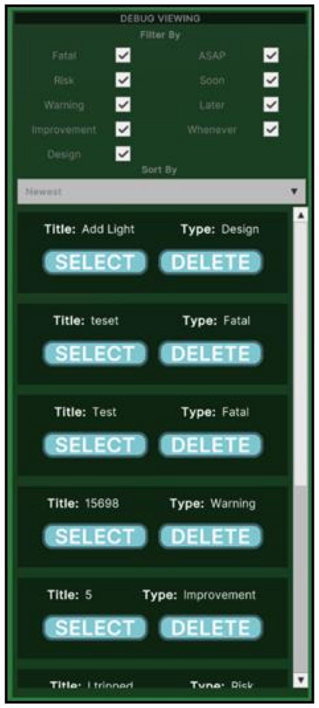
Debug Message: Add spot light here pointing to player spawn

Author: Oscar

Urgency Type: Whenever

Machine Info

Figure 2: Editor UI Document



DEBUG VIEWING

Filter By

Fatal	✓	ASAP	✓
Risk	✓	Soon	✓
Warning	✓	Later	✓
Improvement	✓	Whenever	✓
Design	✓		

Sort By: Newest

Title: Add Light	**Type:** Design
Title: teset	**Type:** Fatal
Title: Test	**Type:** Fatal
Title: 15698	**Type:** Warning
Title: 5	**Type:** Improvement
Title: I tripped	**Type:** Risk

Figure 3: Viewing UI Document

This system is expanded further by allowing to reset all the elements back to nothing in case the current information is not wanted. This system best fits the generation of debug instances as it compacts all the required information into one interface and one output (fulfilling the requirements for task 1).

Editing Debugs

The editing of debugs is implemented similarly to the debug creator in that there is one UI document (displayed in Figure 2) holding the input fields for all necessary elements of the debug information. The difference being with the addition of a dropdown selection box stating which debug to start editing, and an additional button to reset to the last saved information (which completes task 2).

Viewing Debugs

All generated debugs are stored in a list of structs which would be slow to work with if the only method of viewing them is via a dropdown list. To improve the intractability of the list, all debugs are quickly accessible by a UI document that contains a scroll view window with every debug shown with its title, and debug type to differentiate them. This view is also given the functionality of sorting (ordering each debug from top to bottom based on query), filtering (removing visibility based on type or urgency restrictions) to suit the user's needs, selecting them to open them in edit mode, and deleting them to remove unnecessary or completed debug reports (completing task 3).

To further develop the visibility of debugs and directly link them to in-scene scenarios, each debug has a physical game object instance which has multiple features linked with them.

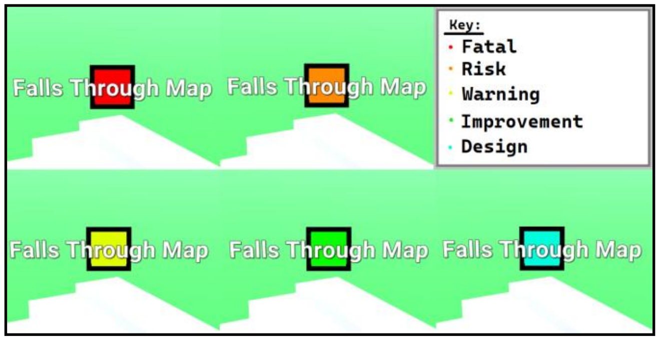


Figure 4: Physical Debug

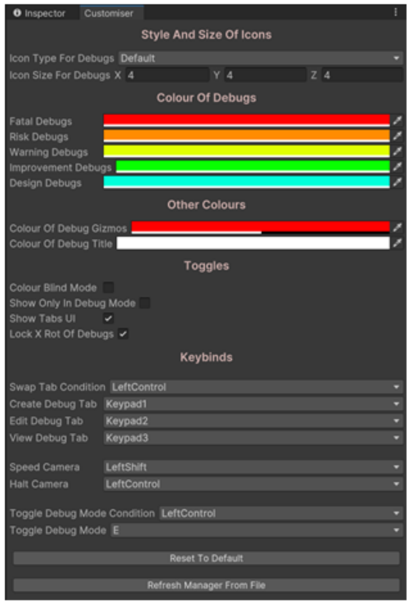
These physical debugs change their appearance based on the type of debug that it is associated with (displayed in the figure above), can be clicked on to select that debug in edit mode, and are only visible at the location and scene of the debug (completing task 4).

Saving Debugs

During run-time all created debugs are stored in a public list of debug instances, however when runtime ends this list will be deleted making the debugging system useless, meaning this information needs to be stored in a long-term place. Saving the list onto a file local to the project solve this issue, this can be done by converting the list into a JSON-formatted .txt file and loading all the information when starting up the scene in run-time and editor mode (further solidifying task 1).

Tabs Selection

With the three main UI document systems made, the screen would be overly cluttered and, in most cases, not need more than one system active. So, a tab system is in place to specifically toggle one system to be highlighted at a time, which disables the visibility of the other systems, helping with the workflow (task 5).



Inspector | Customiser

Style And Size Of Icons

Icon Type For Debugs	Default
Icon Size For Debugs	X 4 Y 4 Z 4

Colour Of Debugs

Fatal Debugs	🟥
Risk Debugs	🟧
Warning Debugs	🟨
Improvement Debugs	🟩
Design Debugs	🟦

Other Colours

Colour Of Debug Gizmos	🟥
Colour Of Debug Title	⬜

Toggles

Colour Blind Mode	☐
Show Only In Debug Mode	☐
Show Tabs UI	✓
Lock X Rot Of Debugs	✓

Keybinds

Swap Tab Condition	LeftControl
Create Debug Tab	Keypad1
Edit Debug Tab	Keypad2
View Debug Tab	Keypad3
Speed Camera	LeftShift
Halt Camera	LeftControl
Toggle Debug Mode Condition	LeftControl
Toggle Debug Mode	E

Figure 5: Custom Editor Window

Customization

Various developers may want systems to operate differently to accommodate their workflow, such as specific colour schemes, accessibility changes, key binds, etc. To allow for these accommodations, some systems have a form of customization in their arsenal. The main changes are as such:

Physical debugs can have their colour schemes altered, show image specific icons for colourblind assistance, alter their size and icon type to make them more visible or less visible, and toggle their visibility when outside of debug mode.

Key binds for activating certain triggers or systems can be altered both in their condition bind (Ctrl, Alt, Shift…) and their main bind (A, Backspace, 7…), to prevent any system clashes of the base game.

Custom Editor Window

Customizing all these values is very useful, but having to go to the specific system and search for one toggle may inadvertently slow down the workflow. It would benefit to have all 'customization' in one single point of access, which introduces the solution of a custom editor window.

Unity scripts are automatically given the inheritance of 'Monobehaviour', but if exchanged out for another in-built class of 'EditorWindow' then the script is instead constructing the layout and interaction of a custom editor window. These windows can also be integrated with UI Documents to structure them, resulting in the customizer window in Figure 5 (completing task 6).

Feedback

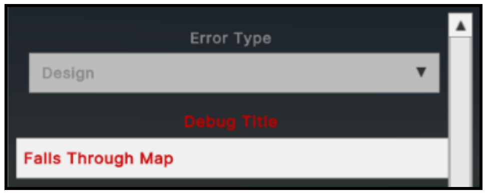
During development the project was overlooked by both supervisors and other peers which allowed for feedback on systems that may need to be added, changed, or removed. This resulted in the creation of the customization of systems, the validation of information, sorting and filtering in view mode, gizmo generation and small fixes/tweaks.

Experimentation

Whilst creating systems there was some experimentation on what might be quicker or more efficient to work with.

The saving system originally converted all the data into a binary formatter, but transferring to JSON had the same level of protection with the bonus of readability and editability.

Originally the customizer window was implemented with standard UI elements without using the UI toolkit, partially due to the absence of knowledge that it could be used in the scenario, but also helped provide a comparison between the two development processes.



Error Type: Design

Debug Title: Falls Through Map

Figure 6: Stylesheet Animation

Stylesheets allow transitions between states which act as animations for visual elements, which (shown in the figure above) can alter the colour, the font size, relative position, scale, and much more in minimal lines.

However, the animation is purely limited to this with no other supporting tools, preventing more controlled animations with specific animation timings and element control. For instance, if you wanted to animate an entire window to come from off the screen then stylesheet animations cannot accomplish this.

Outcomes

All tasks proposed were completed in the implementation process, so the project outcome can be stated as a success, resulting in the following outcomes.

Due to UI Toolkit generating its documents and stylesheets with markup languages it follows on the same principles of HTML and CSS, so any developer that has prior knowledge or experience in HTML, CSS, or web development will find UI Toolkit easy to learn and quick to expand upon.

The stylesheets allow for dynamic visual customization upon multiple elements at the same time, greatly reducing the time of changing small colour values for large quantities of UI elements in standard unity UI.

To reference specific elements, you must directly reference the name of the visual element resulting in very hard-coded solutions, there is no public game object selecting or referencing.

Themes make CSS-like animations very easy to apply and customize, but there are no forms of bespoke animation tools like Unity’s animation component.

Evaluation

UI Toolkit is suited for developers that are used to the workflow of HTML, CSS and other web development practices which is a very different workflow of Unity and C# scripting.

The animation is very limited and, in most cases, cannot reach the standard for UI animation for most games and is better suited for foundation/beta of UI layouts.

Due to these restrictions and workflow patterns, it is better most of the time to use the standard Unity UI resources. The tools can be useful but, in most cases, they are not as beneficial in comparison.

A potential avenue for improvement lies in adopting a more game-object-centric approach and enabling element referencing through them, rather than relying solely on names. Additionally, implementing animation tools would improve UI Toolkit's efficiency, potentially allowing it to surpass the standard UI tools in certain applications.

References

Unity UI Toolkit Documentation - Technologies, U. (2022). Unity - Manual: UI Toolkit. [online] docs.unity3d.com. Available at: https://docs.unity3d.com/Manual/UIElements.html.

Unity Tutorial Project Package - assetstore.unity.com. (2023). UI Toolkit Sample – Dragon Crashers | Tutorial Projects | Unity Asset Store. [online] Available at: https://assetstore.unity.com/packages/essentials/tutorial-projects/ui-toolkit-sample-dragon-crashers-231178.