Game Engine Programming: Creating a versatile inventory system in Unity and Unreal Engine

A screenshot of a video game

Description automatically generated

Martyn Elwell 20021817

Department of Computer Science and Creative Technologies

University of the West of England

Coldharbour Lane

Bristol, UK

[martyn2.elwell@live.uwe.ac.uk](mailto:martyn2.elwell@live.uwe.ac.uk)

# Author Keywords

Inventory; Unity; Unreal Engine; Object-Oriented Programming; Scriptable Object.

# Introduction

The task presented is a research project into creating an inventory system for a third person game, within Unity and Unreal Engine. This report will showcase the dual implementations of inventory systems across the two engines. Explore how they were made in each, what features were used and what programming techniques were needed to create.

In addition will evaluate the benefits and downside to each engine as well as the challenges imposed unique to each. How each engine’s strengths could be used to expand upon the systems created. Discover if one engine is stronger than another for this use case, and which is more applicable to the task.

# Background & Research

For background and research, inspiration was taken from the most successful examples of systems such as those in Minecraft (Mojang, 2011).

A screenshot of a video game

Description automatically generated**Figure 1:** Minecraft Inventory

A screenshot of a video game

Description automatically generated  
**Figure 2:** Minecraft Item Slots

The grid-based system within Minecraft works excellently for clearly showing the items the player has and the quantity of them, using an icon and integer field.

Furthermore, the item Slots included were an interesting addition which could be replicated. In addition, the crafting element offered a unique way of getting new items.

# Implementations

What shared aspects were there between each of your implementations. A section for each engine on how your system was implemented and which aspect of each game engine were required.

## Unity Engine Implementation

In Unity the implementation features a grid-based inventory system with dedicated item slots for tools and equipment. A crafting system to create upgraded items and a skill tree to unlock new abilities. All within separate tabs within the inventory

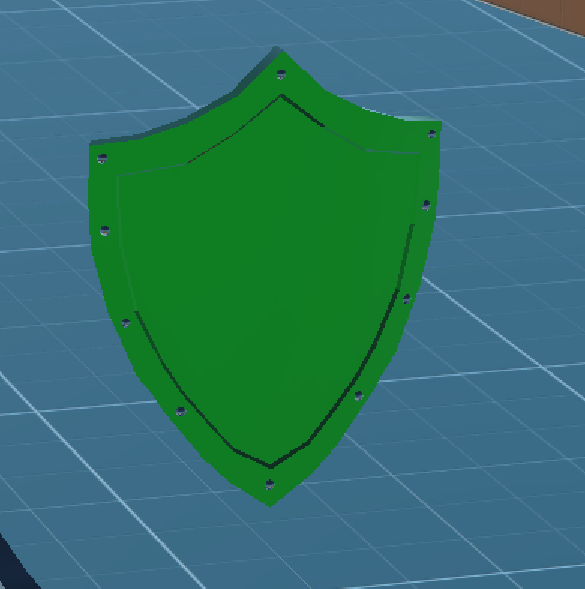
A screenshot of a video game

Description automatically generated**Figure 3:** Main UI screen for inventory

Each Item has a game world game object which stores a scriptable object with all the data for the item. When one of these game objects is picked up by colliding with the player character the information is passed to the inventory system.

A robot with a sword

Description automatically generated  
**Figure 6:** Game model With Items attached



**Figure 4:** Game World Item Pickup

The base inventory is made using a list which is dynamically adds to and removes items. Each item in the list is stored the scriptable object along with a quantity for that item. The visual UI Inventory is a separate UI game object which is derived from a prefab and receives information from the scriptable object, including the item name and icon as well as the Item type.

A screen shot of a computer program

Description automatically generated**Figure 5:** Scriptable Object Class

In addition to the base inventory, there are Item slots specific to tools and armour. The slots are empty by default and allow the player to move items they pick up from the inventory to the slots. However, they will only be moveable if the item is of the type matching the slot. The items consist of one of 5 types, Gems (Standard), Helmet, Armour, Shield, Sword and Useable.

Items will then be applied to the player model and be usable in game. These objects can then be used and referenced by other scripts to take their stats and apply them to other actions the player may do. For example, the Sword slot could pass data on a particular sword, which could be used as a multiplier for an attack function. Useable items have the IUseable interface applied to them meaning that when clicked they each run their own custom use script, this is expandable to include, and useable items designers or future developers may want.

A green shield with white text

Description automatically generated**Figure 7:** Item Types Slots

The Second tab is a crafting window where you can combine 2 items together to create new items. The player can place an item within each slot and if the two items placed correspond with a crafting recipe a new item will appear in the third slot. When they take the third item the two ingredient icons will be destroyed.

The Final tab is a skill tree tab where the player can use XP points collected from items to spend on abilities which once spent will enable the player to use new abilities. This could also be referenced by a theoretical ability manager script within a larger game.

## Unreal Engine Implementation

The Unreal Engine implementation was designed to be as similar to the Unity system to offer an easy comparison. As a result, many of the components are similar. However, due to the differences in the engine the way this is implemented is slightly different.

A screenshot of a video game

Description automatically generated **Figure 8:** Unreal Inventory Implementation

The Unreal implementation consists of game world game objects which can be collected throughout collision to be added to a UI inventory. However rather than storing the information in a scriptable object which is passed from one element to another, the game world object passes data stored at variables as well as a reference to the original actor, to the slot element directly.

In the Unreal implementation all items are usable, by clicking on the UI element with left mouse button, the player can use the item. Which runs their specific use function using an interface, visually demonstrated through a particle system. If the player wishes to drop the item, they can do so through clicking the X element on the slot.

In addition, there is a skill tree included similarly to Unity which links to Booleans within the player blueprint, to enable/disable theoretical abilities.

Creating new Items, is slightly different to unity. Instead of using a pre-made scriptable object, the easiest way is to duplicate the template asset and edit that to make new items to be used.

# Evaluation

Overall, the project was a success, both systems worked within their own engine and reached the initial criteria. The Unity system was the stronger of the two, including more features and feeling smoother. However, the Unreal system would be useable within a game and is an excellent foundation for expansion.

## Unity Engine Evaluation

For Unity the process of creating the inventory system was very smooth due to pre-existing knowledge of the engine in addition to strong documentation and ample reference material online. The process of creating a basic system was very quick and seamless, creating a strong platform to create and expand on the system.

A screenshot of a computer

Description automatically generated**Figure 9:** Pickup system Blueprints

Once an inventory was created, the only significant issue ran into was the creation of the custom slots and moving the items to and from the inventory without inadvertently making ways to duplicate or destroy items unintentionally.

A further idea for expansion would be to create more interactable objects within the world which would have their own inventory and create the interactions between them and the player. In addition to this one way, I would expand upon the useability of the system would be to create an in-editor custom item creator using in-editor tools such as Unity’s UI Toolkit. This would make the creation of new items from a design perspective much easier to add.

The system could be easily applied as part of a larger game, the only addition required would be scripts and work on how and when these items are created within the game, through ideas such as loot drops, puzzle rewards or hidden chests. This is because each element of the system is designed separately to be as modular as possible.

## Unreal Engine Evaluation

The Unreal Inventory presented more challenges than Unity as there were fewer resources to be found and less documentation. Although the use of Blueprints does not require the creator to have as much coding knowledge, which is a benefit to new developers. Blueprints can often be frustrating and hard to understand at first, due to this Unreal became more difficult despite requiring less coding knowledge as the relevant support was harder to find.

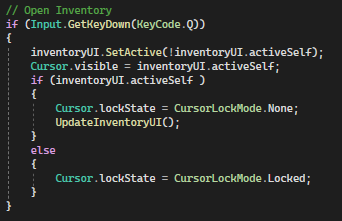
Despite this the Unreal System did work smoothly and effectively once the initial challenges were overcome. Some aspects were easier to implement, such as adding usability to the item was much simpler within Unreal. On of the main obstacles was altering default settings for unreal, for example when in UI and Game input mode holding Right Mouse button will refocus the game and not allow you to click on the UI. Working around this was challenging as the information was hard to find.

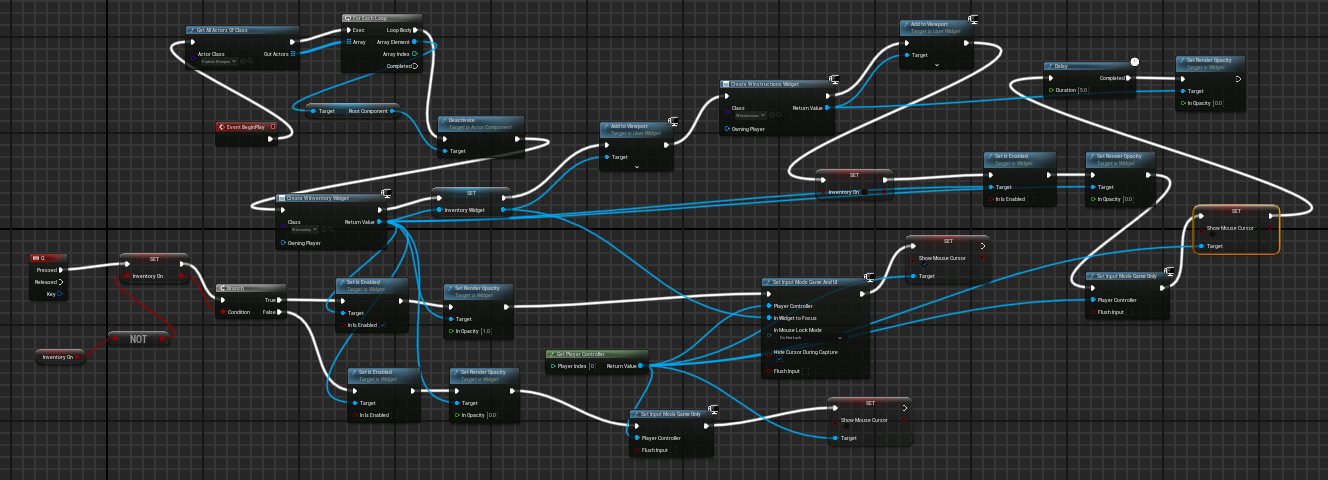
To improve this system the natural additions would be to add type specific slots to the inventory, for items and tools including adding them to the player models. Also, the addition of crafting would add more options of how to use and get items.

Similarly, to Unity the system would be quite easy to apply to a larger game due to its modularity. However, creating many items may be time consuming due to the way each item blueprint is created.

## Comparison

Overall, despite the accessibility of Unreal, I believe that Unity was the better engine for this research project due to the increased documentation and referenceable support online. The Blueprints system is much less overwhelming on first entry, however due to its quickly complicated string of objects and connections it becomes hard to understand.



 **Figure 10 / 11:** Functions for turning the Inventory on and off in both Unity and Unreal Engine for comparison.

However, despite the requirement for coding knowledge, which could be a barrier to entry for some prospective developers, Unity is much easier to implement due to the accessibility of information.

Both Engines are perfectly suited to creating similar systems, the applicable of the engine depends more on the developer’s skillset and prior knowledge than any intrinsic advantages of one engine over another.

**Good Utilization of this Space Sample, as Side Bar**

**Preparation**: Do not change the text box size or position.

**Materials:** This should not appear higher or lower on the page because of pagination and specific headers added during the indexing and pagination process.

**Process:** A 0.75 inch rule is beneficial to break this apart from the body text. The text in this text box should remain the same size as the Body Text: 8.5 Verdana or Arial (with use of **bold** and *italics* to highlight points)

# References

Unity Technologies. (2024). Unity Documentation. [Online] Available at: https://docs.unity.com/ [Accessed 15 January Day, 2024]

Mojang Studios (2011) Minecraft [Video game]. Microsoft Studios. Available from: https://www.minecraft.net/en-us [Accessed 15 January 2024].