# Inventories TDD

## V\_00.01

## Author: Finn Varvell

# Introduction

## Rationale

This TDD is for the creation of an inventory system, which will allow the player to carry items on them. Items will be able to be picked up from the world, taken from chests or bought from shops and the player will be able to view all their items and the details of each one in a toggleable inventory screen. The items will also be able to be used or discarded, and the effects of using will vary depending on the type of item, for example being consumed or equipped. Equipped items will also be shown and can be unequipped back to the inventory.

## Background

Inventory systems are systems included in a large range of different games that let players collect all kinds of items and use them in different ways. Inventory systems can add progression to games the player gains more items and also choices that will change on different playthroughs such as how the player manages their items and money and how it helps them throughout the game. There are many types of inventory systems in different games, and the type of game determines what type of inventory would fit best. The ideal inventory is easy to understand by the player and efficient to manage.

## Terminology

GUI – Graphical user interface

TDD – Technical design document

UML – Unified Modeling language

## Non-Goals

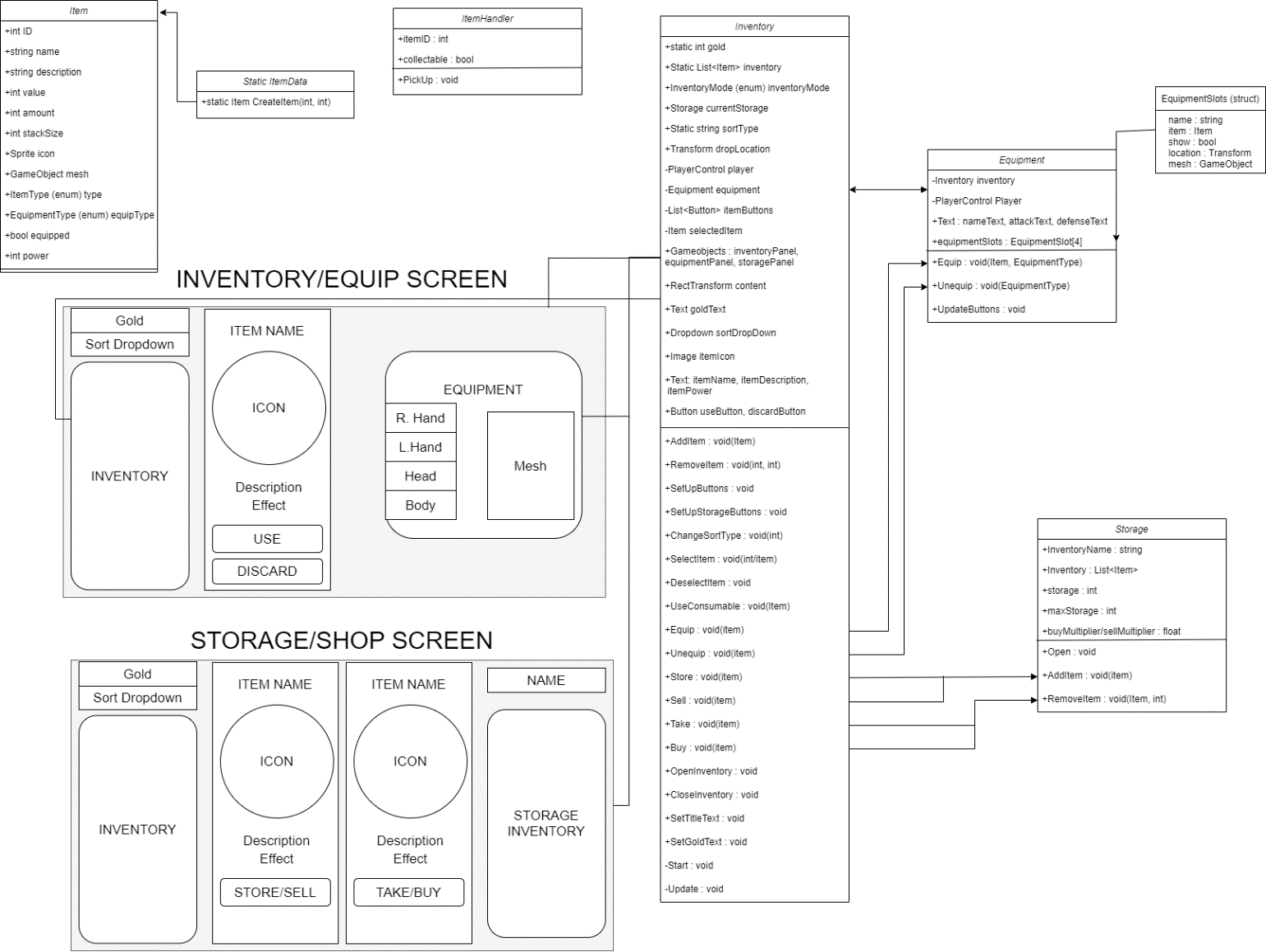
* Different items can stack to different amounts and new stacks are created when one is full
* Inventory saves along with other data, so it is not reset when loading the game

## Proposed Design

* Player has an inventory of items that can be opened and closed
  + Inventory shows lists of items that can be selected
  + Selecting an item shows its information such as icon, name, description and what it does
  + Inventory can be sorted by type
  + When selected, certain items can be used depending on type
  + Consumable items can be used when the player is below maximum health to restore health
  + Certain items can stack, using them decreases the stack by one or removes it if it is out
* Items in world can be picked up
  + Each item has a mesh for when it is spawned in the world
  + By interacting with an item in the world, player can pick it up and it is added to the inventory and removed from the world.
  + Player can discard items from inventory, spawning them into the world
* Equipment tab shown next to inventory
  + Equipment tab will have slots for head, body, right hand and left hand.
  + Certain items can be equipped to these slots such as an axe in hand or a helmet on head.
  + Equipping items removes them from the inventory and can be selected in the equipment menu and unequipped. Equipping an item of the same type as one equipped will swap them.
  + Equipped items will change the attack or defense of the player
  + Some equipped items are shown on the player in a model in the inventory and in the world.
* Storage
  + Interacting with storage opens it up and shows the players inventory next to the storage inventory
  + Items can be taken from the storage to the inventory and stored from the inventory into the storage
  + Storage has a name such as ‘Barrel’ or ‘Crate’ shown above it, and also the amount of items inside it.
  + Storage has a maximum size, and items cannot be stored in it if it is full.
  + Storage has default items in it that are placed when the game is loaded
* Shops
  + Certain NPCs can have shops that items can be bought from
  + Player has an amount of money that is shown in the inventory screen
  + Opening a shop displays the players inventory next to the shop inventory
  + Items can be bought from the shop if the player has enough money
  + The players items can be sold to the shop for gold and are removed from inventory
  + Items are sold for cheaper than they are bought for
  + Shops prices can change based on the npc

# System Architecture

If the design consists of a collaboration between multiple large-scale components, list those components here — or better, include a diagram [UML].



## Data types

### Item

* Ints for ID, value, amount, stack size and power
  + Integers are numerical data types for whole numbers, and any pieces of data on the item that can be stored as a whole number such as the value, power, amount of that item in the stack and maximum stack size will use ints.
  + The ID of the item will also be an int, and each item will have a unique ID so it can be referenced and created with the itemData class
* Strings for name and description
  + Strings are arrays of characters that make up text. The name and description of the item are both just text that needs to be displayed, so they will both be strings
  + Text UI elements on the inventory panels will display the name and description strings when the item is selected
* Sprite for icon and gameObject for mesh
  + Each item needs an icon for when it is selected and a mesh for when it is placed in the world
  + The icon will be a sprite, which is a 2D graphic, and the mesh will be a prefab game object with the itemHandler script attached so it can be picked up.
  + Both the icon and the mesh will be loaded from the resources folder using Resources.Load based on the name and type of item.
* Enums for ItemType and EquipType
  + Enums are custom lists of identifiers in no particular order. Because the type of item will be needed to be accessed by name, an enum works best rather than a string which could be mistyped.
  + ItemType will have values such as Consumable, Weapon, Armour, Material etc. The ItemType of an item will determine what the player can do with it such as consume or equip.
  + EquipType will be used for weapons and armour only and will be out of R.Hand, L.Hand, head and body. This will determine where the item is equipped.
  + Equipment will also use the ‘Equipped’ bool, which can be true or false and lets the game know whether the equip button should be equipping or unequipping

## Data Model

* Each item only in one place at a time
  + Shops and storage use same script, just a variable that changes whether money is used. Moving items between shops and storage removes item from one side and adds to the other.
  + Dropping items removes from inventory and places a gameObject in the world with the items ID attatched
  + Picking up dropped items creates a new item with the ID and adds it to the inventory.
* The itemData script contains data for every item organized by ID, with the first 100 being consumables, followed by weapons, armour and then other types.
  + CreateItem script is passed an ID and generates a new Item class based on the ID.
  + If an ID without an item is passed, it creates an item of ID 0 instead.

## Interface/API Definitions

Describe how the various components talk to each other. URL and the format of the data and parameters used.

Libraries used:

UnityEngine.UI – Allows references to the Unity UI such as the text, buttons and dropdowns/sliders/toggles. All the functions of them can then be used, and the values can be changed. The inventory is made using canvas and so every part of it is some kind of UI element that needs to be interacted with.

## Impact

* Performance – doing things to reduce the performance cost of running the game
  + Running events that only need to be run once a single time rather than every frame, for example updating buttons and text only when things are changed in the inventory rather than every frame
  + Coroutines to wait for responses over multiple frames
* Security
  + Public variables vs private variables – Public can be accessed outside of the class and changed from the editor and private can only be accessed from within the class.
  + Private variables with public properties – The items will all have private variables, which store the data of the item, and public properties, which are how other scripts access the variables. Properties can be made with special get and set functions for when they are being accessed, which gives more control over how the variables are affected from the outside.

## Risks

If there are any risks or unknowns, list them here. Also, if there is additional research to be done, mention that as well.

* The only models for the mesh being used are the helmet and axe, so primitive shapes will have to be used for the weapons and the other armour slots will have to not show

## Alternatives

If there are other potential solutions which were considered and rejected, list them here, as well as the reason why they were not chosen.

* Trying to make simple models for the weapons so I can tell what is equipped would be a solution, but I don’t have a program for 3D modelling and it would probably take a long time trying to figure out how to get it working in unity
* Using primitive shapes to show when armour is equipped wouldn’t look right, so just not showing anything on the model is a better solution

# System Testing

## Testing

Show progress, Error reports and explain fixes you used.

**Item stacking –** There were a lot of problems with trying to make different items stack. Trying to include stacking from every way to get each item was too complicated, so I made every time an item needed to be added to the inventory call the AddItem function, which handled the stacking. Because different items had different stack sizes, and stacks of items could be in any spot of the inventory, the function had to go through every item in the inventory until it found a stack that wasn’t full and add it there. The problem was that when stacks of items were taken from storage, they needed to stack on to items in the inventory but not go over the stack limit. This was eventually solved by adding a remainingAmount variable and running through the inventory adding items to the stack and decreasing the variable until it was zero or the stack was full. If the stack was full, it then created a new stack and continued from there.

**Discarding equipped items –** Items had a bool for whether they were equipped and could be selected in the inventory and equipped or unequipped. However, when discarding them, they also needed to be unequipped, so it went through the items until it found an item with the same ID and unequipped it. The problem with this was that discarding an axe while a different axe was equipped would also unequip the other axe, which shouldn’t happen. I couldn’t find a good solution to this so I just removed the option to discard equipped items and instead they had to be unequipped first and then it worked fine.

**Returning to menu and inventory –** Exiting the game reset the inventory, but just returning to menu did nothing to affect the inventory but the chests were all reset, meaning the player can take some items, return to the menu, load the game and take those items again to now have multiples of each. Fixing this would mean the players inventory and the inventory of every storage would have to be saved, and I wasn’t really sure how to do that so it was never fixed.