Modular Complex System Brief – Inventory Manager System

Documentation for the system by Joseph Huskey

# System Overview

I am designing an inventory system that will allow a designer or programmer to make an inventory for a roguelike or rpg game with the purpose of sorting, removing, adding, and stacking items in an adjustable list.

# Packaging

* Unity
  + A Unity package (.unitypackage)

# Quick Start – Using the Modular Complex System

## Integrating the System

Import the unity package into your Unity 2020.1.10f1 and you should be able to right click in your project view and able to access the items/inventory scriptable objects.  
  
You can make use of these objects simply by attaching them to the player character or whatever other object you need an inventory on (chest etc), design your specified objects by setting their parameters and either using the provided display method or designing your own.

## Building Upon the System

To build onto this system you may want to design more item types with their own abilities this can be done by inheriting from the ItemType class which provides some basic variables. Note when creating a new item type to add to the ItemType Enum as well as to use the asset menu code to be able to generate the object.   
  
(Example of asset menu)  
CreateAssetMenu(filename = “New [type] Object”, menuName = “Inventory System/Items/[type])

# Underlying Mathematical Operations and Algorithms

## Mathematical Operations

This inventory system does not currently make use of any mathematical operations removing and adding items is entirely based on the index/ID of the items.

## Advanced Algorithms

Currently no advanced algorithms are in the base version.

# Additional Third-Party Libraries

Please identify and provide a link and license for each third-party library used to implement this modular complex system.

When discussing each third-party library, discuss why it is included rather being completely implemented by yourself.

# Research Material

Optionally, please include links or references to other research material that helped you develop this modular complex system.

Appendix I – Technical Design

This **optional appendix** is included to encourage you to start thinking about considerations that you will have make as a part of the technical design process. It is outside of the requirements as outlined by the Complex Game Systems subject but will be reviewed if included.

This is a preview of what is to come when we revisit the Technical Design Document (TDD) in Minor and Major Production.

# Technical Requirements

The technical requirements are the needs that must be met for your project to perform as required (i.e. meet the needs of the client or user of your modular complex system).

* Target Frame Rate
  + You may choose to vary this in different situations (i.e. 60fps gameplay, 30fps cutscenes)
* Target Render Resolution
  + You may specify more than one, if applicable
  + This should serve as the baseline for how you author your user interface
  + Omit this if rendering is uninvolved
* Memory Usage Budget
  + Please specify if any memory will need to be dynamically allocated (and deleted/GC’d)
* Target Device and/or Hardware
  + If specifying a PC, please provide an approximation of the capabilities of the PC.

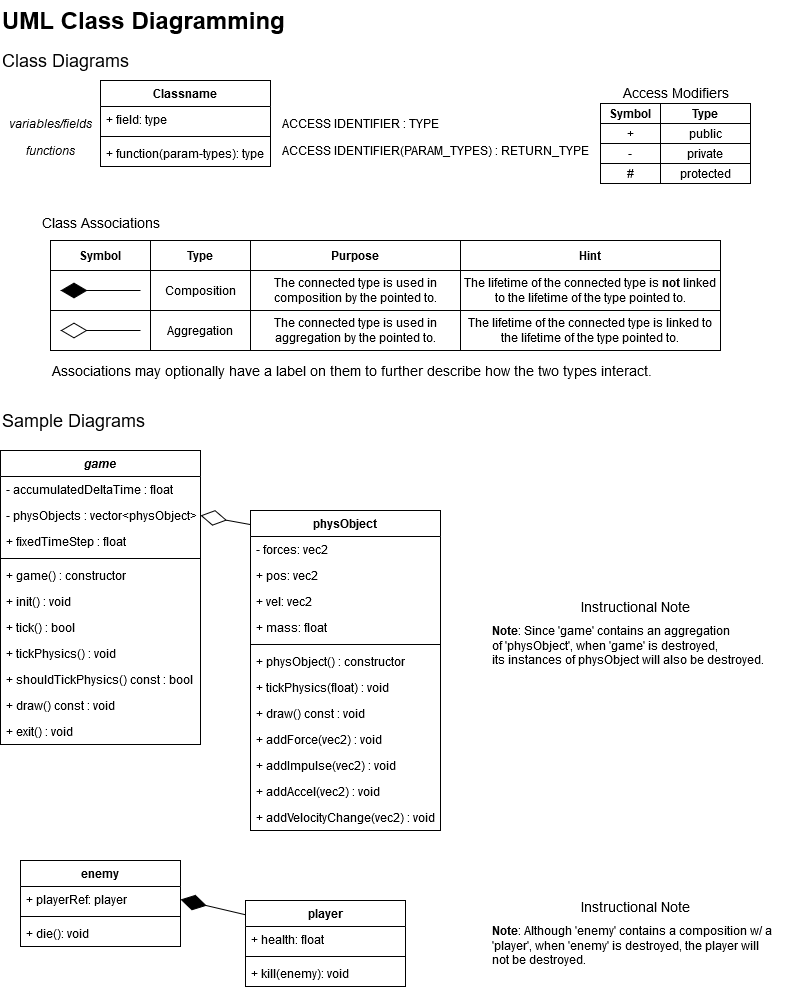
# Technical Design

The technical design of a project is a provides a high-level understanding of how your project will be engineered to meet the requirements laid out above.

* Major Systems and/or Data Types
  + (Please provide a highly level description of how the systems or data types interact with each other)
  + (This can be partially addressed by a class diagram)
* Online Services (if any)
* Global and/or Persistent State
  + Singletons
  + Global Variables
  + Static Variables
  + Assets / Files
* Unity (if applicable)
  + Scripting Runtime
  + Render Pipeline
    - Built-in Render Pipeline
    - Universal Render Pipeline (previously known as the Lightweight Render Pipeline)
    - High-definition Render Pipeline
    - Custom Render Pipeline
  + Tags and Layers
* Rendering Settings (if applicable)
  + Color Space (Gamma / Linear)
  + Render Path (Forward / Deferred / Hybrid)
  + Lightmap
    - Environmental Lighting Settings
    - Use of Dynamic Lighting?
    - Use of Static Lighting?
      * Lightmap resolution
      * Will lightmap seam stitching be needed?
* External Tools (if any)
  + Unit Tests?
  + Profiling/Debugging Tools?
  + Automated Build Systems?

## Class Diagram

**Optionally**, you may choose to include a class diagram to show how you plan to create this system.



The above diagram was created using [draw.io](file:///C:\Users\s209062\Downloads\draw.io) in “Class Diagram” mode. Elements for things like the “composition” and “aggregation” associations can be found in the element palette in the bottom-left corner of the interface.

# Conventions

**Optionally**, list the conventions that you will adhere to over the course of the development of your project.

* Programming Style (see C# Style Guide or C++ Style Guide on SharePoint for a basic style guide)
* File Naming Conventions
* Folder Structure
* File Format Specifications

# Technical Risks

No risks obviously it’s perfect

# Credits

Joseph Huskey © 2022