InvenStory  
Story Asset Generator

Sean Anderson

Regis University

Revision History

**Date Revision Author Description Document Tracking**

1/17/2017 **1.0 Sean Anderson Genesis**

1/29/2017 **1.1 Sean Anderson Corrected Use Case Terminology**

**Updated with Domain Diagram**

2/4/2017 **1.2 Sean Anderson Updated Domain Diagram**

**Added Service Diagram**

2/12/2017 **1.3 Sean Anderson Updated Service Diagram**

2/19/2017 **1.4 Sean Anderson Updated Domain & Service Diagrams**

**Added Business Diagram**

Table of Contents

[1 Introduction 5](#_Toc474658042)

[1.1 Intent 5](#_Toc474658043)

[1.2 High Level Requirements 5](#_Toc474658044)

[1.2.1 World Inventory Generation 5](#_Toc474658045)

[1.2.2 Inventory Object Generation 5](#_Toc474658046)

[1.2.3 Mote Definition 5](#_Toc474658047)

[1.2.4 Event Definition 5](#_Toc474658048)

[1.2.5 Effect Definition 6](#_Toc474658049)

[1.2.5 Save/Load/Export of World Inventories 6](#_Toc474658050)

[1.2.6 Inventory Inheritance 6](#_Toc474658051)

[2 InvenStory Use Cases 7](#_Toc474658052)

[2.1 Intent 7](#_Toc474658053)

[2.2 User Use Cases 7](#_Toc474658054)

[2.3 Use Case Diagram 8](#_Toc474658055)

[3 Fully Dressed Use Cases 9](#_Toc474658056)

[3.1 Intent 9](#_Toc474658057)

[3.2 Create New World Inventory 9](#_Toc474658058)

[3.3 Save a New or Existing World Inventory 9](#_Toc474658059)

[4 Class Diagrams 12](#_Toc474658060)

[4.1 Domain 12](#_Toc474658061)

[4.2 Service 13](#_Toc474658062)

[13](#_Toc474658063)

[5 Summary 14](#_Toc474658064)

# Introduction

## 1.1 Intent

This document details the requirement and development progress of the InvenStory Story Object Generator. This will include a list of use cases, strategically selected fully dressed use cases, and the design details of the Invenstory Story Object Generator

## 1.2 High Level Requirements

InvenStory is a game and genre agnostic content generation tool for game development. It allows a game developer to use a single, streamlined inventory style system for all major characteristic tracking for game entities (characters, objects, events, locations, etc.). The Story Object Generator is the tool that allows the creation of these inventories for use in a game built on the InvenStory system.

The basic object and event systems will be developed in the generator first, then (outside the realm of this course) it will be expanded to a fully functional system and implemented into a full game. The system allows a designer to create specific game assets by defining them as an inventory and adding defining objects to that inventory (i.e. a ray gun may contain a “weapon” tag, a “laser” tag, and 2 smaller inventories, one containing “5” and “MinimumDamage”, and the other containing “”15” and “MaximumDamage.” Resulting in a ray gun that fires a laser that does 5-15 damage.) The actual effect of the tags is dependent on game-specific code, InvenStory merely generates and tracks the tag data.

No gameplay, graphics or IP content will be produced as part of the generator. This is a development tool, not a game.

### 1.2.1 World Inventory Generation

Users will be able to create a new top tier inventory (a “World Inventory”) and set the global parameters for the world. There can be only one active World Inventory at a time and no additional World Inventories can be created within an existing World Inventory.

Optionally, a generic, undefined World Inventory can be used if a more definite World Inventory is not necessary.

### 1.2.2 Inventory Object Generation

The use will be able to define lower tier inventories within the scope of an existing World Inventory. These objects may be created to contain other objects (motes, events or other inventories) in their default state.

### 1.2.3 Mote Definition

The user will be able to define static motes as a single phrase or keyword. Motes are the most basic objects in the InvenStory system can be places within inventories but have to inventory of their own.

### 1.2.4 Event Definition

The user will be able to define events which is a check for a specific condition within an inventory. If the condition is true the event becomes active, if the condition is not true the condition is inactive. The effects of an event are stored as objects within the event’s inventory. The effects of a triggered event are not handled within the asset generator but the user can verify whether or not an event’s parameters are met within the default inventory configuration.

### 1.2.5 Effect Definition

The user will be able to define the particular effects an event has or, if they are present in a non-event inventory, cause a static effect on the inventory itself. The effects of a triggered events are not handles within the basic asset generator.

### 1.2.5 Save/Load/Export of World Inventories

The will be able to save a World Inventory to a .txt file or open a previously created world inventory from an existing save file. The game arm of the code will be capable of extracting a world inventory from the same .txt file format.

### 1.2.6 Inventory Inheritance

Inventories will automatically detect and inherit the properties of items contained within it, allowing complex inventory interactions and object locations. These will be represented by a single inheritance state for each inventory that contains objects. The inheritance state will differentiate between an inherited trait/mote and a local trait/mote.

# 2 InvenStory Use Cases

## 2.1 Intent

This section outlines the use cases that relate to the users interaction with the InvenStory Story Object Generator. Game implementation and player experiences are not within the prevue of this document.

## 2.2 User Use Cases

These are the InvenStory use cases ranked by anticipated priority:

Create New World Inventory

Save A New or Existing World Inventory

Create Custom Inventory Objects

Load A Previously Saved World Inventory

Create Custom Events

Create Custom Effects

Create Custom Motes

Check Inventory Inheritance

Check Event Status

Export Non-World Inventory Objects

## 2.3 Use Case Diagram

Create New World Inventory

Save A New/Existing World Inventory

Create Custom Inventory Objects

Load A Previously Saved World Inventory

Create Custom Events

Create Custom Effects

Create Custom Motes

Check Inventory Inheritance

Check Event Status

Export Non-World Inventory Objects

# 3 Fully Dressed Use Cases

### 3.1 Intent

### 3.2 Create New World Inventory

Primary Actor: User

Assumptions: *None*

Stakeholders: User  
Pre-Conditions: The system has no active World Inventory in which to work.

Post-Conditions (Success Guarantee): There is a new World Inventory which the user can customize.

Flow of Events:

1. The user opens the software.
2. The system prompts the user to either create a new World Inventory or load a previously saved world inventory (The second option will not be available until the save/load system is implemented).
3. The user selects the option to create a new World Inventory.
4. The system prompts the user for a name for the new World Inventory with a default value of “MyWorld”.
5. The system verifies the name is valid (excluding non-alphanumeric symbols)
   1. ALTERNATE FLOW The name is invalid.
   2. The system prompts the user to select a different name without invalid characters and returns to #4.
6. The system prompts the user to select the number of initial inventory slots the new World Inventory will have.
7. The user selects a number of inventories (values available 1-*int*).
8. The system verifies the value is in the acceptable range.
   1. ALTERNATE FLOW The user input is above the max value for *int.*
   2. The system displays a message that the value will be set to “2,147,483,647”.
9. The system repeats steps 4-8 for each of the new inventories.
10. The system finalizes the World Inventory creation process and informs the user.

(Ideally, additional custom development would follow but that is beyond the scope of this use case)

### 3.3 Save a New or Existing World Inventory

Primary Actor: User

Assumptions: The user has already created a World Inventory which is currently active in the system

Stakeholders: User  
Pre-Conditions: There is a World Inventory with any number of inventory objects, events or effects associated with it.

Post-Conditions (Success Guarantee): The World Inventory has been recorded into a properly documented file and is ready for transfer or to be reloaded into the generator at a future date.

Flow of Events:

1. The user prompts the system to save the currently active World Inventory
2. The system checks if the current World Inventory Name is available in the default save location.
   1. If the name is not available the system prompts the user to choose to either overwrite the old file or rename the currently active World Inventory.
      1. The system confirms overwrite OR
      2. The system prompts for and validates new name.
3. The system creates a new txt file with the chosen name.
4. The system creates a string with current software build information.
5. The system writes the string to the new txt file.
6. The system converts the inventory objects to strings starting with the World inventory and working down by tiers.
7. The strings are written to the txt file.
8. The system confirms that the World Inventory has been saved.

# 4 Class Diagrams

## 4.1 Domain

INVENTORY  
-------------------------------------------------

name: String  
inventoryMax: int  
inventoryStored: int  
tier: byte  
objectInventory: Storable array

-------------------------------------------------

getName()  
setName(String name);  
getTier();  
setTier(byte tier);  
getInventoryMax();  
setInventoryMax(int inventoryMax); getInventoryStored();  
setInventoryStored(int inventoryStored); getObjectInventory();  
setObjectInventory(Storable[] objectInventory);   
equals(Object obj);   
hashCode();

WorldInventory  
-------------------------------------------------

setTier(int) <<override>>

BasicInventory  
-------------------------------------------------

DataSphere  
-------------------------------------------------  
worldInventory: WorldInventory  
inventory1 :Inventory  
inventory2 :Inventory

## 4.2 Service

WorldInventoryServicesImpl  
-------------------------------------------------

nameIsUnique(WorldInventory worldInventory, String name);

saveWorldInventory(WorldInventory worldInventory);

validateWorldInventory(WorldInventory)

InventoryServicesImpl  
-------------------------------------------------

addToInventory(Inventory inventory, Inventory item);

validateInventoryName(String Name);

validateInventorySize(String size);

checkInventoryAdd(Inventory inventory, Inventory item);

<<Interface>>  
WorldInventoryServicesImpl  
-------------------------------------------------

nameIsUnique(WorldInventory worldInventory, String name);

<<Interface>>  
InventoryServicesImpl  
-------------------------------------------------

addToNext(Inventory inventory, Inventory item);

validateInventoryName(String Name);

ServiceFactory  
-------------------------------------------------

getInventoryServices();

getWorldInventoryServices();

USE CASE 1

Business Layer

<<Interface>>

IService

PropertiesCustodian  
-------------------------------------------------

loadProperties(String fileLocation)  
getProperties(String key)

## 4.3 Business (with collapsed Service Layer for space)

MANAGER  
-------------------------------------------------

factory:Factory  
  
performAction(String command, DataSphere data)  
getService(String)  
loadProperties(String,Datasphere)

InventoryManager  
-------------------------------------------------

newWorldInventory(DataSphere)  
saveWorldInventory(DataSphere)  
addToInventory(DataSphere)

SERVICE LAYER

# 5 Summary

This document contains the ongoing details for requirements, use cases, fully dresses use cases and design details of Fleet of the InvenStory story object generator.