Inventory System

Architecture/Design Document

**Table of Contents**

[1 Introduction 3](#_Toc131876418)

[2 Design Goals 4](#_Toc131876419)

[3 System Behavior 4](#_Toc131876420)

[4 Logical View 4](#_Toc131876421)

[4.1 High-Level Design (Architecture) 4](#_Toc131876422)

[4.2 Mid-Level Design 5](#_Toc131876424)

[4.3 Detailed Class Design 6](#_Toc131876425)

[5 Process View 7](#_Toc131876426)

[6 Physical View 8](#_Toc131876428)

[7 Use Case View 9](#_Toc131876429)

Change History

**Version:** 0.1

**Modifier:** Seth

**Date:** 03/13/2021

**Description of Change:** Details created for prototype.

**Version:** 0.2

**Modifier:** Sahil

**Date:** 04/17/2021

**Description of Change:** HLD ,Mid-level,Module diagrams and description changed.

# Introduction

This document describes the architecture and design for the Siege application being developed for HLD#2 group project. Siege is PVP multiplayer tower defense and offence game, one player puts down towers defending the castle while the opponent spawns troops and tries to destroy the castle while moving through towers.

The purpose of this document is to describe the architecture and design of the Siege application in a way that addresses the interests and concerns of all major stakeholders. For this application,

the major stakeholders are:

* Developers – they want an architecture that will minimize complexity and development effort.
* Project Manager – the project manager is responsible for assigning tasks and coordinating development work. He or she wants an architecture that divides the system into components of roughly equal size and complexity that can be developed simultaneously with minimal dependencies. For this to happen, the modules need well-defined interfaces. Also, because most individuals specialize in a particular skill or technology, modules should be designed around specific expertise. For example, all UI logic might be encapsulated in one module. Another might have all game logic.
* Maintenance Programmers – they want assurance that the system will be easy to evolve and maintain on into the future.

# Design Goals

The design priorities for the system are:

* The design should minimize complexity and development effort.
* The design should make the code readable and ready to use for new programmer without needing much looking into the class.
* The design should make it easy to reuse the class for future projects with little changes required.

# System Behavior

The inventory class is responsible for keeping track of all items owned by its owning player. It is responsible for activating selected items, making it possible to spawn specific troops or towers during gameplay. Items are placed in the inventory by buying them in the shop system. Once an item is spawned, it is taken out of the owning player’s inventory.

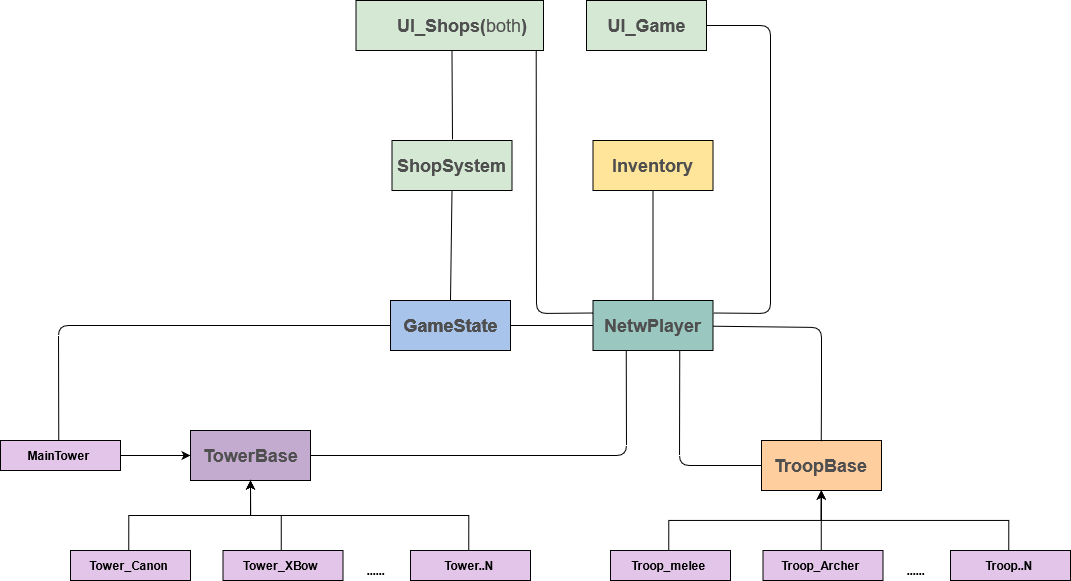
# Logical View

The logical view describes the main functional components of the system. This includes modules, the static relationships between modules, and their dynamic patterns of interaction.

In this section the modules of the system are first expressed in terms of high level components (architecture) and progressively refined into more detailed components and eventually classes with specific attributes and operations.

## High-Level Design (Architecture of the Entire system)

The high-level view or architecture consists of 6 major components:



**NetwPlayer:** This handles events happening during a match, such as spawning troops or towers on spawn points on user input. It stores player data like

Inventory,roundsWon.It also has functions for UI to interface to inventory.

**Inventory**: It keeps tracks of player’s bought items and player’s money. It also provides functions to other classes such as shopsystem, player to add or remove a troop pr tower from inventory.

**ShopSystem**: This handles the shop interface for players. It has blueprint functions that are triggered on the user interacting with the shop UI to purchase items.

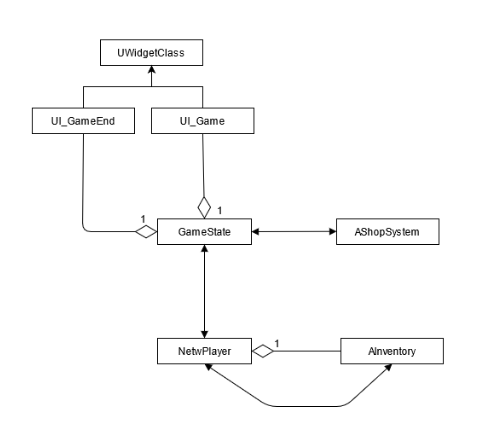
**GameState:** GameState is responsible to switch between UI widget based on current state of the game. It tells ShopSystem to display which widget based on the player's role(attacking/defending).It transitions between states like play, intermission(buying period),GameOver.It performs UI actions and clear units from world when round ended.

**TowerBase:** This is responsible for detecting troops in range and calling attack() on the child class. All Tower are similar when it comes to troop detection so that code is defined in the base class while each different tower has a different type of attack, hence that code is present in the child class called by the TowerBase class on detecting troops in near range.

**MainTower:**It is the main tower in the map that when destroyed notifies GameState and declare victory of the player attacking.

**TroopBase:** This class detects towers in its path and notify the child class to perform attack or any other response. Similar to TowerBase, TroopBase does the tower detection while the actual attack code which is unique to the troop is in the derived troop class.

## Mid-Level Design of Module Inventory



The **Inventory** object is spawned by NetwPlayer, it contains functions that interface UI to inventory.

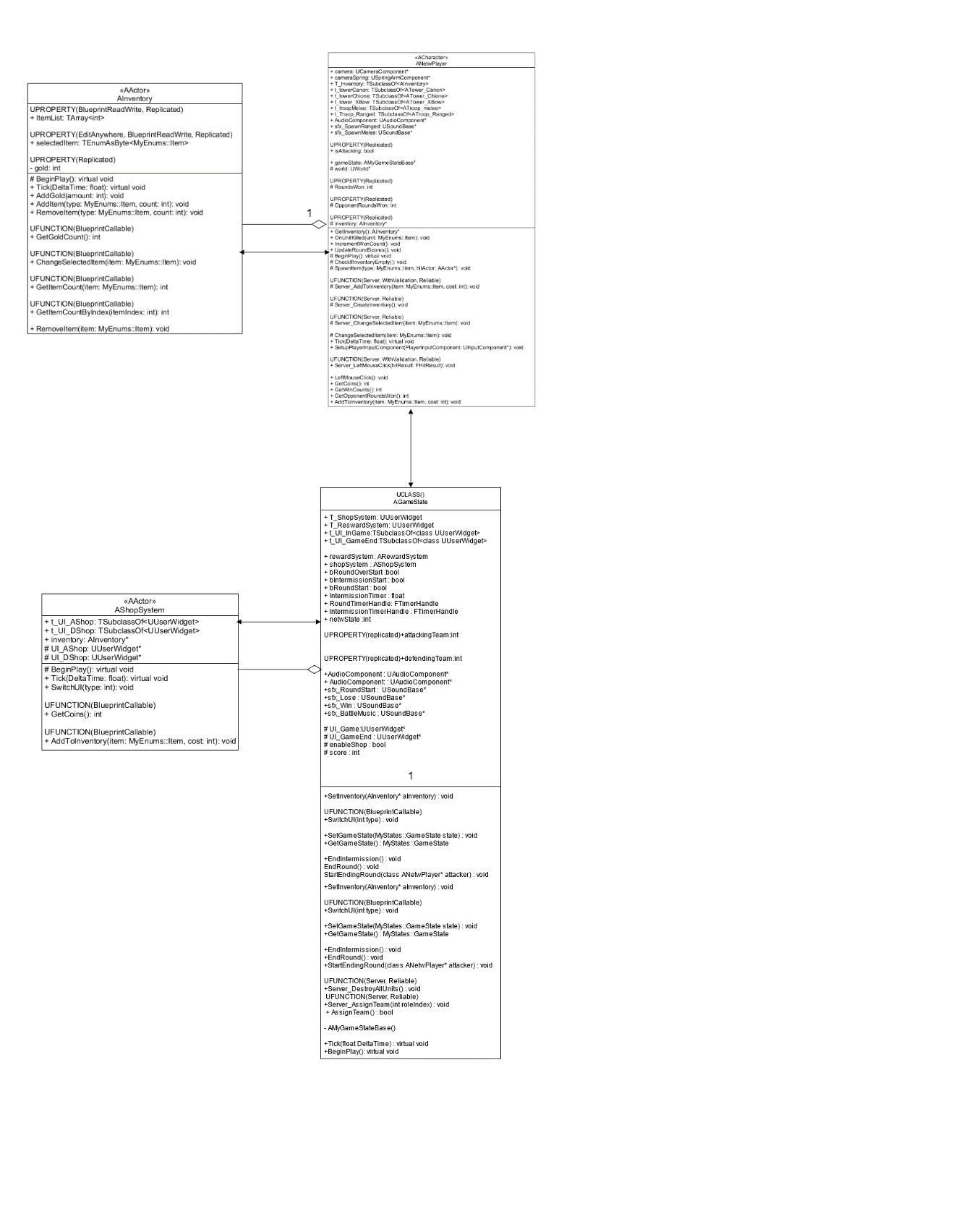
NetwPlayer performs checks when spawning by asking inventory if the requested unit is there or not in the inventory.

**GameState** spawns UI\_Game and UI\_GameEnd, and contains object of ShopSystem, GameState switches between them depending on the game’s state.

**UI\_Game** shows the units player have, spawns them in runtime.

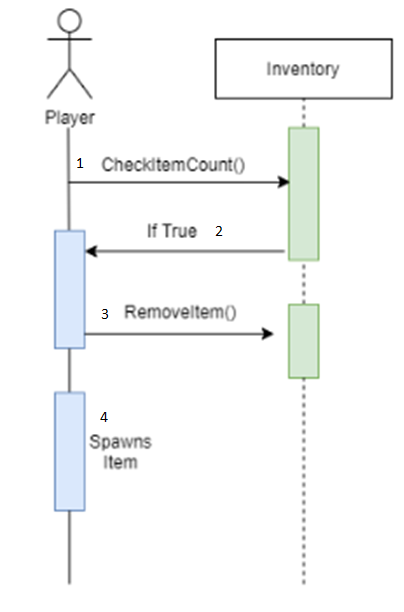
**UI\_GameEnd** is enabled at the end of the round to show if player won or lost the round.

## Detailed Class Design of Module Inventory



# Process View of Module Inventory

Process of spawning troop after the player clicks the left mouse button.



# Physical View (Applies to Multiplayer)

Inventory class is exist on Server and the owning player’s system.It doesn’t contain RPC as its object is spawned by the PlayerCharacter(NetwPlayer) and the operations performed through UI ,goes through NetwPlayer’s RPCs.

In Inventory variables like gold,ItemList and selectedItems are being replicated.

# Use Case View

When the player selects an item from the shop screen, the inventory system assigns the item a type based on the tags in that item’s class. This allows the inventory to keep track of how many items a player owns of each type. The inventory also tracks the amount of gold a player has, which limits the amount a player can purchase at any given time.