Tower AI

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:** Sahil Shaikh

**Date:** 02/14/2020

**Description of Change:** Logical diagrams added.

**Version:** 0.2

**Modifier:** Sahil Shaik

**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(/attack) game, 1 player puts down tower defending the castle while the opponent spawn troop and try 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 inventory 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 inventory class.
* The design should make it easy to reuse the class for future projects with little changes required.

# System Behavior

TowerAI system defines the behavior of the towers in different situation. It is responsible for detecting troops in its range. Then the derived classes have the code to shoot which is unique in each tower, which is called by the troop detecting code ,which also sets the target troop to shoot at.

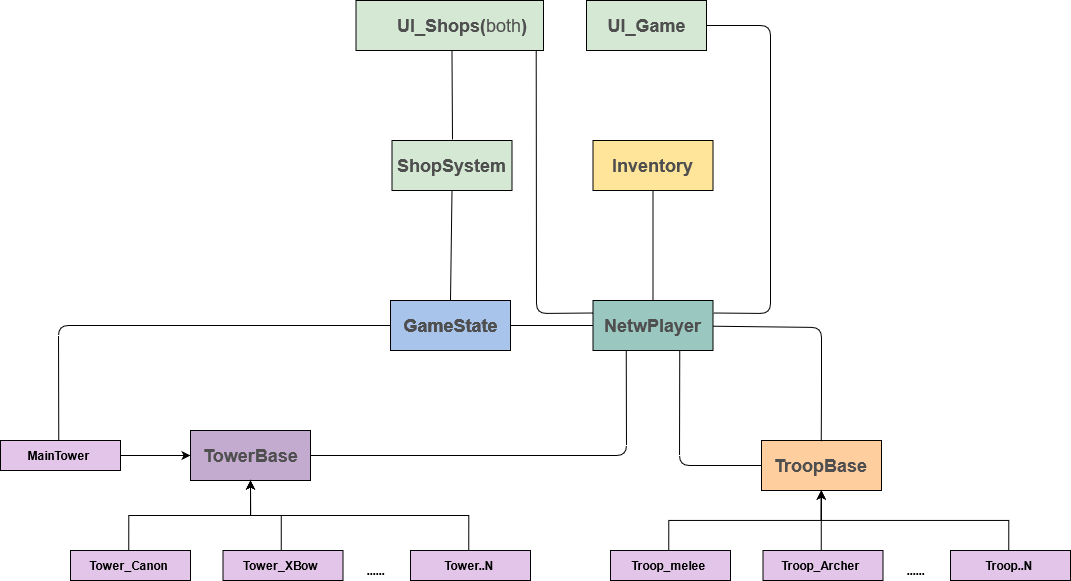
# Logical View

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.

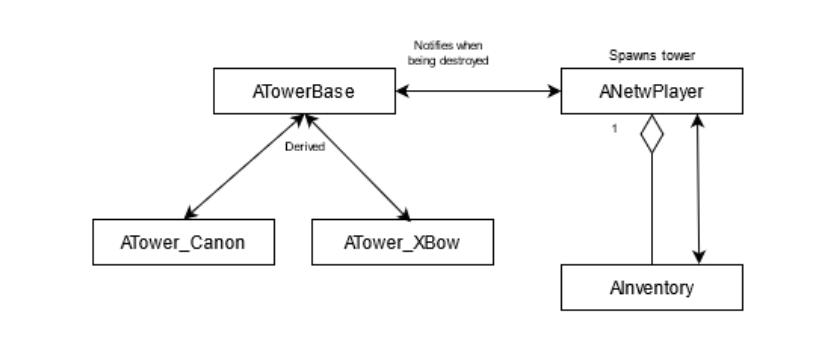
**GameState:** GameState is responsible to switch between UI widget based on current state of the game. It tells ShopSystem to display which widget based on the player's role(attacking/defending).It transitions between states like play, intermission(buying period),GameOver.It performs UI actions and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

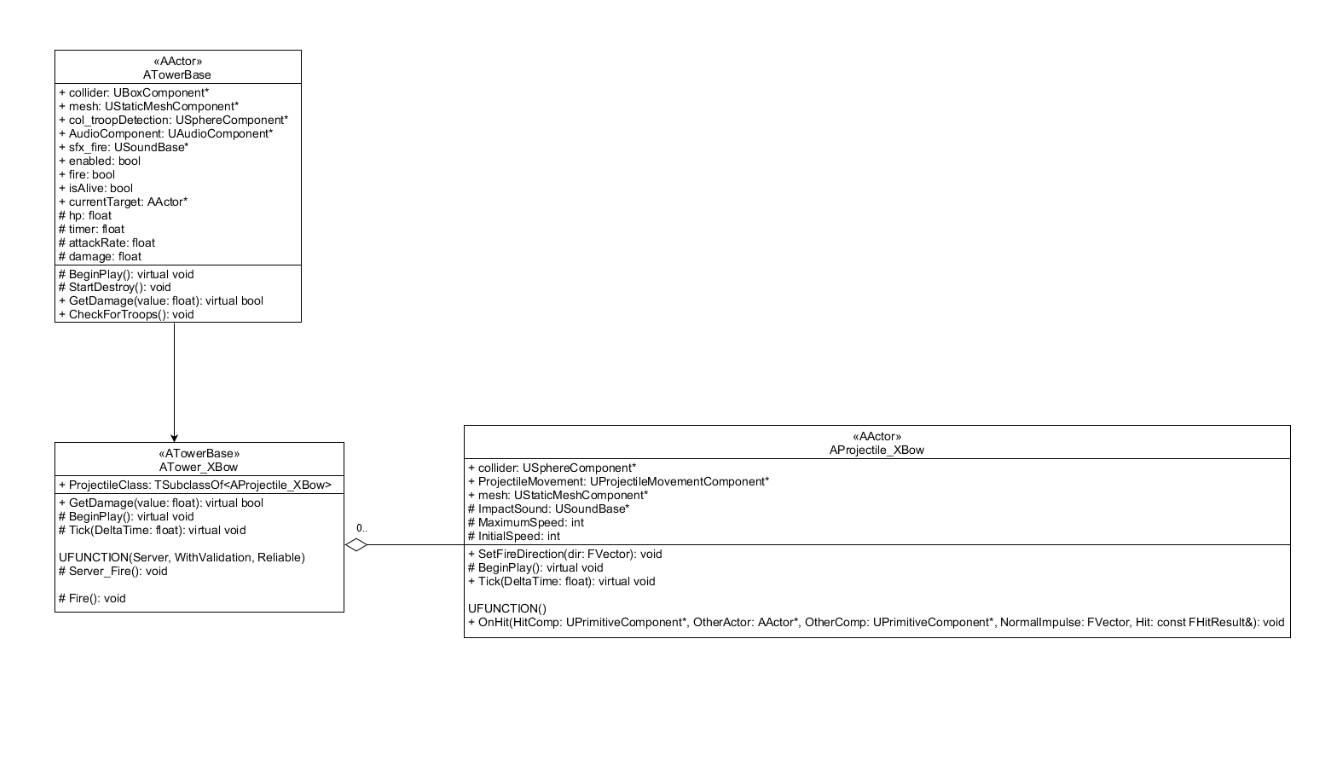
## Mid-Level Design of Module TowerAI



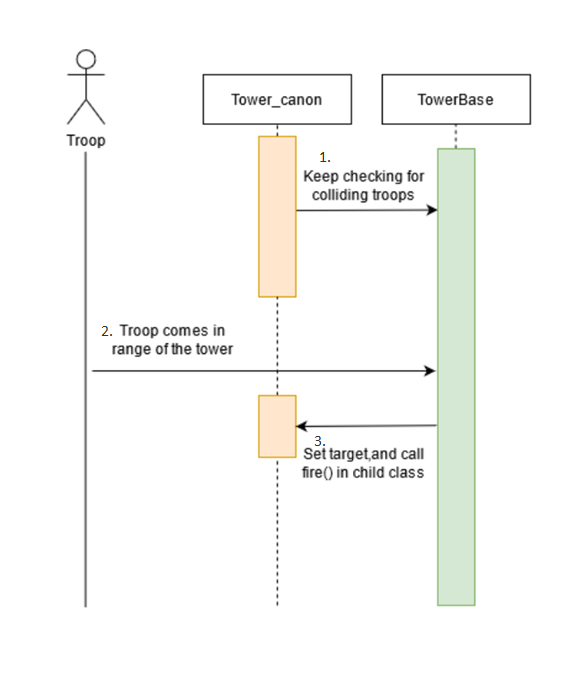
Each tower is derived from ATowerBase,it contains common variables, components and virtual function .

When ever a tower is being attacked, its GetDamage() is called, which also check if HP is less than 0,if it is it calls StartDestroy() which is in the base class, that calls OnUnitKilled(type) function ,which perform actions like rewarding the opponent and add the amount of gold to inventory.

## Detailed Class Design of Module TowerAI



# Process View of Module TowerAI



# Physical View (Applies to Multiplayer)

Tower is spawned by ANetwPlayer class, through a Server RPC which is called by client on mouse click.

Tower is set to replicate. And on detecting troop near, it calls fire() RPC on server.

# Use Case View

**Adding a new type of tower:**

1. Add a new class derived from TowerBase class, and a name that follows the naming convention: Tower\_name.
2. In the constructor start by setting mesh, assigning any collision function to the collider, set attack rate, tag and any other initialization if required.
3. In Tick(),call the parent’s function CheckForTroops() ,within conditions like isAlive.
4. And check the existing tower class to define other conditions.
5. Depending on the type of attack write the code for attacking troops in the Fire function.
6. Override the GetDamage() function .