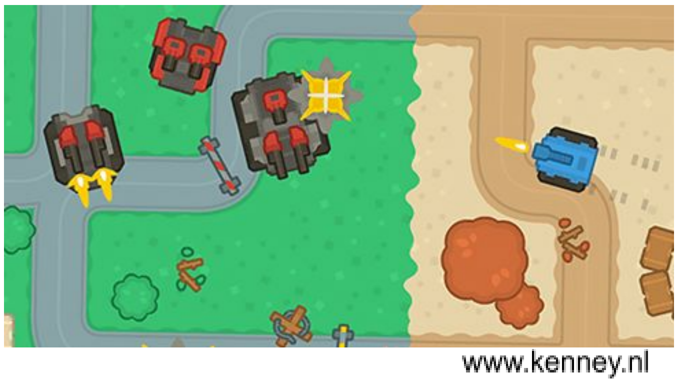
OOP Project 4

Summary

- Name: **BattleCOD 3: Eastern Front 2**
- Members:
    - Austin Albert
    - Andrew Hack
    - Ian Meadows

BattleCOD 3: Eastern Front 2 is a top-down, 2D survival tank shooter. Players attempt to stay alive for as long as possible while battling an unrelenting horde of enemies from the protection of a Panzer mk VI Tiger. Points will be earned for destroying enemies and surviving waves. Tank teams will be able to compare themselves and evaluate their performance with the game's scoreboard.

The game has 2 main components: the gamehost and the controllers. The host will display the game (game field, tank, enemies, health, points, etc) while the controllers allow players to operate the tank remotely. To force teams to work together, the tank controls will be distributed across multiple controllers. The game will support 1-6 players, and each player will run their controller from their own device. Possible controls include tank movement, weapon aiming, weapon loading, and shooting. Each controller will be assigned roles and can only command those specific tank features. Additionally, controllers will only show information relative to their roles - this prevents any one team member from knowing everything about the tank. In this way, the players will have to communicate to operate the tank and avoid blowing themselves up.



However, we have not forgotten our player base - we know that not everyone can pull together 5 (or even 1) friends for a game. The gamehost will dynamically allocate tank operator roles based on the player count. If only a single player is present, then all roles will be allocated to a single controller. To streamline the process, a single player will be able to control the game directly from the gamehost.

This action-packed shooter is guaranteed to ruin friendships and leave players bickering like married couples. It's minutes of fun!

Requirements

- Players must be able to work together to drive the tank and shoot enemies
- 1-6 players
- Local or remote play (allow input from host if player count = 1)
- Support multiple games on 1 network without cross-talk
- User-configurable control scheme
- Cross platform controllers (Win10, Ubuntu, Mac, Mobile)
- Gamepad support
- No crosstalk between tank operator roles (ie driver cannot shoot)
- Transparent controller reconnection (network failure recovery)
- GameHost
    - Host ArdNet server
        - Wait for client connections
        - Manage client roles
        - Listen for operator commands
    - Run TankSim game
        - Manage gameplay
            - Manage game field
            - Manage tank operation state
            - Manage enemy generation
            - Manage enemy AI
            - Manage scoring
                - Save/load scoreboard info to/from file
        - Respond to operator commands
- Client
    - Host ArdNet client
        - Connect to server
        - Push operator commands
        - Listen for operator commands
    - Load operator modules based on roles assigned from server
    - Handle user input
        - Capture input
        - Map input to operator commands
        - Generate tank command and broadcast via ArdNet
    - Handle UI notifications
        - Some operator roles are collaborative or time based. The UI should show some elements of the tank operational state to the relevant operators.

Use Cases

"Get Player Amount"
<u>Main scenario:</u>

1. User picks how many players there are.
2. Say go

<u>Extensions:</u>

1. 1 Player
   a. Can play game locally without controller client
2. N Players

"Role Division"
<u>Main scenario:</u>

1. Based on lobby size, roles will be divided accordingly
2. Wait for players to join lobby
3. Start game when all players have joined

<u>Extensions:</u>

1. If there is only one player they will be given all the rolls.
2. If there is more than one player, the roles will be divided accordingly

"Join Game"
<u>Main scenario:</u>

1. User types in room code
2. User types in user name
3. User will be assigned one role or many roles.

<u>Extensions:</u>

1. If game does not exist user is notified

"Tank Control"
<u>Main scenario:</u>

1. Server receives information from clients
2. The tank behaves according to information received:
   a. Moving
   b. Tank rotation
   c. Range finding
   d. Fire Control(fires gun)
   e. Gun loading

        f. Gun Rotation

<u>Extensions:</u>

1. Tank dies when the tanks health reaches 0

“Player Survival”

Main Scenario:

1. Player is alive
2. Wait a set amount of time
3. Spawn enemies based on how long the player has lived for

Extensions:

1. If tank dies then enemies stop spawning

“Score Submission”

<u>Main scenario:</u>

1. Tank has died
2. Go to score submission scene
3. User can enter a name
4. A score is generated based on how long the tank survives and enemies killed.
5. The score, name, and player amount will be submitted to a database.
6. High scores are shown

<u>Extensions:</u>

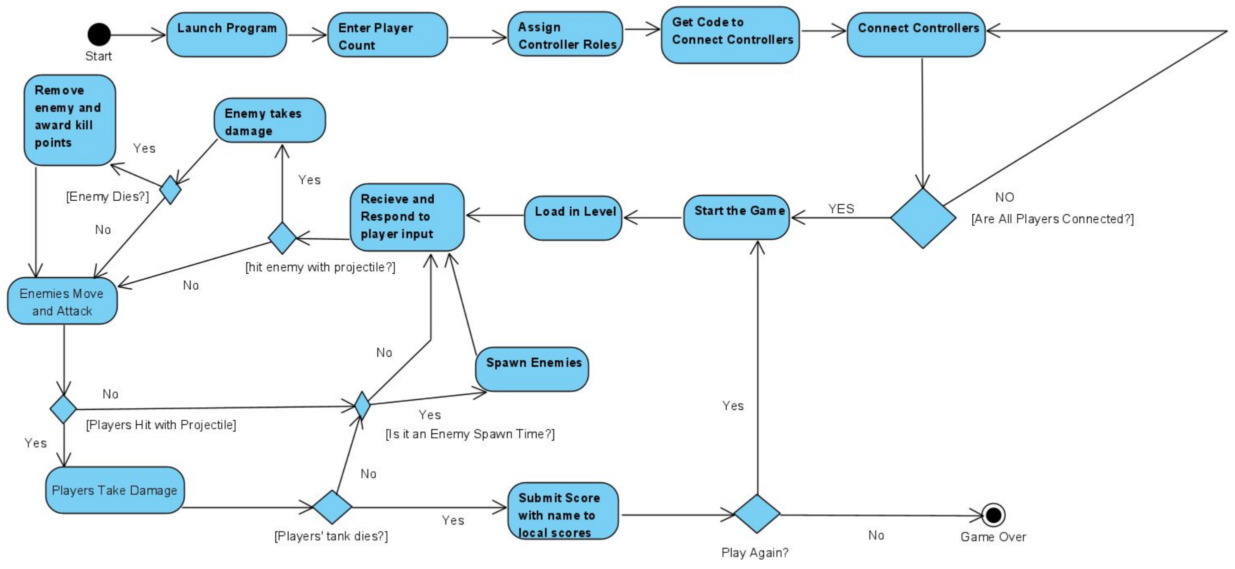
1. A multiplier will be added to the score based on how many players there are
2. User can either start another game or quit

“View High Scores”

<u>Main Scenario:</u>

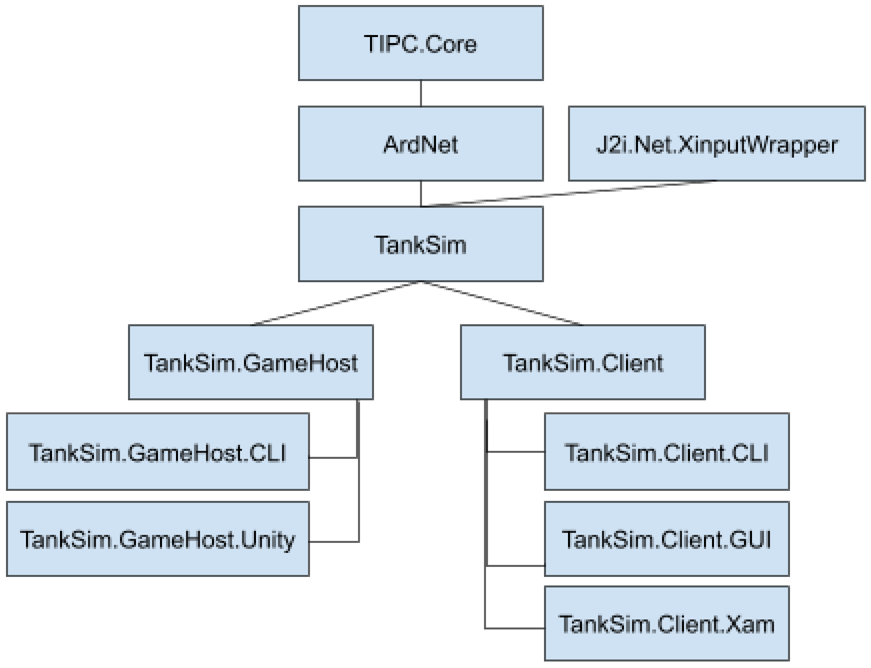
1. High score scene is entered
2. High scores are retrieved from database
3. High scores are shown
4. User leaves high score scene

Activity Diagram



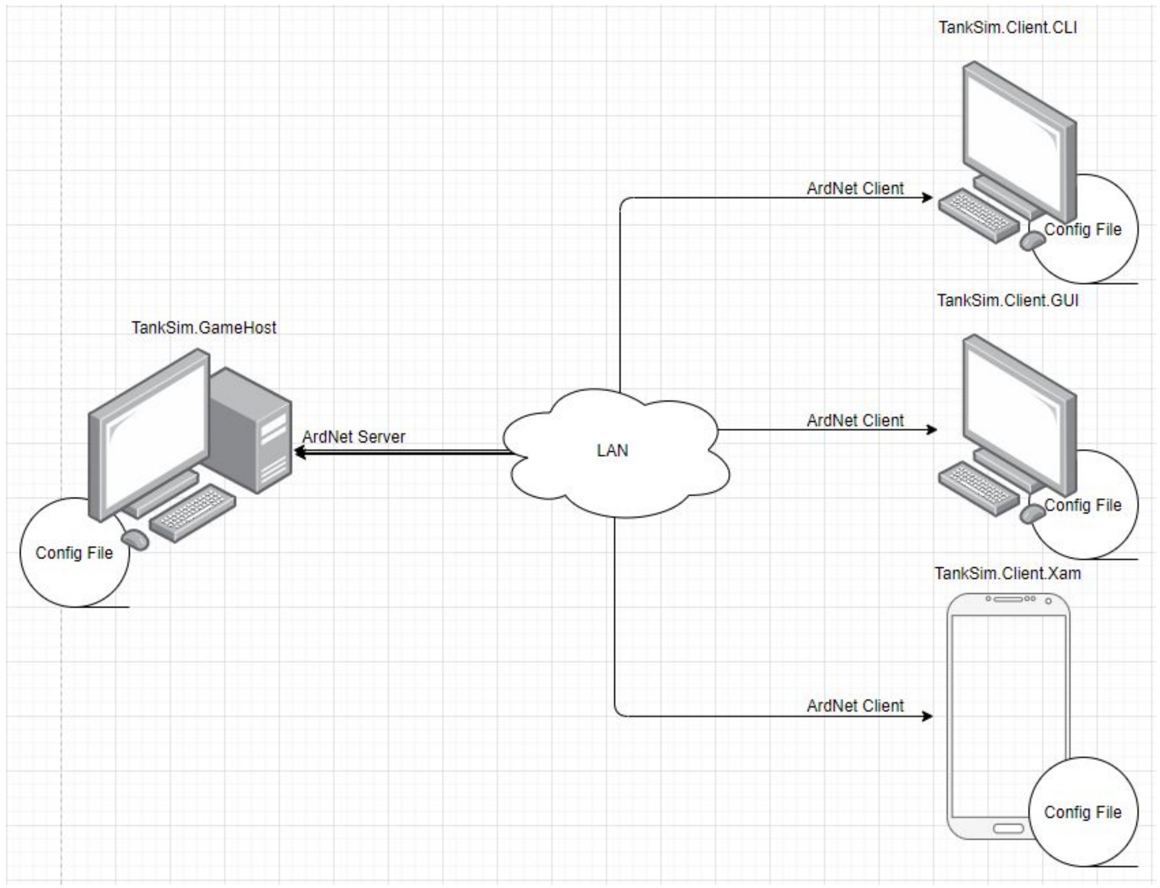
Architecture Diagram

Module Dependency Diagram



We will attempt to share as much code as possible between the various projects. All ArdNet channel watchers will be implemented in TankSim. TankSim.Client will implement client command components, leaving each client module to simply route user input to the underlying message handlers. TankSim.GameHost will wrap server setup and command aggregation.

System Interop Diagram



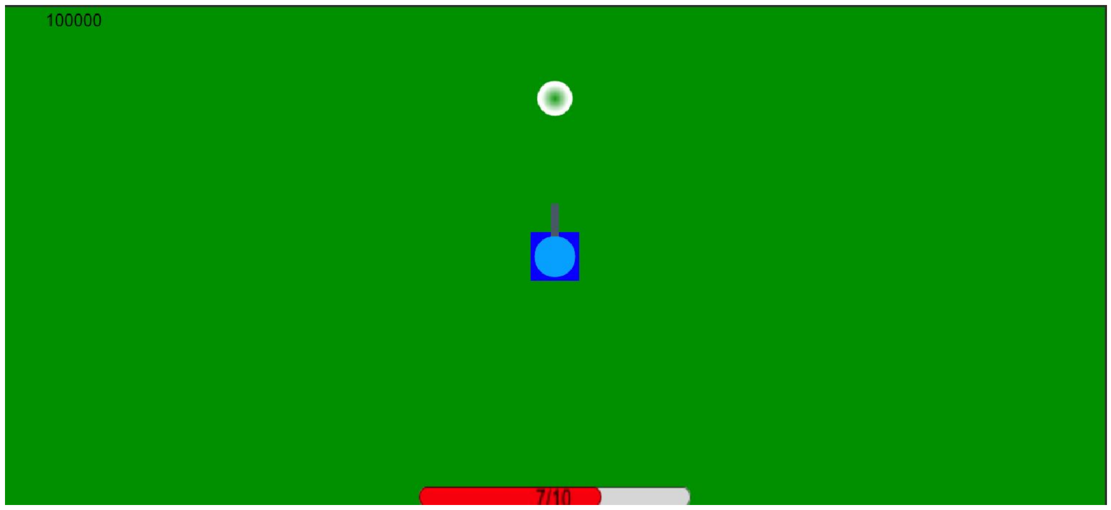
We will use ArdNet to send messages between the gamehost and connected clients. ArdNet supports basic data messages, request/response messages, and named channels. We will use a request/response sequence to synchronize new clients (get device info, assign user roles, mark as ready). After joining a lobby, clients will use named channels to broadcast command packets to other devices. Devices will listen to channels based on their assigned roles and process relevant commands. The GameHost will respond with tank actions, and clients with UI notifications.

Data Storage

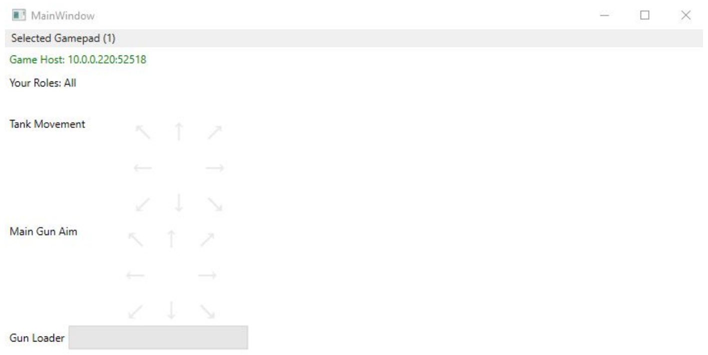
There is not much data that needs to be stored, the config files and high scores will be kept in Json files.

UI Mockups

Server view while in game:

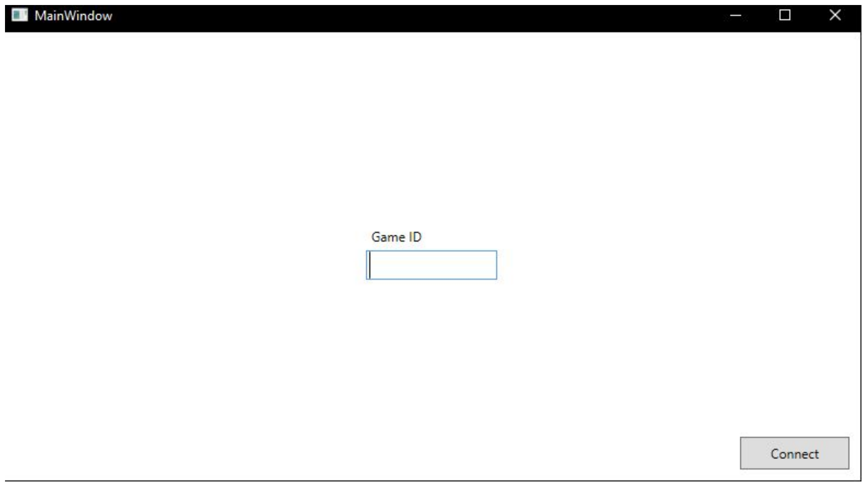


Client view while in game:

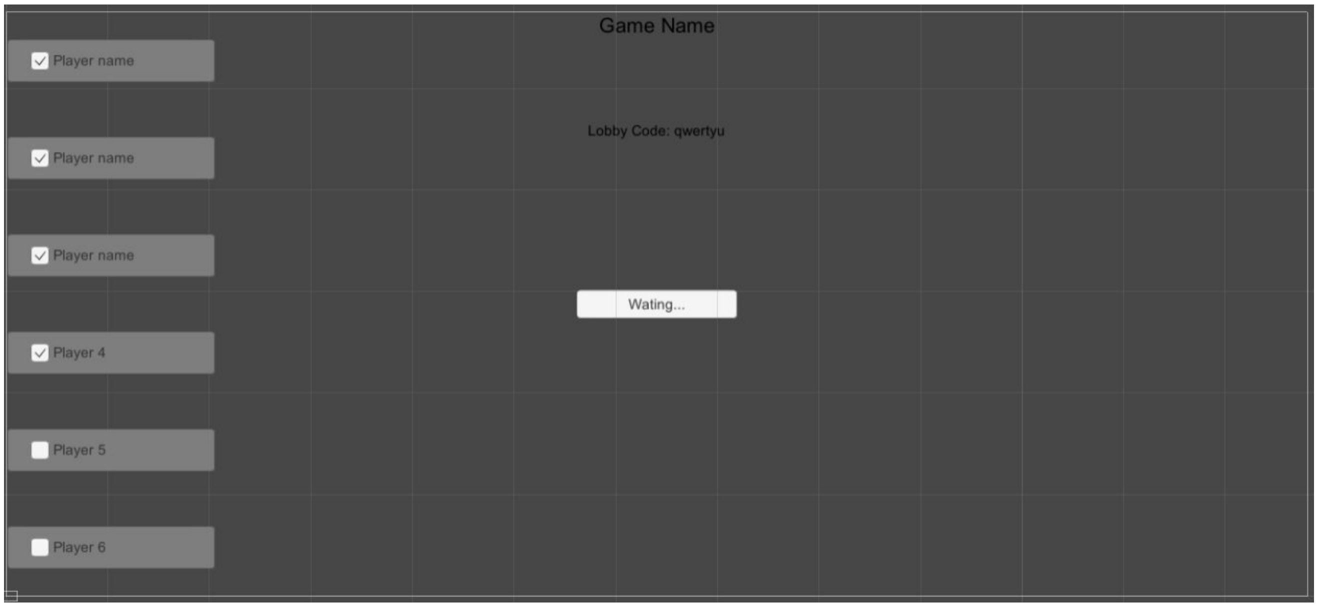


Selected Gamepad (1)
Game Host: 10.0.0.220:52518
Your Roles: All
Tank Movement
Main Gun Aim
Gun Loader

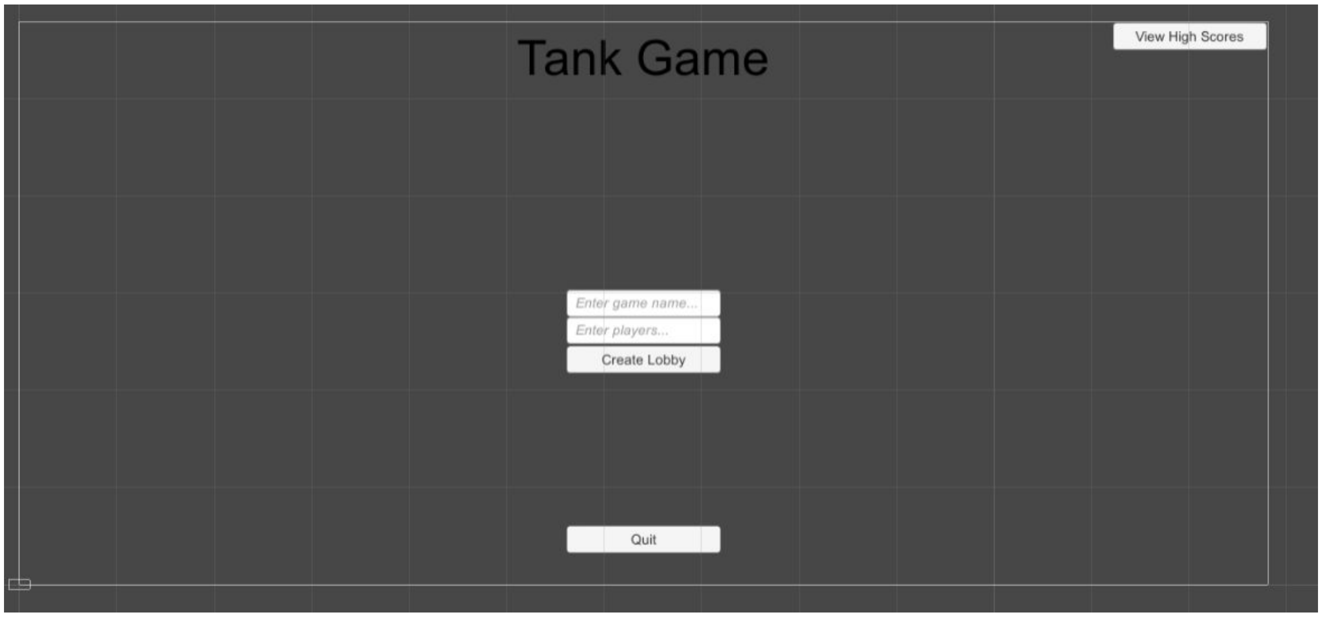
Client Joining a game view:



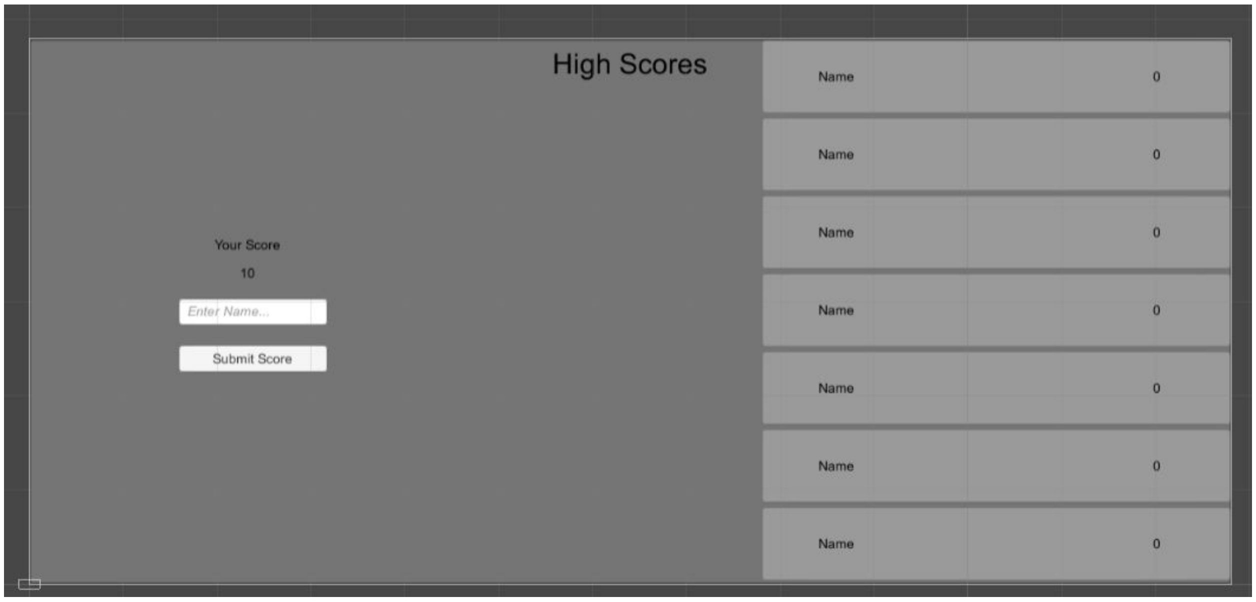
Server waiting view:



Server player setup view:

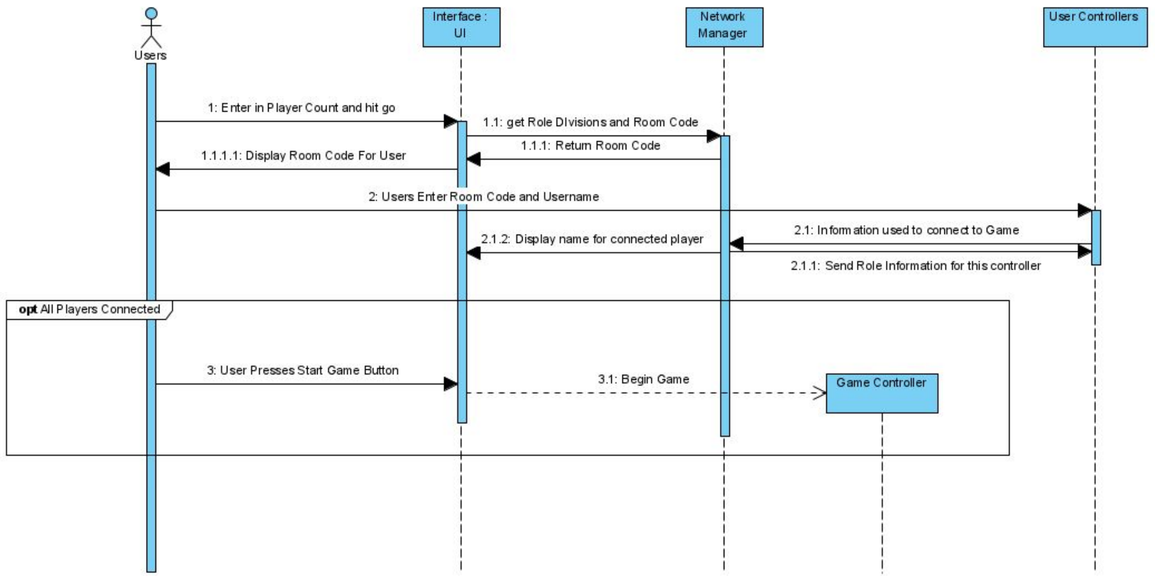


Highscore view:

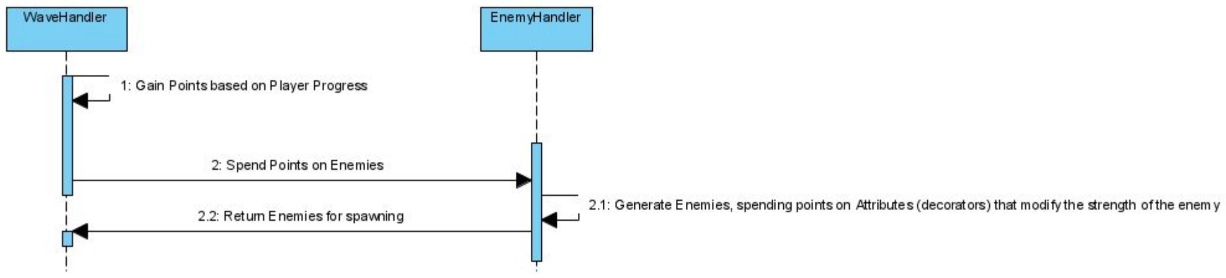


User Interactions/Sequence Diagrams

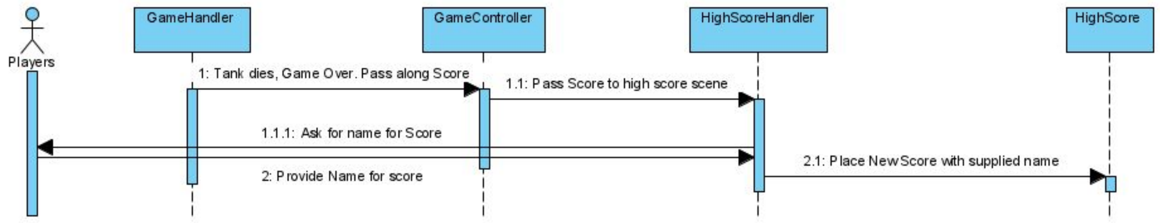
Interaction 1: Connecting Controllers to the game
The game will use a room code system to connect controllers to the game. First, the game needs to know how many players there will be. From there, the room code will be displayed and users can join. Once all users are in the game, the host can press the start game button to get into the action.


Users
Interface : UI
Network Manager
User Controllers
1: Enter in Player Count and hit go
1.1: get Role DIvisions and Room Code
1.1.1: Return Room Code
1.1.1.1: Display Room Code For User
2: Users Enter Room Code and Username
2.1: Information used to connect to Game
2.1.2: Display name for connected player
2.1.1: Send Role Information for this controller
opt All Players Connected
3: User Presses Start Game Button
3.1: Begin Game
Game Controller

Interaction 2: Enemy Spawns
While this is a simple interaction, it is extremely fundamental to our game. In order to have the game be able to go on indefinitely, given that your skills are good enough, the game needs to be able to come up with its own spawn patterns. This will happen regularly throughout the game, creating 'waves' of enemies.
The system is quite simple, applying decorators to enemies using a point-based system. The further the players get, the more points the game will get to spend, causing there to be a difficulty curve while allowing us to make the game theoretically infinite.

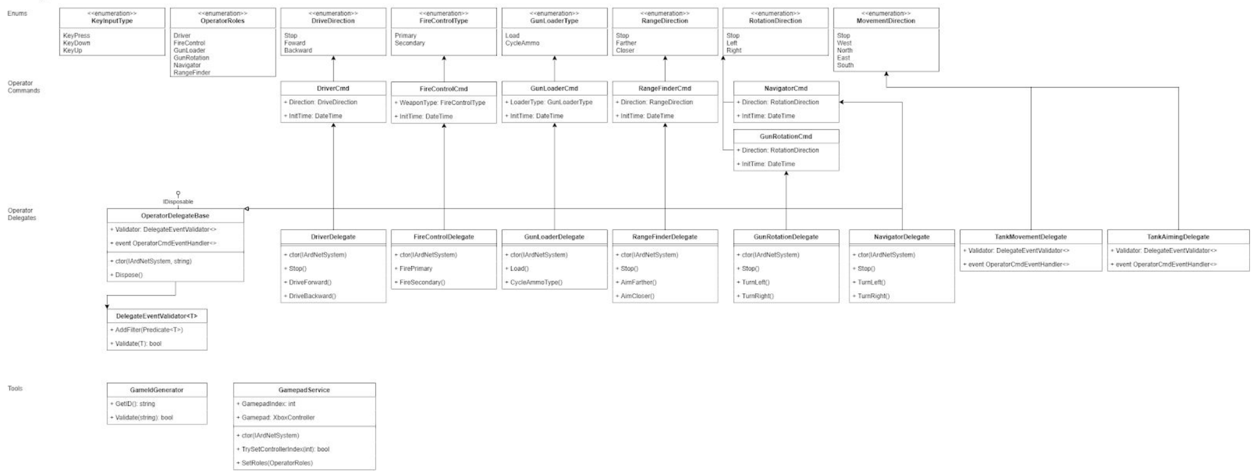

WaveHandler
EnemyHandler
1: Gain Points based on Player Progress
2: Spend Points on Enemies
2.1: Generate Enemies, spending points on Attributes (decorators) that modify the strength of the enemy
2.2: Return Enemies for spawning

Interaction 3: Submit Score
Once The player dies, they will be provided with a scoreboard of past attempts and an input box to give their high score a name.

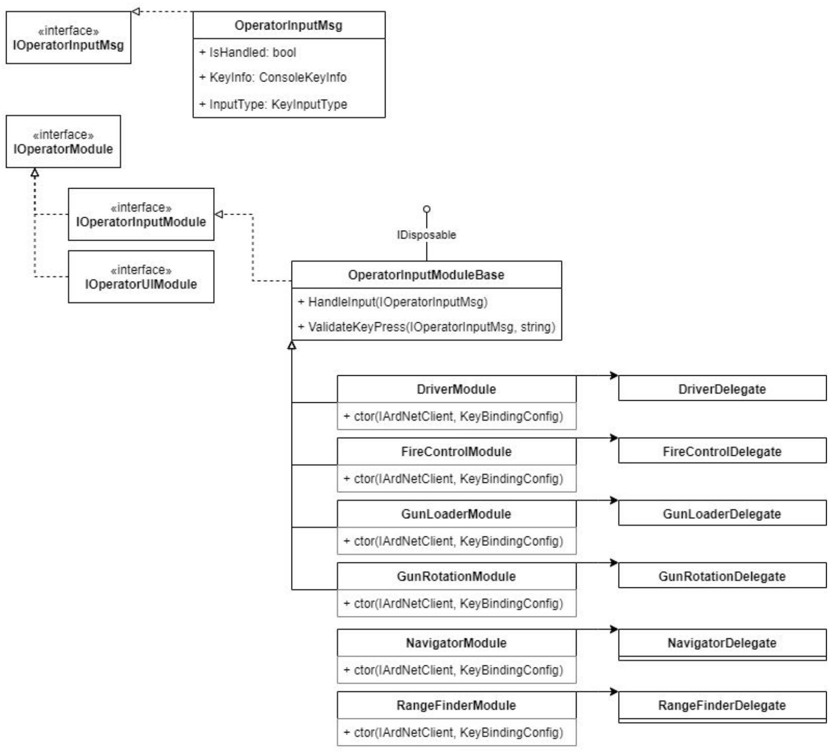

Players
GameHandler
GameController
HighScoreHandler
HighScore
1: Tank dies, Game Over. Pass along Score
1.1: Pass Score to high score scene
1.1.1: Ask for name for Score
2: Provide Name for score
2.1: Place NewScore with supplied name

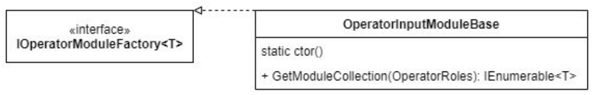
UML Class Diagram & Pattern Use

Package: TankSim

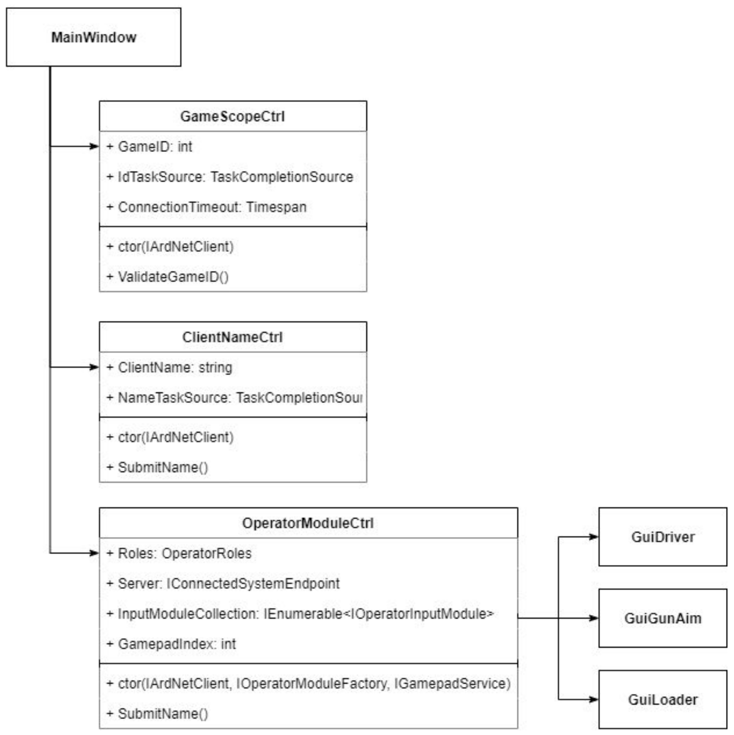


Package: TankSim.Client

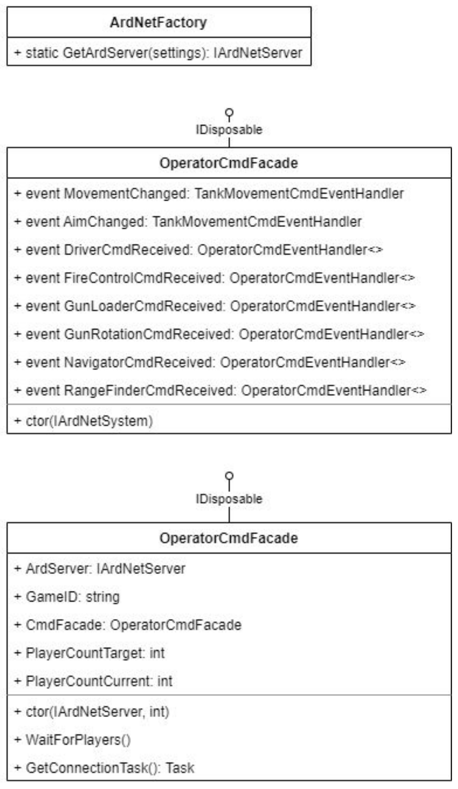

«interface» IOperatorInputMsg
OperatorInputMsg
+ IsHandled: bool
+ KeyInfo: ConsoleKeyInfo
+ InputType: KeyInputType
«interface» IOperatorModule
«interface» IOperatorInputModule
«interface» IOperatorUIModule
IDisposable
OperatorInputModuleBase
+ HandleInput(IOperatorInputMsg)
+ ValidateKeyPress(IOperatorInputMsg, string)
DriverModule
+ ctor(IArdNetClient, KeyBindingConfig)
DriverDelegate
FireControlModule
+ ctor(IArdNetClient, KeyBindingConfig)
FireControlDelegate
GunLoaderModule
+ ctor(IArdNetClient, KeyBindingConfig)
GunLoaderDelegate
GunRotationModule
+ ctor(IArdNetClient, KeyBindingConfig)
GunRotationDelegate
NavigatorModule
+ ctor(IArdNetClient, KeyBindingConfig)
NavigatorDelegate
RangeFinderModule
+ ctor(IArdNetClient, KeyBindingConfig)
RangeFinderDelegate

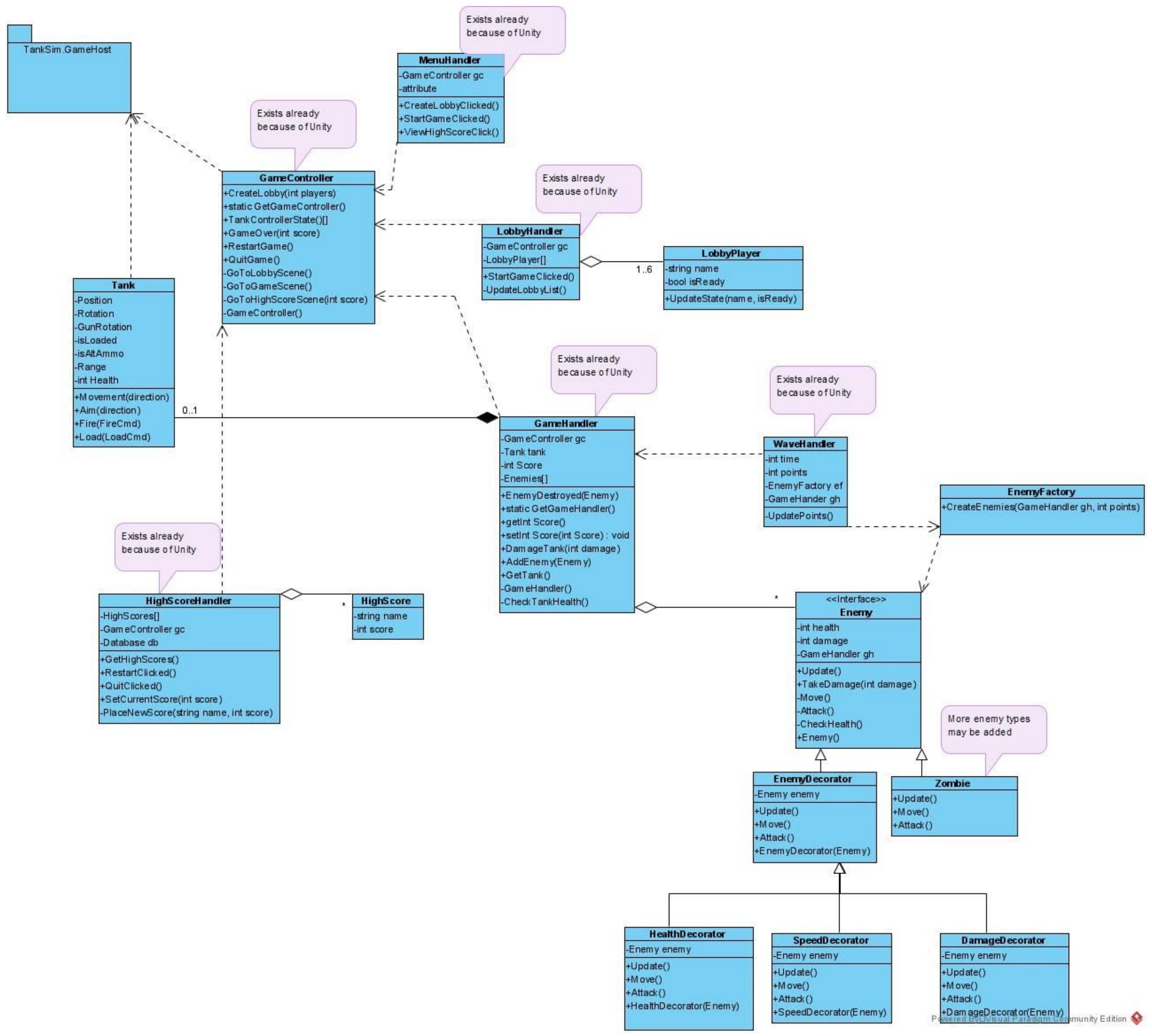

«interface» IOperatorModuleFactory<T>
OperatorInputModuleBase
static ctor()
+ GetModuleCollection(OperatorRoles): IEnumerable<T>

Package: TankSim.Client.GUI


MainWindow
GameScopeCtrl
+ GameID: int
+ IdTaskSource: TaskCompletionSource
+ ConnectionTimeout: Timespan
+ ctor(IArdNetClient)
+ ValidateGameID()
ClientNameCtrl
+ ClientName: string
+ NameTaskSource: TaskCompletionSour
+ ctor(IArdNetClient)
+ SubmitName()
OperatorModuleCtrl
+ Roles: OperatorRoles
+ Server: IConnectedSystemEndpoint
+ InputModuleCollection: IEnumerable<IOperatorInputModule>
+ GamepadIndex: int
+ ctor(IArdNetClient, IOperatorModuleFactory, IGamepadService)
+ SubmitName()
GuiDriver
GuiGunAim
GuiLoader

Package: TankSim.Gamehost


ArdNetFactory
+ static GetArdServer(settings): IArdNetServer
IDisposable
OperatorCmdFacade
+ event MovementChanged: TankMovementCmdEventHandler
+ event AimChanged: TankMovementCmdEventHandler
+ event DriverCmdReceived: OperatorCmdEventHandler<>
+ event FireControlCmdReceived: OperatorCmdEventHandler<>
+ event GunLoaderCmdReceived: OperatorCmdEventHandler<>
+ event GunRotationCmdReceived: OperatorCmdEventHandler<>
+ event NavigatorCmdReceived: OperatorCmdEventHandler<>
+ event RangeFinderCmdReceived: OperatorCmdEventHandler<>
+ ctor(IArdNetSystem)
IDisposable
OperatorCmdFacade
+ ArdServer: IArdNetServer
+ GameID: string
+ CmdFacade: OperatorCmdFacade
+ PlayerCountTarget: int
+ PlayerCountCurrent: int
+ ctor(IArdNetServer, int)
+ WaitForPlayers()
+ GetConnectionTask(): Task



Feature Highlights:

- Modular architecture
    - Multi-platform
    - Multiple UI implementations
    - Dynamic UI generation
- Networking
    - Async requests/commands
    - Async message channels
    - Transparent failure recovery
- Multi threading
- Lock-free thread safety
- Async message hub
- Reflection

- IoC containers
- Live config reloading
- Native API calls
- Gamepad support
- Global key hooks

Controller Patterns:
- Builder
- Factory
- Singleton
- Adapter
- Composite
- Facade
- Flyweight
- Proxy
- Chain of Responsibility
- Command
- Iterator
- Mediator
- Observer
- State
- Strategy
- Template
- Dependency Injection
- MVVM
- SOLID

GameHost Patterns:
- Decorator
- Many Singletons
- Factory