OOAD Project 6

Status Summary

Project Name: **BattleCOD 3: Eastern Front 2**
Team Members:

- Austin Albert
- ~~Ian Meadows~~
- ~~Andrew Hack~~

Github Link:
https://github.com/aual1780/OOP_BigProject
Demo Link:
https://drive.google.com/file/d/1Bch25FVB1-GUa_HctgGQQRsR2z7N34qp/view?usp=sharing

This assignment (Project 6) is being written by Austin Albert without the other team members. They are difficult to get involved and have not made substantial efforts for BattleCOD. They have made no contributions or attempts to communicate since Project 5. This is not due to a lack of work to be done - the github page has a considerable issue backlog.

Work Breakdown (Commits)
Austin: 100+
Ian: 16
Andrew: 9

Work Breakdown (Code Files Added)
Austin: ~100
Ian + Andrew: ~30

I did the vast majority of project planning, API design, and project implementation. As seen in our Project 5 submissions, BattleCOD is comprised of 7 first class modules (along with several secondary and tertiary modules). I completed 6 of these modules, some of which included the most technically complex components of the system.

The deal was that I would take the burden of the connectivity systems and controllers, and in exchange, Ian and Andrew would plan and build the game. They agreed to put in considerable effort to build a good game. Instead, the game component is rushed and haphazard. While the game does technically work, I am very disappointed in the end result and the lack of effort from my teammates.

Features that were implemented:

- ArdNet implementation [**Austin**]
  - Low latency networking solution for interacting between client and server
- Package: TankSim [Austin]
  - Extensible wrapper library to support code sharing between client and server
- Package: TankSim.Client [**Austin**]
  - Extensible client library to support multiple UI types. Includes fully encapsulated API endpoints for easily interacting with the networking subsystem. Manages game connection, failure recovery, command send/receive, dynamic role binding
- TankSim.Client.GUI [**Austin**]
  - GUI client to connect to the gamehost. Uses TankSim.Client subsystem. Player can join a game, set their name, and input commands. A simple HUD shows game state relevant to the user role. Includes gamepad integration
- TankSim.Client.CLI [**Austin**]
  - CLI client to connect to the gamehost. Uses TankSim.Client subsystem. Player can join a game, set their name, and input commands.
- Package: TankSim.GameHost [**Austin**]
  - Extensible server library to support multiple UI types. Includes fully encapsulated API endpoints for easily interacting with the networking subsystem.
- TankSim.GameHost.CLI [**Austin**]
  - Testing platform for server/client connectivity and event binding. Since both this and the Unity project share the same TankSim subsystems, we were able to test the process interactions separately from the game.
- Unity UI implementation [**Ian**]
  - Create a UI implementation in Unity using the TankSim.GameHost subsystem. This allows players to create a lobby, join, and start the game.
- Unity tank operation implementation [**Ian**]
  - Bind on-screen tank operations to the TankSim.GameHost API. This allows a tank to be draw on-screen and operated by the controller modules
- Unity enemy implementation [**Andrew**]
  - Create game enemies and define their gameplay rules. This includes enemy spawning, movement, attacks, and damage
- Unity game field features [**Ian, Andrew**]
  - Add features like barricades, sprites, etc, to the game field background

Features that were not implemented:

- Cross platform GUI client [**Austin**]
    - This was deemed low priority and removed from project scope
- Scoreboard [**Ian, Andrew**]
    - Saving score, viewing high scores, tracking scores tied to a given username
- Game stat tracking [**Ian, Andrew**]
    - Info about tank operations during the game
- Game prettification [**Ian, Andrew**]
    - We planned for the functional features to be finished early enough to leave time for fancy graphics. We planned for background scenes, a real tank sprite, bullet visuals, explosions, etc.
- Game QoL improvements [**Ian, Andrew**]
    - We planned several QoL improvements so the game would be enjoyable to play. As is, the gameplay is in an early beta state. The core game was rushed together for the mid project demo and has not been properly considered.
- The enemy system was not implemented as designed [**Andrew**]
    - In a meeting, we decided that the player team would fight enemy tanks. Andrew decided to use zombies instead, reasons unknown.

Class Diagram

The class diagrams have remained mostly consistent since their inception in Project 4. The largest changes are found in TankSim.GameHost and the Unity project. The GameHost library received some additions to internally manage more of the networking system. This simplified the startup process for consumers. The Unity project employed a “dynamic design” strategy (ie, neither Andrew nor Ian made a class diagram during Project 4) so there is no baseline to compare.

The core modules and controllers were feature complete at the end of Project 5 and did not receive major changes during Project 6 (only debugging).

TankSim.GameHost.Unity is not feature complete and also did not receive updates during the Project 6 development phase because the assignees are nowhere to be found.

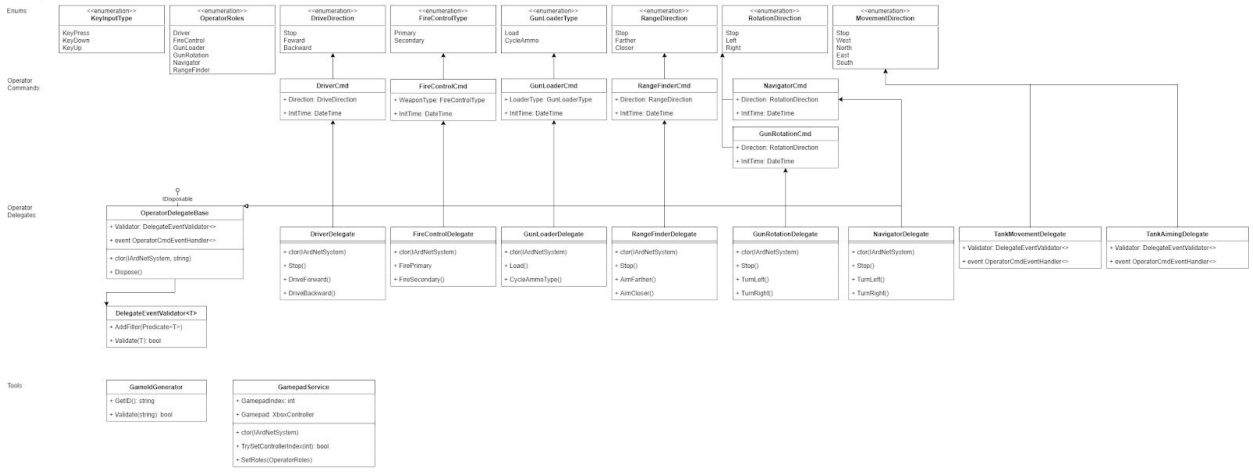
Unfortunately, this project is not conducive to diagramming. Every aspect of the project relies on some form of DI, service locator, or remote procedure call. That means that it’s impossible to get the full picture from a simple class diagram. Sequence diagrams and activity diagrams convey the system complexity more accurately.

Patterns

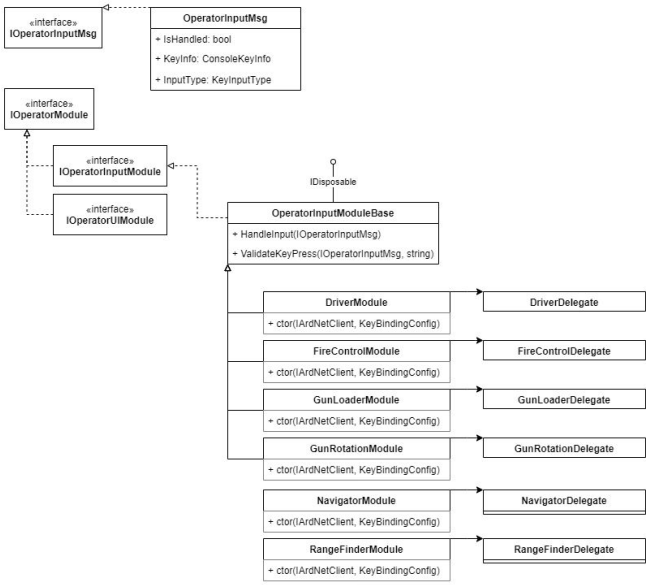
Mediator/Command: The core of the networking system is a mediator/command structure, but the complex parts are abstracted away by ArdNet, so it doesn’t appear in the diagrams.
DI/Flyweight/Singleton: The ArdNet system does appear in the diagrams in other ways, though. You will notice many ctors take an IArdNetClient or IArdNetServer as an argument. This is a shared object injected via DI Container wherever it is needed.
Proxy: We use proxy objects to interact with ArdNet named topic channels. This is another internal feature of ArdNet that we are consuming
DI/Builder: We use the ArdNet fluent builder API to prepare and configure the network interface. This also does not appear in the UML, but can be seen in the client startup code.
Facade: This pattern does appear in the UML. TankSim.GameHost.OperatorCmdFacade simplifies access to ArdNet event hooks to make them easier to access.
Chain of Responsibility: ArdNet.Client uses a CoR to send user input to a collection of operator processor modules. One (and only one) of the processors will respond to the input depending on what the user does.

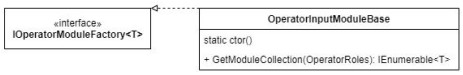
Original Diagrams

Package: TankSim



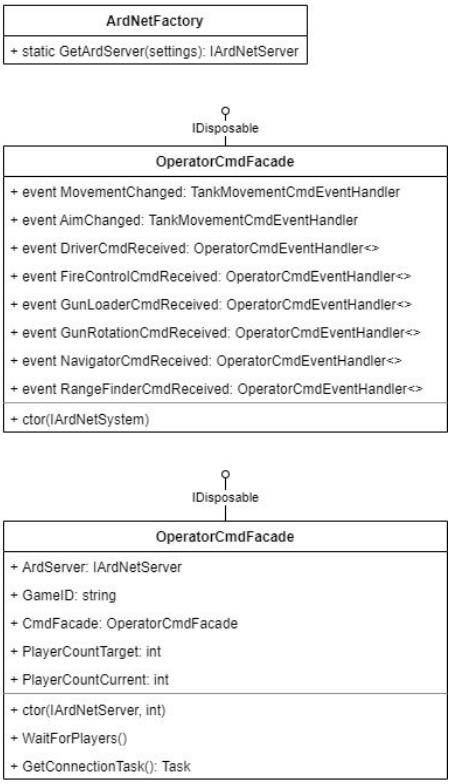
Package: TankSim.Client





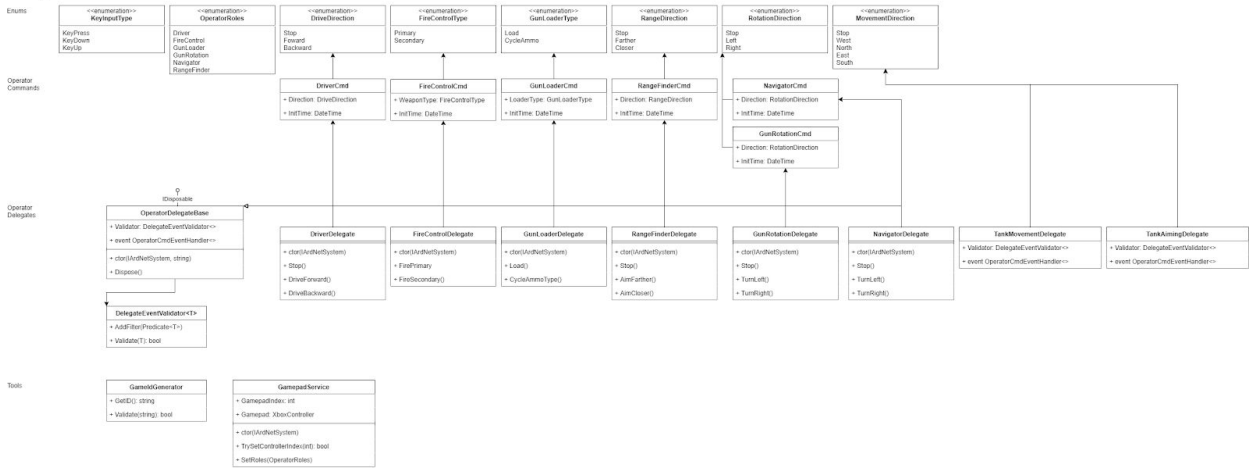
Package: TankSim.Client.GUI

Package: TankSim.Gamehost

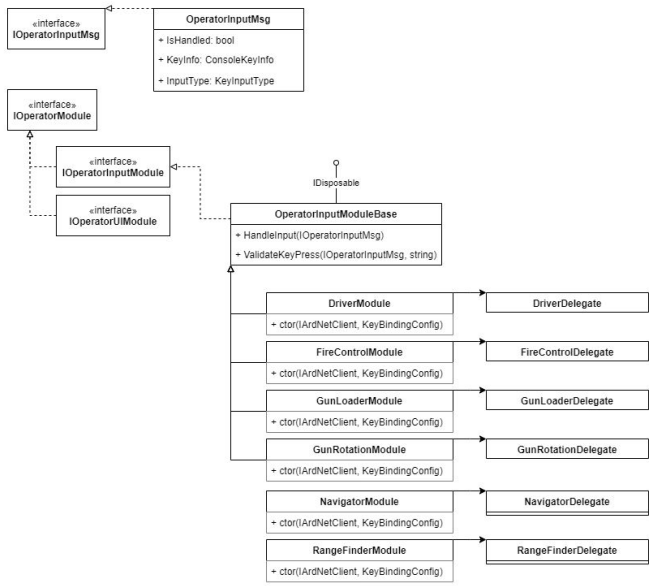


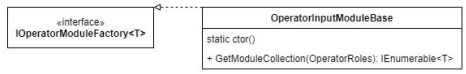
Current Diagrams

Package: TankSim

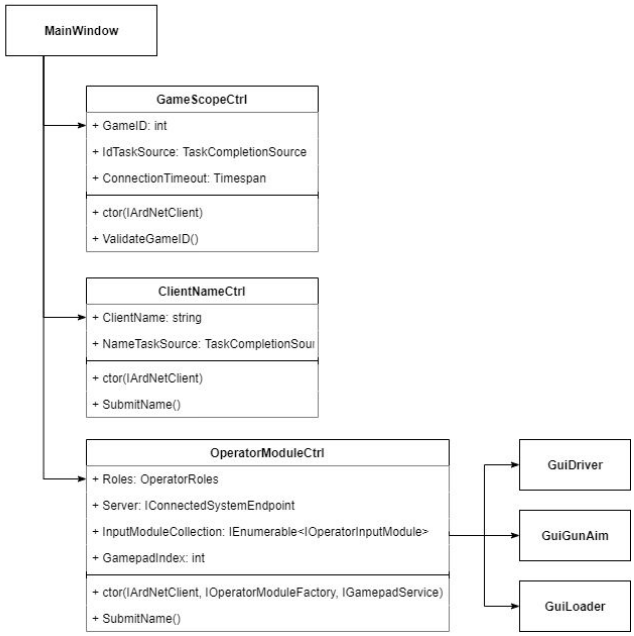


Package: TankSim.Client

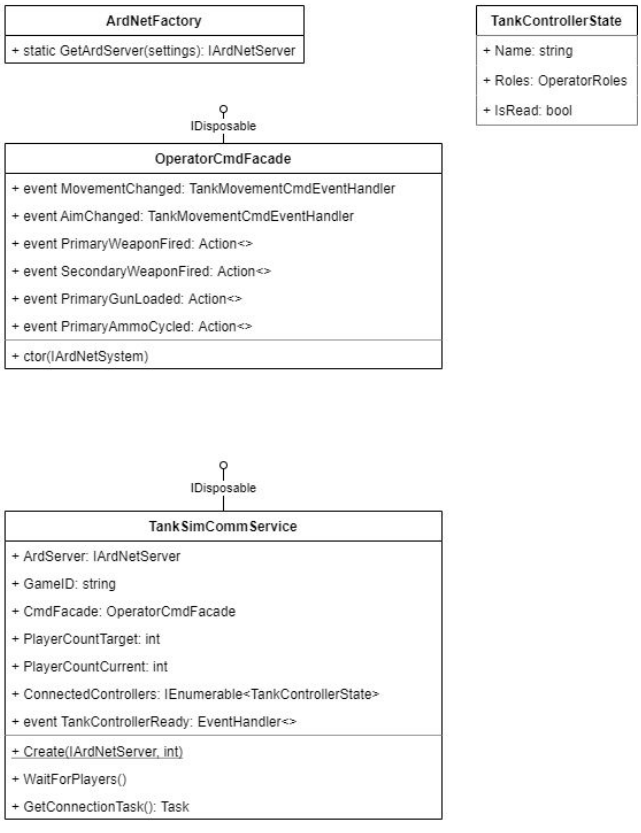


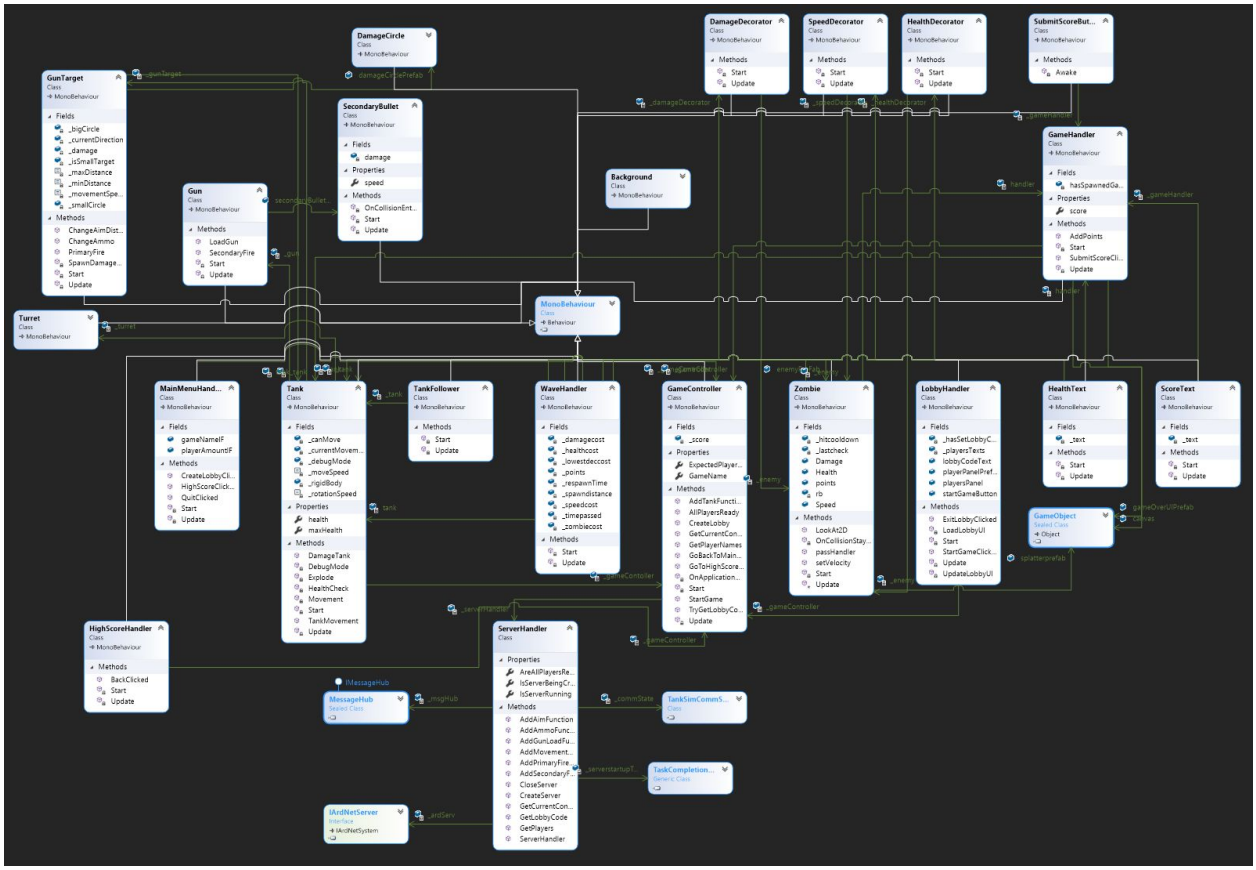


Package: TankSim.Client.GUI



Package: TankSim.Gamehost




DamageCircle
DamageDecorator
SpeedDecorator
HealthDecorator
SubmitScoreBut...
GunTarget
SecondaryBullet
GameHandler
Background
Gun
MonoBehaviour
Turret
MainMenuHand...
Tank
TankFollower
WaveHandler
GameController
Zombie
LobbyHandler
HealthText
ScoreText
GameObject
HighScoreHandler
ServerHandler
IMessageHub
MessageHub
TankSimCommS...
IArdNetServer
TaskCompletion...

Third Party Code Statement

We are using MS .Net and Unity to support our applications.

All TankSim packages are original work. J2i.Net.XInputWrapper was originally taken from a tutorial at one point many years ago that has since been lost. The project relies heavily on the ArdNet and TIPC.Core libraries - these are maintained by Austin, but are considered out of scope for this project. For a more indepth listing, see the github dependency insights.

I assume that many sections of the Unity solution were pulled from tutorials, but as I was not heavily involved in that side of the project, I do not know the sources.

OOAD Design Process

The initial design was a simple 3 component architecture diagram: Client, Middleware, and Server. The middleware component was the first to be outlined. This included networking spec, message types, service endpoints, and rough consumer API. Having all of this in place early allowed the 3 components to be siloed from each other. As long as the middleware networking remained stable, the client and server could be implemented separately from each other. This allowed the project to advance even if some components lagged behind.

Immediately following the architecture diagram was a storyboard combining sequence diagrams, activity diagrams, and UI diagrams. I believe these aspects are more important than class diagrams or class breakdown structures during the early process. This allowed us all to get on the same page about the project. Since the server and client were isolated so well, we were able to continue with more detailed planning as separate teams. In theory, it had all the ingredients to work perfectly. In practice, I question the hands-off approach I took with the rest of the team given their lack of responsibility.

TankSim.GameHost.Unity suffered from a severe lack of planning. Andrew and Ian did not plan ahead during Project 4. Their contribution to that project was essentially copying my existing work - they took my initial pencil sketches and digitized them without necessarily understanding them. Even worse than not planning the code, they also did not plan the game itself. So the rules and interactions were haphazardly thrown together during build-out. This all cultivated in a poorly coordinated, poorly built demo for Project 5.