BattleZone Client Reloaded

Everett Brown Jr., Jordin Carey, Brendon Conlin, James Connolly, Deven Deming, Brady Quigley
Computer Science, University of New Hampshire, Durham, NH 03824

Introduction

- UNH's CS619 (Object-Oriented Design and Development) uses an outdated codebase, presenting technical overhead for students.
- The codebase of CS619 should be built using modern frameworks and design practices and include relevant documentation to aid its users.
- By refactoring CS619's codebase, students will spend less time grappling technical issues, and more time learning and developing OOD skills.
- The refactored codebase should not increase course milestone completion time, and it can use modern tools such as Dart and can be extendable and modular for future students to access modern features.

Requirements

Authentication

- Allow users to create and/or login to an account to store game statistics.
- Securely store username/password combinations.
- Utilize a session token to enforce successful authentication across server calls.

Gameboard

- Map a collection of server values to a defined grid cell type, displayed as a 2-D array of grid cells (16 x 16).
- Provide users with a control panel, allowing commands to be sent to the server (e.g., move or fire).
- Periodically (100–500 ms, across implementations) request game data from the server to update client state/UI.
- Provide developers with a toggle switch, allowing them to update the grid by requesting the full grid from the server, or by using game events.

Events

- Process a collection of game events to efficiently update the internal grid.
- Utilize the publisher/subscriber model to let the gameboard subscribe to published events and update its own UI.
- Define a database, allowing retrieved events to be stored for later use.

Non-Functional Requirements

- All tools (framework, coding languages, server communication, etc.) are up to date and well-documented, e.g., showing recent updates and providing usage guides.
- The codebase is extendable, allowing new additions to be easily integrated.
- The codebase can be used and run on a variety of platforms (Linux, Mac, Windows).
- Logging is used to output the server stream, and for debugging.

Design

Server: The system that stores global gameboard and state, and processes incoming requests from the Client

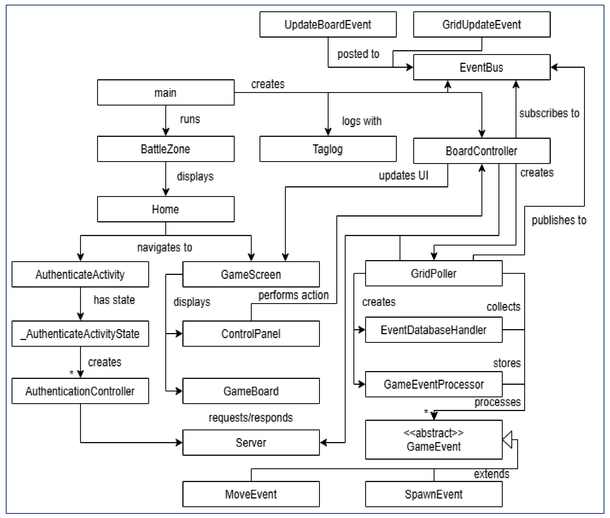
gRPC: A remote procedure call framework that uses Protocol Buffers to define structured communication between the Server and Client.

Dart: The main programming language used to define Client-side logic, including gRPC calls and UI behavior with Flutter.

Flutter: The user interface framework that renders the game interface, including the gameboard, controls, and login screen.

Client: The Flutter application that runs on the user's device and communicates with the Server via gRPC to send and receive game updates.

BattleZone Class Diagram



Implementation

Widgets

The Flutter building blocks used to create the application's user interface (e.g buttons, grids, etc.). This includes aspects like Containers, which are used to structure applications: like spacing between grid cells and positioning of UI elements.

Authentication

The AuthenticateController class processes login/register requests and sends them to the server. The client receives a JWT token for the user that is used for all calls to the server.

Gameboard

The Gameboard uses the GridView Flutter Widget to initialize a 16x16 grid which is constructed with various Widgets to display different visual icons to denote different game entities.

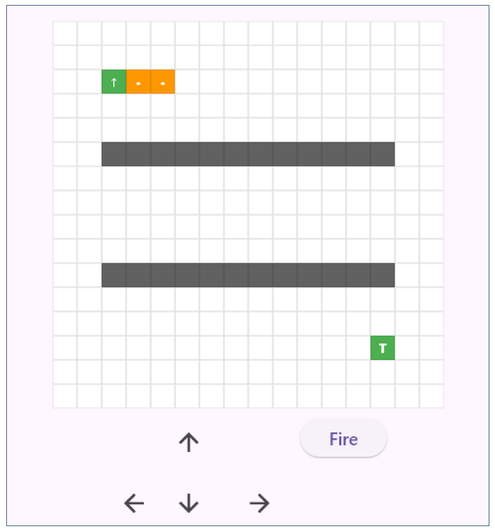
Poller

The Poller uses Timer. periodic from the Async library in Dart to call the server every 100 milliseconds to receive an update of the entire grid or an update via events.

Event

Using EventBus and custom-defined events, the Poller can publish a GridUpdateEvent. The BoardController class subscribes to these events to set the board and update UI.

BattleZone Gameboard



2-D, grid-based gameboard with controls

Testing

gRPC Mock Sever

Designed a Dart-based gRPC server with protobuffer stubs to simulate game states changes through board data and events. This was developed incrementally to test specific behaviors, primarily sending mocked boards.

Logging

Custom TagLog created to log information from separate classes with timestamp for testing. Client logging outputs to terminal, with server logging to debug console.

Unit/Integration Testing

Dummy server tests for connection with remote server. Unit tests created for Client UI and logic, following tests required from students of the CS619 class.

Logging Output

```
[I] TIME: 2025-11-30T11:42:32.997956 [POLLER] Received JSON update)
[I] TIME: 2025-11-30T11:42:32.997956 [EVENT] New BoardUpdateEvent
[D] TIME: 2025-11-30T11:42:33.267833 [GRIDSCREEN] Button 2 pressed
[I] TIME: 2025-11-30T11:42:33.267833 [POLLER] Polling stopped.
[I] TIME: 2025-11-30T11:42:33.267833 [GRIDSCREEN] Polling paused
[I] TIME: 2025-11-30T11:42:34.726311 [GRIDCELL] Selected cell at index 167
```

Evaluation & Conclusions

Authentication: Users can create account to login securely; session token is in development.
Gameboard: Server values can be altered, retrieved, and displayed on a grid-based gameboard.
Events: Event logic can update the grid and UI; a database was created (using SQLite3); game event processing is in progress.

- Proof of concept (grid-based display via server values) successfully delivered December 2025, MVP delivery for May 2026.
- Documentation of design choices and tools have been gathered for future students.
- Flutter/Dart made solo project milestones and BattleZone starter code easier to implement.
- Solo project milestone time should be reduced (from independent testing).
- Flutter/Dart and gRPC involve less overhead than the former tools and should reduce debugging time.
- Future work for the project would include providing implementations for all Battlezone milestones, using the updated codebase.

Acknowledgements

We would like to thank the project sponsor, and current CS 619 instructor, Matthew Plumlee, as well as David Benedetto, our project advisor. Additionally, for their guidance and support on this project, we thank the UNH CS and IT Department.