Jayant Dharwadkar

I WROTE:

- AGENT.C
- FIELD.C
- AGENTTEST.C
- FIELDTEST.C

No extra credit was implemented.

Introduction: In this lab, I took on the solo option to develop a two-player game of BattleBoats. The lab was designed to teach us about communication protocols, state machines, random number generation, and teamwork. As a solo participant, I faced a unique challenge of implementing two core modules and their respective test harnesses. This lab was by far the most demanding in the course, requiring a significant amount of time and intellectual effort.

Overview of the BattleBoats System: The BattleBoats system is an event-based framework where different modules exchange information through events. The main components are:

1. Transmission Service: Handles wired UART communication between agents.
2. Button Service: Detects button events.
3. Agent Service: Manages high-level game playing and user display.

These services depend on four submodules, two of which I implemented:

1. Agent (implemented): A state machine that controls the game flow and UI.
2. Field (implemented): Manages game moves and board states.
3. Message: Translates between UART characters and data structures.
4. Negotiation: Handles the cryptographic scheme for deciding the first player.

The game follows rules similar to Battleship, with a few modifications like different grid dimensions and a more complex method for determining the first player using a commitment scheme.

What Worked Well:

- Modular design: Even as a solo participant, the separation of concerns made it easier to understand and test each module independently.
- Test-driven development: Writing test harnesses for my modules helped me catch bugs early and understand the expected behavior thoroughly.
- State machine diagrams: These were incredibly helpful in understanding the flow of the game and implementing the Agent module.

What Didn't Work Well:

- As a solo participant, managing the workload of two core modules and their tests was challenging.
- Debugging the Agent state machine was complex, especially when integrating with the Message and Negotiation modules I didn't implement.

Testing Strategies: For FieldTest.c, I created an interactive play session where I could manually place ships and make guesses. This helped me validate the Field module's logic in a more intuitive way. In AgentTest.c, I simulated various game states by directly passing events to AgentSM(), mimicking the Transmission and Button services. This allowed me to test how the Agent reacted to different game scenarios.

What I Learned:

- The intricacies of designing and implementing a complex state machine (Agent).
- Strategies for managing game state and AI decision-making (Field).
- The importance of thorough testing, especially when modules need to interact with others you didn't write.
- How to manage a larger workload independently, a skill that will be valuable in future projects.

What I Liked About This Lab:

- The challenge of implementing core game logic and decision-making.
- The opportunity to work on a project that mimics real-world complexity.
- Gaining a deeper understanding of state machines and event-driven systems.

What I Would Change:

- Provide more guidance on testing strategies for solo participants, especially for interacting with modules they didn't implement.
- Include more intermediate milestones to help manage the project's complexity.
- Offer an optional workshop on state machine design and debugging.

In conclusion, taking on this lab solo was a significant challenge but also an incredibly rewarding learning experience. It pushed me to deeply understand and implement key components of a complex system. While it was more work, I believe the skills I've gained in independent problem-solving, system design, and testing will be invaluable in my future courses and career.