EMBEDMON: THE GAME

Embedmon Team
Anton Njvaro (njavro@bu.edu)
Roman Velez(rgva@bu.edu)
Atharva Khandekar(amk14@bu.edu)
https://github.com/njavro/EC535

Abstract

Embedmon is a multiplayer game developed for the BeagleBone platform using the Qt framework, with a focus on simplicity and robustness. The game features two players engaging in a turn-based battle, with each player choosing from a set of four actions: light attack, strong attack, block, and potion. The game utilizes Ethernet networking through TCP to establish a connection between two BeagleBone devices, with one acting as a server and the other as a client. The server manages the game's primary logic, while the client sends and receives action data. The game's graphical user interface is implemented using the Qt graphics library, allowing for intuitive controls and polished visuals. Despite initial challenges with Ethernet networking and integrating C code with the Qt framework, the final implementation delivers a stable and enjoyable multiplayer gaming experience on the BeagleBone platform**.**

1. Introduction

Embedded systems have become an integral part of modern life, providing the foundation for a wide range of applications and innovations. One such application is the development of interactive games tailored for embedded platforms, taking advantage of the unique features and constraints of these systems to create engaging and enjoyable experiences [1]. In this project, we present Embedmon, a multiplayer game specifically designed for the BeagleBone platform, with an emphasis on simplicity, robustness, and captivating gameplay.

Embedmon is centered around two players who engage in a turn-based battle, where each player has a choice between four distinct actions: light attack, strong attack, block, and potion. Each action has its own unique effect on the game, influencing the outcome of the battle and requiring strategic thinking from the players. The game employs Ethernet networking to establish a connection between two BeagleBone devices, with one device acting as the server and the other as the client. The server is responsible for managing the game's primary logic, including verifying action inputs, executing the round, and updating the user interface to reflect the current state of the game. Meanwhile, the client is responsible for sending and receiving action data, ensuring seamless communication between the two devices.

The development of Embedmon involved a series of design decisions, challenges, and iterations, ultimately leading to the creation of a LAN-based game that leverages Ethernet cables for communication between the BeagleBone devices. To build the game, we utilized the Qt framework, a comprehensive suite of tools that simplifies the development of graphical applications and the integration of networking functionality [1]. We developed two separate Qt applications - EmbedmonServer and EmbedmonClient - to handle the game's primary logic and facilitate communication between the devices. The graphical user interface for the game was designed using the Qt graphics library, providing players with an intuitive, visually appealing, and polished experience.

In this report, we look into the creation of Embedmon, examining the difficulties encountered during the project and the solutions put in place to get over them. We go over the several design factors, the technology and tools selected, the creation of the game's fundamental mechanics, and the implementation of the networking feature. We want to provide readers with a thorough grasp of how to develop a reliable, interesting, and pleasant multiplayer game on the BeagleBone platform, emphasizing the insights and lessons we learned along the way.

2. Design Flow

Figure 1 describes the overall structure of our project. A few of the main hardware components of the project are: 2x BeagleBone Black, 2x LCD screens, and Ethernet cable. The LCD screens and BeagleBone devices were paired up as recommended by Lab 5 instructional documentation. We utilized the available Ethernet port in order to achieve connectivity. An instance of server is running on one beaglebone, while an instance of client is running on another device. These devices have their networking configurations hard coded as in this version as we did not implement dynamic discovery of neighboring devices.

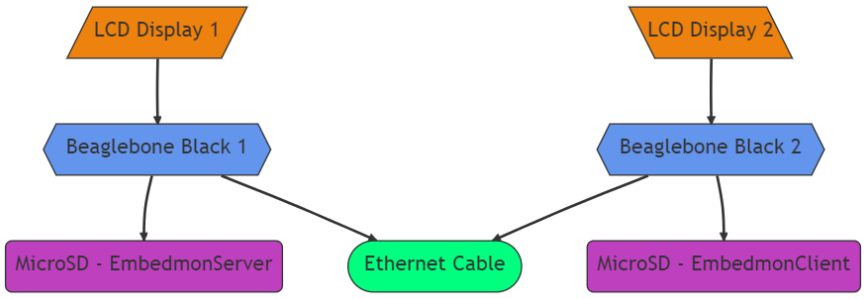

Figure 1: Flowchart of the physical connection between the server and client Beaglebones.

Since the majority of game and networking logic is contained within a single QT program, there is not much additional information besides what was and will be provided in following sections. Roman did a great bulk of the work implementing gameplay logic in QT along with the latter part of networking in QT and visual design (including player sprites, and UI). Anton worked on initial C implementation of networking with sockets and implemented an initial QT networking prototype for server-client communication, along with partial help in debugging at latter stages. Atharva worked in support of the game development for the game mechanics, but was unable to use the Qt IDE due to technical issues.

Contribution Breakdown:

- Roman: 55%
- Anton: 30%
- Atharva: 15%

3. Project Details

Networking:

During the ideation phase of our project, we settled on the idea of a multiplayer game - which went through several iterations and design decisions. Our initial plan involved connecting two BeagleBone boards to the internet and establishing a remote connection to the server that we would have set up. The goal was originally to have the server in the cloud running all of the game logic while the BeagleBones would send and receive the data over the internet. Not long after we have realized that achieving that over the internet would cause multitude of issues which would be unfeasible to solve in the limited amount of time we had. We then opted to turn Embedmon into a LAN game in which the Ethernet cables we had would be used to connect two BeagleBones to each other. Interestingly enough we didn't foresee the issue we were about to cause ourselves by "inheriting" the Ethernet cables that were initially planned to connect to the internet. By continuing with Ethernet cables we unknowingly added an extra layer of complexity to our code as opposed to I2C which would've been a simpler alternative for this project.

However we continued as we did and the initial task we had was to establish a simple connection between two BeagleBones over Ethernet cable. That proved to be more difficult than we expected as it led us down the road of UNIX sockets which we had to read upon from Beej's manual [2]. Initially we opted for C as our preferred language as it had nice and clean implementation principles for UNIX sockets. After a few days of working on sockets and establishing an initial connection between two machines we started to look into implementing our UI framework with sockets. To our surprise it turned out that implementing our C code in QT would be much more complicated than initially expected, probably due to our inexperience with QT itself. After deliberation we decided to discard our C implementation and go for full-on implementation in QT. Since QT is quite an extensive software framework it encompasses many different aspects besides UI/UX itself. It comes equipped with a TCP stack to support networking as needed and integrate it within applications. We started to write the first demo of networking on QT by implementing a simple application in which after the connection was established between the server and the client, the client would request data from the server by pushing the button. Once the button was pushed server would send the package to client and client would render the text message on its end. After we delivered that simple proof of concept our focus was turned on developing a more advanced version of it and combining it with our gameplay dynamics. We proceeded to implement our gameplay with the networking stack of QT after which most of the issues pertained to implementing the gameplay logic accordingly. Few issues were raised regarding proper serialization over the line, but those were successfully tackled as we uncovered bugs which caused the issues.

Game Development:

After establishing the networking framework, we focused on developing the core mechanics of our game. Since our primary concern was ensuring robust communication between the Beaglebones, we kept the game mechanics simple. Although we had several ideas for the game, including multiple characters, distinct offensive powers, and different attack animations, coordinating these ideas became an untenable challenge in the given timeframe. Hence, we settled on a simple game with a single idle animation for each character and four different actions: light attack, strong attack, block, and potion. These actions had specific effects on the game and were transmitted between the Beaglebones through Tcp communication.

Action	Effect
Light Attack	Do 8-12 damage, with a 10% chance to miss
Strong Attack	Do 15-25 damage, with a 25% chance to miss
Block	Reduce the opponent's incoming attack damage by 25%
Potion	Replenish own health by 25%

Figure 2: The actions available to both players in the game, and their effects.

With the core game mechanics and proof-of-concept for Tcp communication established, we started thinking about the best approach to develop our game using Qt. Initially, we planned to implement a peer-to-peer model [3] whereby the game would act as both server and client dynamically. However, this was difficult to implement in Qt in the absence of a web-server. After considering various options, we decided to use a server-client model [3] for our game. In this model, one instance of the game would act as the server, and another instance would act as the client. The Embedmon Server would handle the primary game logic, verifying action inputs for both players, executing the round, and updating the user interface with the appropriate health value on the player health bars. The Embedmon Client would send the action performed by player 2 through the datastream to the server and receive updates to its user interface. After reviewing Qt's documentation, we confirmed that this model would be the best choice for our application.

We then developed two distinct Qt Applications - EmbedmonServer and EmbedmonClient. The first application, EmbedmonServer, contains a header file (see /EmbedmonServer/embedmonserver.h) with the character attributes such as health, potion count, and available attacks. It also includes an enumerated class for all possible actions, which would be useful later as input actions from the Client would come in through the QtDatastream as numbers 1 through 4 (where 1 corresponds to Light Attack, 2 corresponds to Strong Attack, and so on). EmbedmonServer (see /EmbedmonServer/embedmonserver.cpp) initiates a new QtTcpServer [4], with a host address that can take any IP, and a fixed port '9999' through which both the Server and Client communicate. The server has several functions, including checkBothPlayersReady, which verifies that the server has received both Player 1's action through touch screen input and Player 2's action through the datastream, and executeRound, which handles battle calculations, applies damage, updates the healthbars, and checks if the game is over (one player's health has dropped to 0) which then resets the game state. The relationship between the two instances of the game on their respective Beaglebones is expanded in figure 2. EmbemonClient (see /EmbedmonClient2/embedmonclient.cpp) inherited much of the architecture of EmbedmonServer, except we declared it a QtTcpSocket [4] which attempts to connect to a fixed IPv4 address (192.168.1.2) through port 9999, handles none of the game execution, and has slight modifications to the user interface, as seen in figure 3.

Figure 3: Flowchart of the relationship between EmbedmonServer and EmbedmonClient

We tested our game in the Qt environment by establishing a localHost QtTcpServer in EmbedmonServer and having EmbedmonClient connect to it. This streamlined our workflow and ensured that the game would perform as intended once flashed into their respective MicroSD cards. Once we deemed the game stable enough, we flashed it using the process adapted from Lab 4, with one key modification for each. To simplify the process, we statically assigned IPv4 addresses for each Beaglebone, which meant modifying the contents of the 'instances' file in /rootfs/root/etc/networking and setting the address to 192.168.1.2 for the server and 192.168.1.3 for the client.

After properly flashing the game and establishing the networking properties, we used the serial monitor to run EmbedmonServer and EmbedmonClient on their respective Beaglebones, as seen in figure 4. The game performed as expected from our simulations and proved to be quite enjoyable when played against an opponent.

Embedmon Server *Embedmon Client*

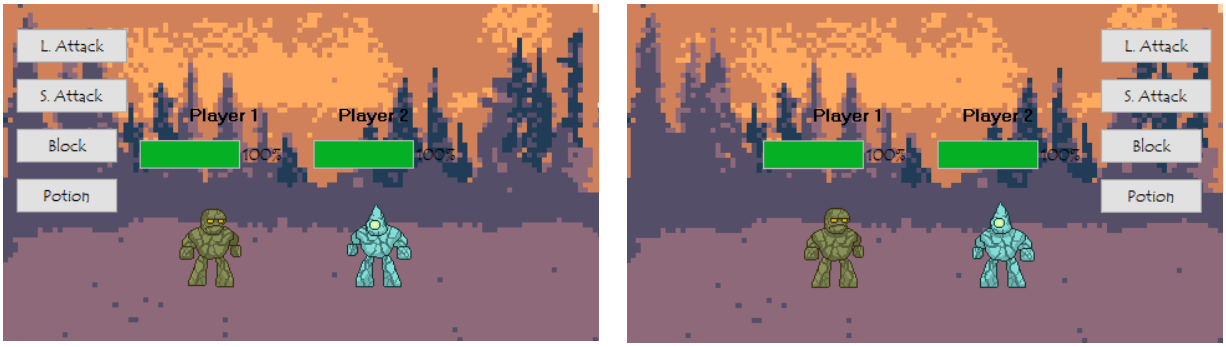

Figure 4: The Embedmon Server instance (left), and Embedmon Client instance (right), as they appear on their respective Beaglebone Black displays.

4. Summary

Our ultimate goal was to create a simple and fun multiplayer turn-based fighting game that could run on two distinct devices, and with Embedmon, we achieved just that. We were able to develop simple mechanics inspired by popular turn-based fighting games like Pokemon, create an intuitive user interface with good-looking graphics, and, most importantly, develop a robust communication framework between the two players. We went beyond the minimum viable product for our application by incorporating these elements into our game.

However, there were a few things we could not achieve due to time constraints. In the game files, there were unused assets for player attack animations that we would have liked to implement to enhance the game's visual appeal. We also had ideas for character selection and character powers that could not be implemented at this point in time. Although we faced a learning curve when using a new IDE such as Qt, we were able to deliver a polished and functional product, gain invaluable experience in designing networked applications for embedded systems, and develop a greater appreciation for the hard work that goes into creating such products.

References

[1] Integra Sources. "Qt/C++ Embedded Development: Pros, Cons, Alternatives." [Online]. Available: https://www.integrasources.com/blog/qt-c-embedded-development-pros-cons-alternatives/. [Accessed: May 2, 2023].

[2] B. Hall. "Beej's Guide to Network Programming." [Online]. Available: https://beej.us/guide/bgnet/. [Accessed: May 2, 2023].

[3] M. Hatori. "Peer-to-Peer vs. Client-Server Architecture." [Online]. Available: https://blog.hathora.dev/peer-to-peer-vs-client-server-architecture/. [Accessed: May 2, 2023].

[4] "Qt Network Programming." [Online]. Available: https://doc.qt.io/qt-6/qtnetwork-programming.html. [Accessed: May 2, 2023].