Software Overview

Year: 2021 Semester: Fall Team: 8

Project: Sink or be Sunk

Creation Date: September 4, 2021 Last Modified: September 10, 2021

Author: Garrett Brillhart Email: gbrillha@purdue.edu

Assignment Evaluation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Score (0-5)** | **Weight** | **Points** | **Notes** |
| **Assignment-Specific Items** | | | | |
| **Software Overview** |  | x2 |  |  |
| **Description of Algorithms** |  | x2 |  |  |
| **Description of Data Structures** |  | x2 |  |  |
| **Program Flowcharts** |  | x3 |  |  |
| **State Machine Diagrams** |  | x3 |  |  |
| **Writing-Specific Items** | | | | |
| **Spelling and Grammar** |  | x2 |  |  |
| **Formatting and Citations** |  | x1 |  |  |
| **Figures and Graphs** |  | x2 |  |  |
| **Technical Writing Style** |  | x3 |  |  |
| **Total Score** |  | | |  |

5: Excellent 4: Good 3: Acceptable 2: Poor 1: Very Poor 0: Not attempted

General Comments:

1.0 Software Overview

The microcontroller has software and firmware responsibilities to manage gameplay and user I/O. To remove ambiguity, firmware refers to code that interacts with physical hardware in the real world while software refers to higher level processes that control the firmware. Via software, the microcontroller must handle information from the server, manage game requests, and maintain the board’s own information (boat positions, attacks/hits, etc.). The microcontroller is responsible for initializing the websocket to the server in order to connect the device to WiFi. The software is also responsible for deciphering user inputs from the firmware to put in the appropriate requests and communications to the game server. These communications include requests to play, the attack of that players turn, and boat positions. The software must also be able to understand the data from the defense board and transform it into actual positions of the boats. The microcontroller software is also responsible for interpreting and understanding communications from the game server and implementing the correct response. This includes telling the firmware to update LEDs, play sound files, and when to look for certain user I/O. The *Microcontroller Gameplay Algorithm* in Appendix 1 details the process of the microcontroller software activities. More specifically the *Microcontroller State Transition when In-Game* in Appendix 2 details the transitions the software takes when actively in the game.

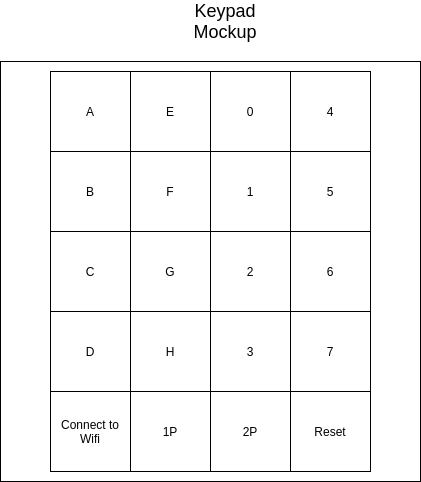
For firmware, the microcontroller will handle user I/O as directed by the software aspect. The firmware of the microcontroller is responsible for scanning the keypad via asserting rows and listening to the columns to determine user inputs. Firmware also includes detecting the placement of a boat and letting the software know of the voltage and necessary data so the software can determine orientation and type of boat. The microcontroller firmware will also automatically update the addressable LED strings as software indicates changes in the string from hits/misses/attacks.

The game server is purely software. The functionality of this software is to determine the results of attacks, manage turns and start/ends of games, play as an AI for single-player, linking two boards for a match, and to provide as a communication mediator and simplifier between the two boards. The server maintains a lobby and is answerable for matchmaking and game creation. The server’s main responsibility is to manage overall gameflow as it acts as a controller between each board. The server communicates to boards when to attack, when to wait, and when the game ends; essentially directing each game board what to do and when to do it. A responsibility of the server is to determine the results of the attacks via managing the state of both game boards ship placement and determining if the attack hits anything. The server is then responsible for communicating the result to each game board. The software will also be responsible for playing as an AI, in the event of a single player game. The *Server Gameflow Algorithm* in Appendix 1 details the process of the server.

2.0 Description of Algorithms

Determining boat position is determined through hardware specific characteristics and an algorithm to determine information about the placement. Through analog multiplexed inputs, the microcontroller will iterate through 64 ports on the defense board. Once a voltage is detected, the position within the rows and columns will be noted and look for the other end of the boat via another voltage detection. Upon finding the placement of two ends, the microcontroller will check the ADC measurement of the voltage divider from the boat's placement and use a look up table to identify what boat was plugged in. The microcontroller will remove those coordinates from the iterative search of 64 pegs and continue searching until all 5 boats have been placed. The microcontroller will then communicate to the webserver the positions of the ends of the boat and type for all 5 of the user’s ships.

The initialization for the websocket uses multiple communication protocols and algorithms in its process. Once powered on, the microcontroller uses BLE in ESP32-specific provisioning packets. Using a phone with an ESP32 provisioning application, the microcontroller connects to the phone using a peer-to-peer connection. The application then prompts the user to provision the device with a default password that is autofilled. Once provisioned, the application scans for WiFi networks within range of the ESP32 and allows the user to provide authentication for a specified WiFi host. The ESP32, now armed with authentication data, breaks the peer-to-peer connection and establishes a websocket via WiFi directly to the server. Much of this process is built into the ESP32 and supporting networks.



**Fig 2.1 Keypad Mockup**

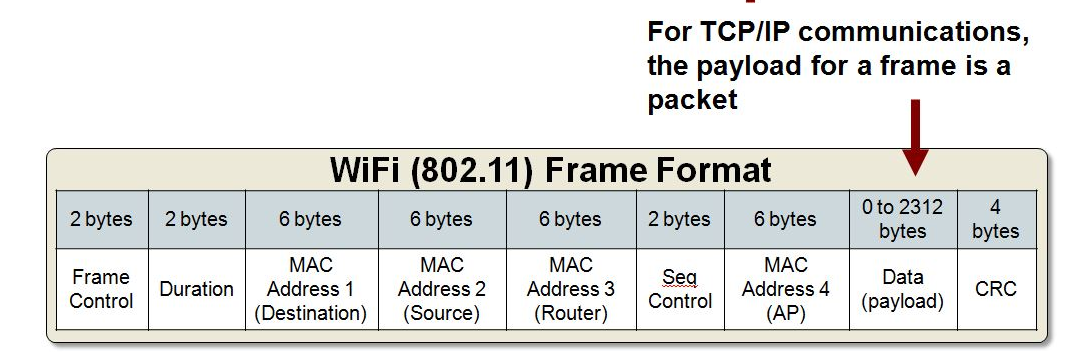
The scanning keypad algorithm relies on the type of input desired and hardware characteristics to reduce the amount of user error and debouncing. When waiting for game feature decisions on the keypad (Connect to WiFi, 2-player, 1-player, reset) the microcontroller will assert only the bottom row/keys (only relevant inputs) and listen into each column to determine the correct button press. Since the design does not need to know anything else and does not care about multiple button pushes, there is no need to debounce as the microcontroller will not need additional input immediately after the initial push. When scanning for attack coordinates, the first four rows (letter and number rows) will be asserted one-by-one and only the first two columns (letter columns) will be listened to at first. Once an input is detected, the microcontroller will decipher the push and then instead only listen to the last two columns (number columns). Therefore, only valid coordinates can be input by disregarding all irrelevant keypad pushes before the data even makes it into the software. Implementation may not be a strict keypad and buttons and could be separated around the LED rows and columns, but the algorithm still generally applies to scanning a “row” of letters and a “row” of numbers. Only relevant data is ever considered, and irrelevant keypad presses are ignored through deliberate inactivity of the keypad.

Updating LEDs is straightforward with the individually addressable LEDs and server information. The packet from the server will contain which board was attacked, the coordinate, and the result. The microcontroller algorithm will parse through the char\* to get the necessary data. The microcontroller determines whether the attack or defense board needs updating based on the packet information specifically directing which board was attacked. Via the coordinate, the letter will determine which row of LEDs needs updating and the number specifies the LED within that row. The result of attack indicates to the microcontroller what color that LED needs to turn, done via writing in the RGB value to the LED. The microcontroller simply addresses the LED to change, and writes in the raw RGB values through the built-in LED strip interface (similar to a shift register).

The AI/Computer battleship strategy will be relatively simple, relying mostly on RNG to pick targets. The AI will have 3 modes of attacking, denoted as searching, identifying, and destroying. The AI begins in searching mode, using a RNG to choose an un-attacked square to attack for each turn. Once the AI in search mode gets a hit, it then transfers into identifying mode. Identifying mode attempts to find the orientation of the boat, by attacking counterclockwise around the last hit until the AI gets another hit. After the next hit, the AI transitions into destroying mode. This mode follows the line of the two hits until the AI results in a miss, has hit 5 in a row, or knows that any boat “higher” than the one currently being attacked has already been destroyed (ex. If the 5-hit boat has already been destroyed, no need to attack more after 4 hits). After, the AI will attack the square on the other side of the line, if identifying mode has not already attacked it. This is done to ensure that the enemy’s boat is annihilated, by lining each end with a miss. After destroying mode is finished, the AI transitions back into searching mode. See the *Server AI Attack Pattern* in Appendix 2 for a clearer view of the transition between modes of attacking.

3.0 Description of Data Structures

The Wifi packet communication structure will follow the WLAN 802.11 standard as shown below:



**Fig 3.1 WiFi Frame Format**

The preamble information will vary from connection to connection, as this information is determined during the setup of the websocket. For the data payload, the server and ESP32 will communicate through JSON utilizing 3 data fields. Request field (“req”) for the type of request, ID for board and user identification, and an optional DATA field dependent upon the type of request. The req field is composed of NEW\_GAME, MAKE\_MOVE, JOIN\_GAME, GAME\_TYPE and POSITION\_SHIPS. The data field of char\* is built according to the following structures:

|  |  |
| --- | --- |
| **Table 3.1 - ESP32 To Server Payload Format** | |
| Description | Format |
| Request to play, 1-player | RTP1P |
| Request to play, 2-player | RTP2P |
| Boat placement of all 5 boats, where W is the row letter and X is the column number of one of the ends and Y and Z is the other end. The number in the front describes the amount of pegs the ship covers | BP;5WXYZ;4WXYZ;3WXYZ;3WXYZ;2WXYZ |
| Attack Turn where XY is the row column coordinates in letter number format, respectively | ATXY |

|  |  |
| --- | --- |
| **Table 3.2 - Server to ESP32 Payload Format** | |
| Description | Format |
| Match Ready, 1 Player | MR1P |
| Match Ready, 2 Player | MR2P |
| Game Starting, Player’s Turn | GSPT |
| Game Starting, Other Player’s Turn | GSNT |
| On Attack board, miss at grid square XY, not end of game | AAMXYN |
| On Attack board, hit at grid square XY, not end of game | AAHXYN |
| On Attack Board, hit at grid square, end of game | AAHXYY |
| On Defense board, miss at grid square XY, not end of game | ADMXYN |
| On Defense board, hit at grid square XY, not end of game | ADHXYN |
| On Defense Board, hit at grid square, end of game | ADHXYY |
| End of Game, win by default for a variety of reasons (disconnection between boards) | EOGW |

Also note that standard WiFi packets (ACK, NACK, etc.) are utilized for communication between server and board to ensure communication is valid and successful between them. The server utilizes a JSON deconstructor to parse the JSON fields and the microcontroller will use a reverse parser in order to utilize the information from the server.

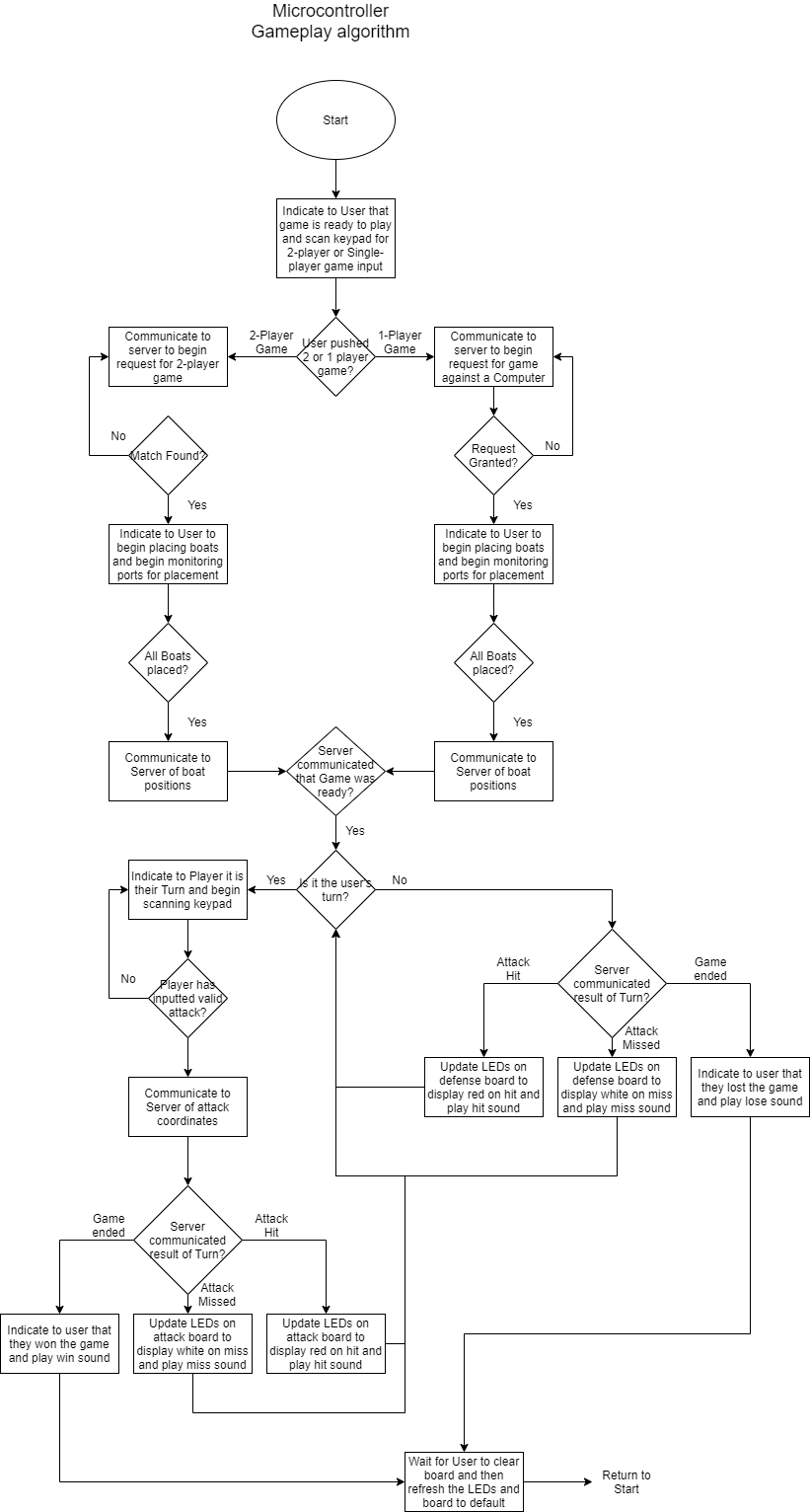
There will be two gameboard data structures on the server made up of a created class/structure. The structure will include relevant user information, including profile name, IP at connection, and other necessary data to identify and find the user. The structure will also include a boolean variable that identifies if it is the user’s turn currently. The main part of the data structure will be a 64-byte long char\* containing the relevant information for that player’s defense board. The board will be used to determine the results of attacks and boats. The data structure will be filled according to the table below:

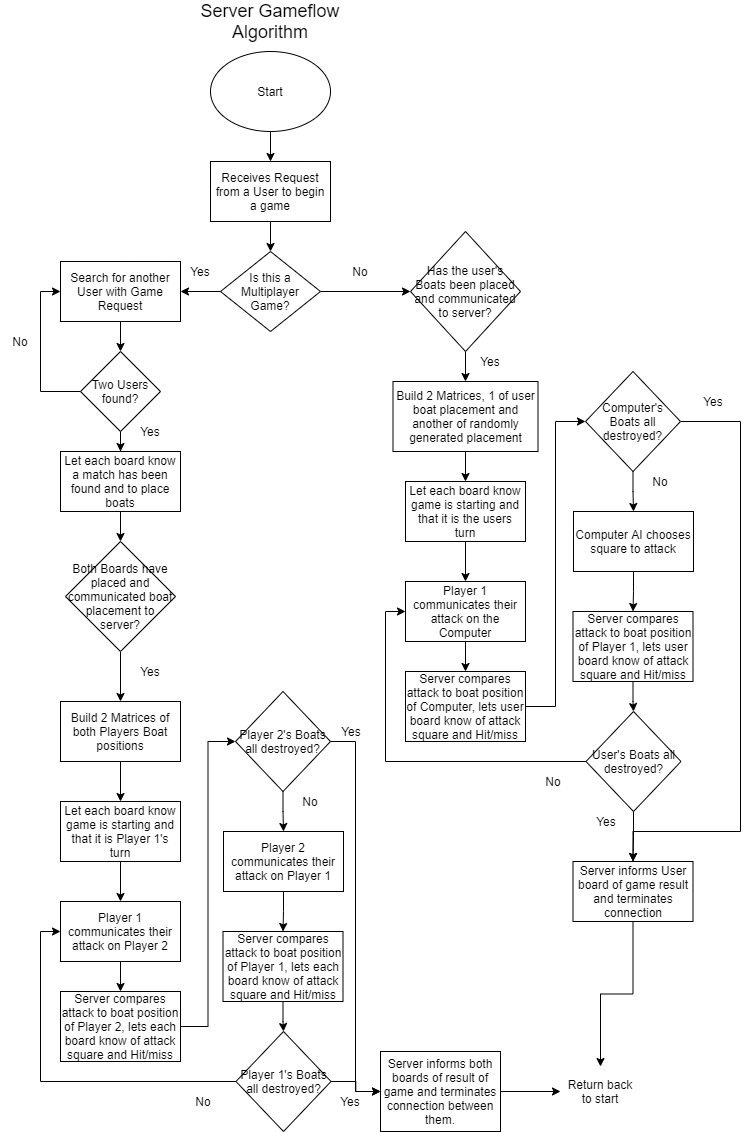
|  |  |  |
| --- | --- | --- |
| **Table 3.3 - Char\* Data Structure Identification** | | |
| Type | Number | Description |
| Empty | 0 | Square with no attacks or boats |
| Miss | 1 | Square that has been attacked but with no boat piece on it, a miss |
| Boat-Functional | 2 | Square with a boat piece on it, but with no attack on that square |
| Boat-Destroyed | 3 | Square with a boat piece on it and has been attacked on that square |

4.0 Sources Cited:

“Wifi packet structure,” *makerlasopa*. [Online]. Available: <https://makerlasopa215.weebly.com/wifi-packet-structure.html>. [Accessed: 05-Sep-2021].

Appendix 1: Program Flowcharts





Appendix 2: State Machine Diagrams

