Team Members

- Khai Phan (U25285116)
- Jerome Andaya (U41337769)

EC450 Final Project Report: Pokémon

Goal

The goal of our project is to develop and recreate a Pokémon battle game, inspired by the nominal video game series. The game is centered around Pokémon's battle feature. Our version of the game entails a turn-based 1-on-1 battle, in which the players select moves to use against their opponent's Pokémon until either Pokémon's health bar is depleted. Although the original Pokémon game contains hundreds of Pokémon, our game includes a selection of our ten favorite Pokémon. We also incorporated the traditional and iconic type advantage feature into our game by implementing inherent strengths and weaknesses for each Pokémon. Thus, the players will find themselves at either an advantage or disadvantage based on their Pokémon selection.

Design

The requirements for the goals of our game can thus be summarized into the following list.
The game must include:

- A way to display our Pokémon (in the form of text and images).
- A means of player input/output for the player to make decisions.
- A private Pokémon selection screen to prevent knowledge of potential type advantages.
- A method to determine type advantages and move effectiveness.
- A battle phase, in which the players can select moves and transmit that information to the other player.
- Lastly a means of determining the winner of a battle.

To fulfill these requirements, we designed the game for the MSP432P401R microcontroller, using the EDU-MKII Booster Pack as a combination player controller and display device. We use the Booster Pack's LCD screen to handle all visual output. To handle player inputs, we use the Booster Pack's joystick and buttons. Additionally, we used the SPI communication protocol to transmit data between the two MSP432 microcontrollers.

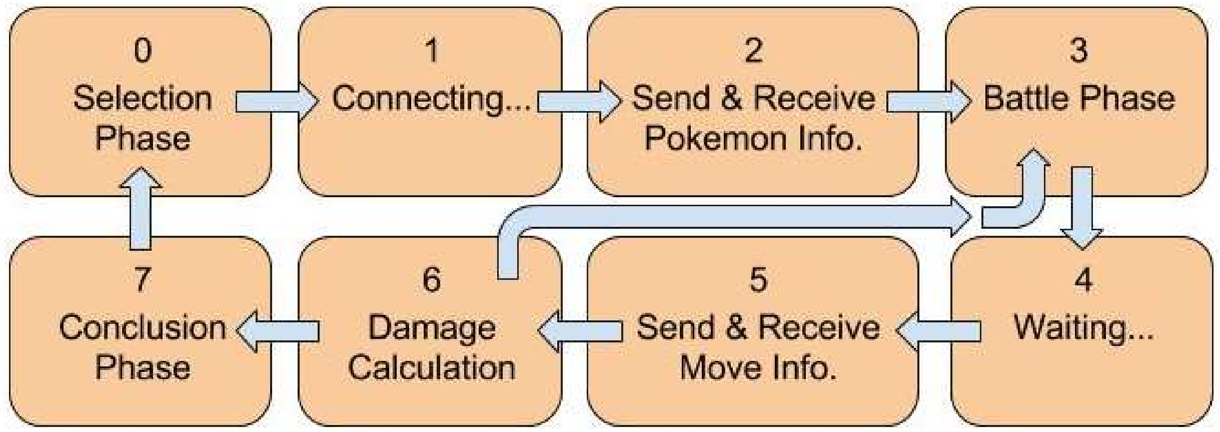
To display Pokémon, we chose to display both images and text to offer a graphically appealing interface, recreating the actual Pokémon battle menu to the best of our ability. To create these graphics and text on the screen, we used existing images, libraries, and programs. This design choice indubitably saved time while enhancing visual appeal. We would not have been able to create our own high-quality images and text given the time to completion. Thus, the process of displaying graphics on the LCD screen is as follows. A pre-existing image was converted into a BITMAP array in C using the MSP430 TI program *Image Reformer Tool*. This BITMAP array was then passed through a function featured in TI's graphics library to draw the image onto the LCD display. Additionally, we used a similar function featured in the same graphics library to draw text onto the LCD display.

Having the base framework for displaying output and receiving user input, as well as a means of communication between two devices, the remaining design requirements to be fulfilled are the game contents. Overall, the design of the game contents can be divided into three phases: the *selection* phase, the *battle* phase, and the *conclusion* phase.

During the *selection* phase, both players select the Pokémon they wish to battle with. To make this process private, only the respective player's Pokémon is displayed on their respective LCD displays. During the *battle* phase, both Pokémon are displayed on the LCD screen, along with the respective Pokémon's abilities. To create a system of strengths and weaknesses, each Pokémon has up to two types, and each ability/move has one type. When a Pokémon uses an ability on the enemy Pokémon, the damage is modified depending on the effectiveness. For example, if Pokémon A uses a grass move on Pokémon B whose type is fire, the ability's damage is halved due to grass' weakness against fire. However, if Pokémon A uses a fire move on Pokémon B whose type is grass, the ability's damage is doubled. The damage dealt to each Pokémon is subtracted from the Pokémon's health. The health bars are adjusted and redisplayed based on the new values. This process of selecting abilities and calculating new health is repeated until a health bar reaches zero. If either health bar (or both) reach zero during the calculation, the game is moved into the *conclusion* phase. During the *conclusion* phase, a message is displayed depending on which Pokémon's health reaches zero. If both health bars reach zero at the same time, then the battle ends in a draw. The *conclusion* phase re-enters the *selection* phase when the user presses the button.

Implementation

To implement our design, we create a state machine for our game. As such, each state is described separately. Additionally, the respective implementations of the joystick and buttons are also described separately as they play an important role in every state of the game. The database for Pokémon data is also described since it plays an important role in all states of the game. For reference, the following diagram is a visual representation of the game's state machine.



A. Joystick

The joystick is implemented by using an analog to digital converter. The conversion is performed at a constant rate for both the X and Y axes of the joystick. Whenever both axes are calculated, a flag is raised to send an interrupt; this interrupt is handled by the ADC14 handler featured in the microcontroller's libraries and design. The X and Y values are then interpreted as LEFT, RIGHT, UP, or DOWN inputs from the user. These inputs are used throughout various states of the game's state machine. It is also important to note that because the ADC14 handler is called regularly, this handler is used as the "clock tick" of our game. Each time this handler is called, our game proceeds by a step/frame.

B. Button

The button is implemented by using the ADC14 handler rather than using an interrupt. Since the ADC14 interrupts at regular intervals, the 5.1 pin (the button) is checked during each ADC14 interrupt. If the last state of the button is LOW and the last state of the button (the state of the button during the previous ADC14 interrupt) is HIGH, the button is pressed. This button press performs a different action depending on which state the game is currently in. For example, if the game is in the *conclusion* phase, the game progresses into the *selection* phase if the button is pressed.

C. Pokémon Data

All Pokémon data such as health points, abilities, and type are stored in multi-dimensional string arrays. The Pokémon images/graphics are stored in BITMAP C arrays generated by the MSP430 *Image Reformer Tool*.

D. State Machine

State '0': *Selection Phase*

During this state, the Pokémon banner is drawn once using the graphics library *Graphics_drawImage* function. The reason for only drawing the banner once is because the drawing process is computationally heavy. Thus, if the same image is drawn per frame of the LCD display, the display is noticeably lagged. Additionally, during this state, the Pokémon image, name, and type(s), along with the interface are drawn using the *Graphics_drawString* and *Graphics_drawRectangle* functions. The arrows are drawn by with lines of different lengths (since there is no function to draw a triangle) using *Graphics_drawLineH*. The joystick is checked for RIGHT or LEFT inputs to switch between Pokémon. If the user presses the button, the state changes to 1 and a string "Connecting…" is drawn onto the LCD screen. The selected Pokémon is also locked and saved via an unsigned 8-bit integer variable.

During state 0, the microcontrollers transmit 0x00 and do not expect any data from the other microcontroller.

State '1': *Connecting...*

During this state, no action is performed on the LCD and the user inputs are not accepted. Rather, the microcontrollers are transmitting 0x11 and are expecting to receive 0x11 to progress to the next state.

State '2': *Send & Receive Pokémon Information*

During this state, both devices continue to send 5 more packets of 0x11. This is implemented to prevent the error caused by one device moving ahead first. Occasionally, one device progresses to the next state and no longer transmits 0x11. The other device is then stuck in the previous state waiting for the 0x11 signal to move forward. This implementation prevents this error from occurring.

After the first 5 packets, the devices then transmit packets representing the respective Pokémon selections during *selection* phase. The same packet is transmitted 8 times, but only the fifth packet is accepted. The implementation is performed this way to lower the risk of falsely receiving 0x00 and 0x11 packets as Pokémon data. The state then progresses to state 3.

State '3': *Battle Phase*

During this phase, the LCD screen is cleared and both Pokémon are drawn on both devices using the packets received during the *Send & Receive Pokémon Information* state. The Pokémon's names and abilities are drawn by using *Graphics_drawString*, *Graphics_drawImage*, and *Graphics_drawRectangle*. The Pokémon's health points are loaded from the stored data and their health bars are initiated to full (green) by using *Graphics_fillRectangle*. The joystick is checked for LEFT, RIGHT, UP, and DOWN inputs to display which Pokémon ability the user is hovering over. If the user presses the button, the state changes to 4 and the ability number is saved via an unsigned 8-bit integer variable.

During state 3, the devices transmit 0x00 and do not expect any data from the other microcontroller.

State '4': *Waiting…*

During this state, no action is performed on the LCD and the user inputs are not accepted. Rather, the microcontrollers are transmitting 0x11 and are expecting to receive 0x11 to progress to the next state.

State '5': *Send & Receive Ability/Move Information*

During this state, both devices continue to send 5 more packets 0x11. This is implemented to prevent the error mentioned in state 2. After the first 5 packets, the devices then transmit packets representing the respective Pokémon abilities selection (saved in the unsigned 8-bit integer variable during state 3). The same packet is transmitted 8 times, but only the fifth packet is accepted. This implementation is performed this way to lower the risk of falsely receiving 0x00 and 0x11 packets as the Pokémon ability data. The state then progresses to state 6.

State '6': *Damage Calculation*

During this state, both devices perform the damage calculations using the *calcDamage* function. Note that both devices will calculate the damage for both the respective device and the opposing device. This function calculates the damage based on the effectiveness and power of the move. The arguments for this function are parsed from the stored Pokémon data. The moves used and effectiveness of the moves are drawn onto the screen by using *Graphics_drawString* and *Graphics_fillRectangle* functions. For example, if the enemy used a fire move on your grass Pokémon, a string "Super Effective!" is drawn. Instead, if the enemy used a grass move on your fire Pokémon, a string "Not effective…" is drawn. The updated health points are then calculated for both Pokémon and a red rectangle is drawn on top of the green health rectangle to reflect the damage taken by both Pokémon. If either Pokémon's health points reach zero during this state, the state is progressed to 7. Otherwise, the state is changed to 3 to continue the *battle* phase.

During this state, both devices transmit 0x00 and do not expect any data from the other microcontroller.

State '7': *Conclusion Phase*

During this state, both devices clear the LCD display. A string is then drawn using *Graphics_drawString* to present the winner, based on which Pokémon's health points reaches zero in the previous state. The three possible strings are: "You Win!!", "You Lose!", or "You Draw!". If the user presses the button, the state is changed to 0 to return to the *selection* phase of the game.

During this state, both devices transmit 0x00 and do not expect any data from the other microcontroller.

Success Assessment of Project

Reviewing the original project goals, the project is a success. The game includes properly display of Pokémon in the form of both text and images. There are functioning means for the user to interact with the game: the joystick, LCD screen, and button. We successfully incorporated the "rock-paper-scissors" element into the Pokémon selection both from privatizing the *selection* phase and implementing a model for determining strengths and weaknesses. Lastly, the *battle* and *conclusion* phases function properly and correctly declare the winner of the Pokémon battle. It is also important to note that the SPI communications are functioning properly, allowing for the games to properly operate together.

Potential Next Steps

Some potential areas of improvement are:

- Fixing the starting SPI connection issue. The SPI connection is occasionally out of phase, causing all bytes transferred to be "corrupted". To fix this, we can add a fourth wire to prevent the master from sending until the slave is ready to receive.
- Adding LEDs. Adding LEDs would increase the aesthetic appeal.
- Adding music and sound effects. Adding music and sound effects would remove the quiet environment while playing to game and make the game more engaging.

Summary of Team Member's Contributions

Jerome Andaya

- Developed framework for the game, including the basic state machine (*selection*, *battle*, *conclusion* states).
    - Framework includes the framework of the selection menu, battle screen, and data storage.
    - Framework also includes creating functioning joystick inputs and LCD display.
- Attempted to implement a fix for the bit shifting error.

Khai Phan

- Created BITMAP C arrays for Pokémon images.
- Implemented the SPI communications aspect of the game.
- Improved the state machine to allow for smooth SPI communications.
- Contributed to *selection*, *battle*, and *conclusion* states by adding images, text, and extra features.
- Improved Pokémon data storage to account for more information.

Code

The code for this project is attached in the zipped folder.