Transforming the tic-tac-toe game

Gabriel Seger

Video games and digital tools are more frequently being used for educational purposes. In this article, we are transforming the traditional tic-tac-toe game into a dynamic, motion-controlled digital experience.

Intro

Our project explores the synergy between physical computer inputs and video games. Highly physical video games may offer a playful mechanism for developing one's coordination, perhaps making them of future relevance in grade schools and occupational therapy offices. Plus, they're fun. With its simplicity, our game could become an inclusive tool that is accessible, educational, and entertaining to learners of different ages and with diverse abilities.

As a proof-of-concept example of a video game with a physical interface, we designed and built a tic-tac-toe game that one plays via physical motion tracking of a game controller. Two RP2040 microcontrollers control the whole system. One microcontroller, equipped with an MPU6050 sensor and a Bluetooth module, functions as a dynamic motion controller. The other, serving as the main unit, manages a VGA display and the game logic. We focused on creating a seamless and intuitive gameplay experience, utilizing the motion data from the MPU6050 sensor for cursor movement and leveraging Bluetooth for data transfer. An important limitation to note is that we only had four weeks to complete this class project, some decisions on our design would have differed if we had more time.

High-Level Design

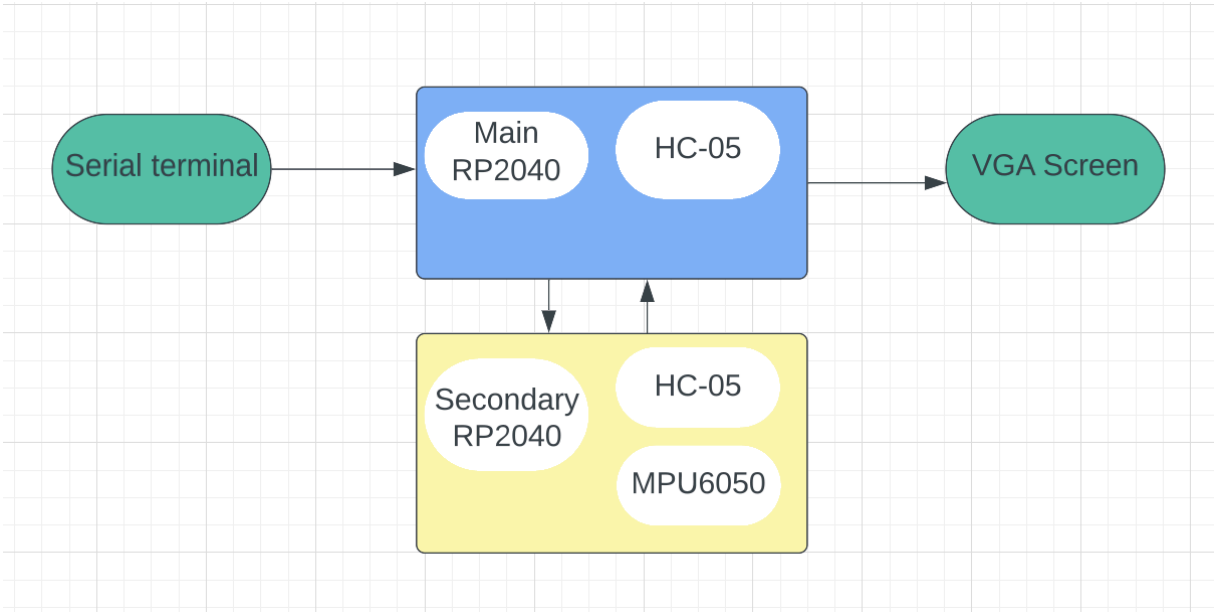

Figure 1: High Level Block Diagram

Figure 1 presents an overview of our project. At the heart of our hardware configuration were the two RP2040 microcontrollers. The main RP2040 microcontroller interfaces with peripheral devices through various GPIO pins. It communicates with the VGA screen to manage the game display, with the HC-05 Bluetooth module for wireless communication with the game controller, and it handles all game logic. The second block consists of another Pico RP2040 that is connected to the secondary HC-05 Bluetooth module. It's also connected and communicates with an MPU6050 sensor over I2C. Each part and its functionality will be covered in more detail later.

The game starts when a player sets a reference point, by pointing the game controller toward the screen and confirming the action on the serial terminal. This step calibrates the system for movement calculations. With the system calibrated, the player then maneuvers the cursor by aiming directly at the VGA screen and adjusting the position of the controller. To start the game, the player selects the "start game" option from the menu and then continues by making the game's first move. After the player has made their move, the game will toggle players and the other player takes the controller and places a move. Each player chooses a

box by physically moving the cursor and confirming their selection via the serial terminal. If we had more time, we would have implemented buttons on the controller that the players could use for these selections.

Hardware details

Hardware Component	**Purpose**
Two (2) RP2040 Microcontroller	One Pico for the Controller, as well as one main Pico that controls the VGA display, general game logic and managing communication protocols.
VGA Screen	Display Menu and Tic Tac Toe Board.
Two (2) Breadboards	Base for mounting and interconnecting all hardware components, including power and ground. One acts as the controller.
Putty Interface	Enables UART communication with the RP2040, allowing for user input.
MPU6050 IMU	Gather positional data for the controller.
Two (2) HC-05 Bluetooth Adapter	Bluetooth adapter to facilitate communication between Picos.

Table 1: Hardware Components

Figure 2 shows our project schematic. The core of our setup was the Raspberry Pi Pico Development Board. We tested our circuit as we built it, to ensure each part worked for itself. Figure 3 demonstrates our physical hardware setup. Note that the main RP2040 is connected to the left breadboard and the secondary RP2040 is connected to the right breadboard, which also acts as our handheld controller.

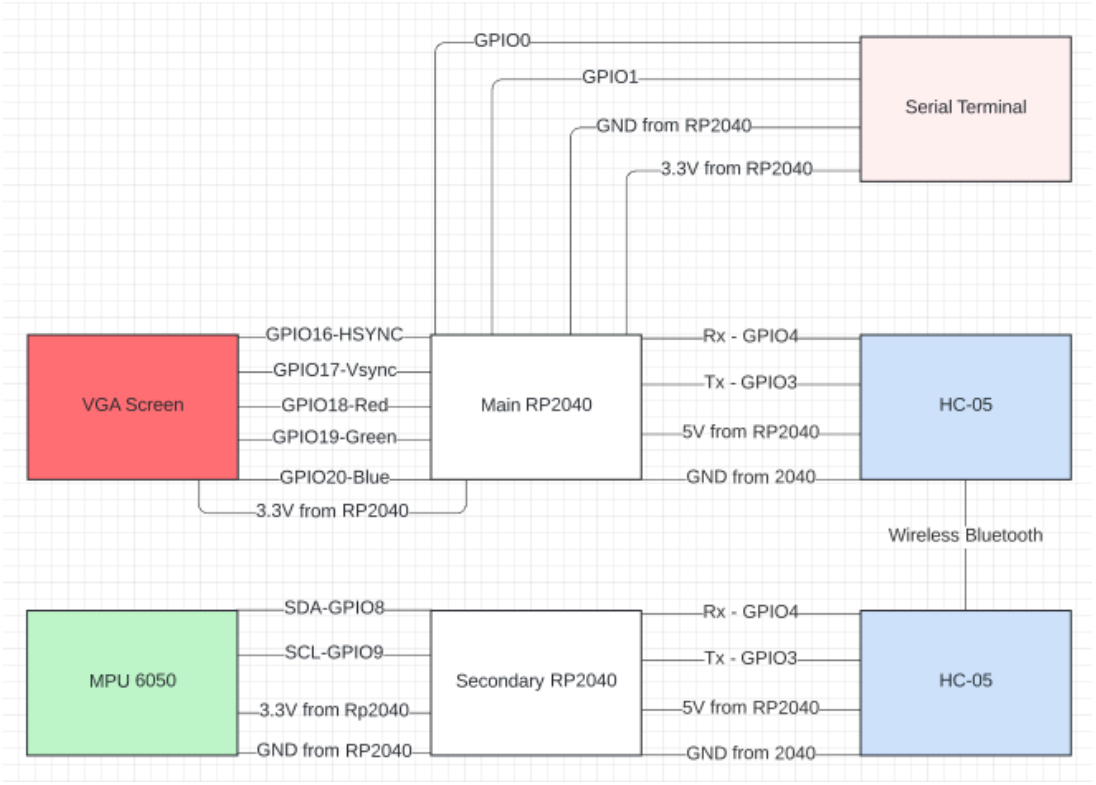

Figure 2: Hardware Schematics

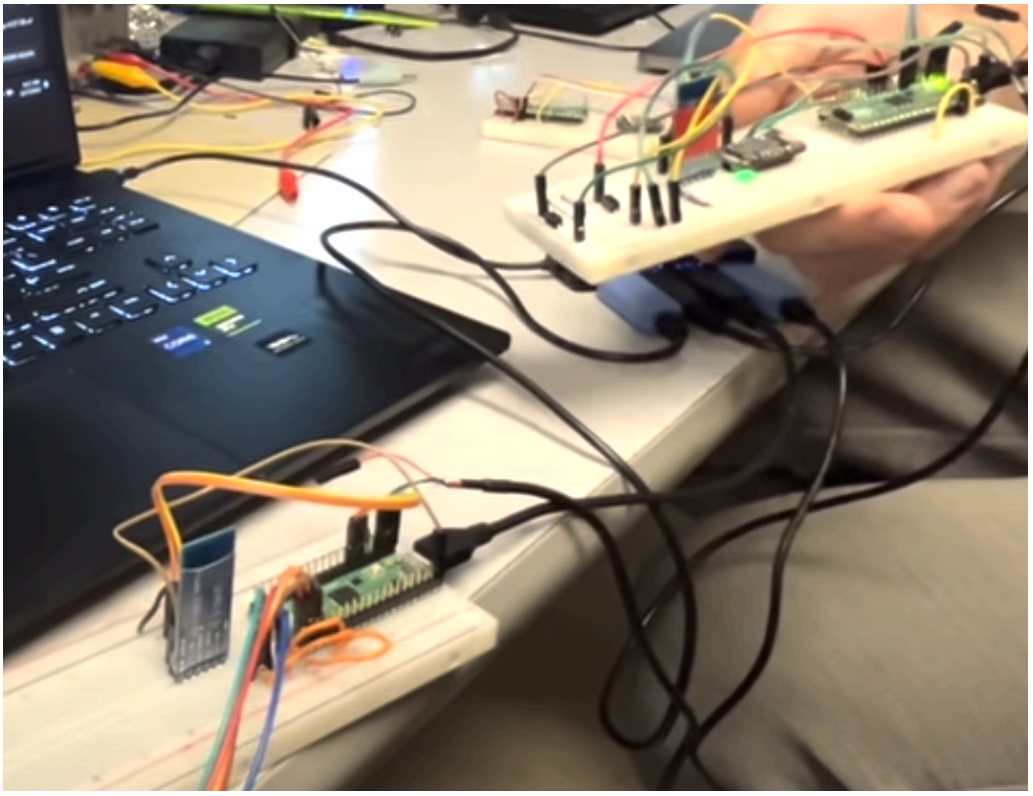

Figure 3: Hardware setup

Game logic

One could say that the game logic works as the center of our project and pieces together different parts. So, we had to be careful when designing this part. The file itself isn't that complicated, the challenge lied in making the parts work together.

The game starts with initializing a global board array, which represents the Tic Tac Toe grid. This array holds values indicating the state of each cell, which is either empty, the first player, or the second player. Then we created a function that visually constructs the game board on a VGA display by drawing vertical and horizontal lines to create a 3x3 grid.

Players use the controller to move a cursor on the VGA screen. When a player makes a move by aligning the cursor to a desired cell and confirming it in the serial terminal, the program translates the cursor's screen coordinates into the corresponding cell on the Tic Tac Toe board. Then, we implemented a function that updates the board array with the player's symbol, X or O, based on the cursor's position. It also alternates between players after each move.

We have one function that continuously updates the VGA display with the current state of the game board, including any 'X's or 'O's placed by the players, and another function that checks for winning conditions after each move. It does so by scanning rows, columns, and diagonals to see if either player has aligned their symbols in a line. If a winner is found or the board is full, the game concludes, displaying the result on the VGA screen and announcing the winner or declaring a draw. Figure 4 shows the output of the VGA screen when the game is over.

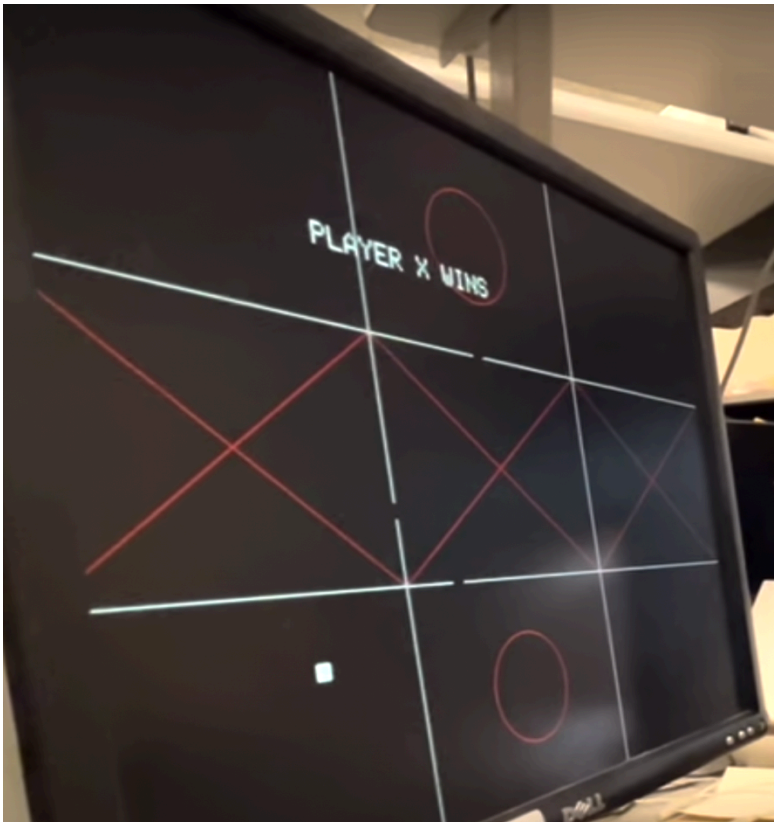

Figure 4: VGA screen when the game is over

Integration

Our main file incorporates two threads. One thread is responsible for communicating with our Bluetooth module and this thread exists on its own in case UART communication blocks the execution of the core. The menu state, menu inputs, and drawing to the VGA screen are handled by a separate thread on core 1. Since this thread handles the menu, it also calls the necessary functions to begin the tic-tac-toe game if the appropriate menu item is selected. As a result, this thread also handles all game logic, as a sort of game engine for the tic tac toe game. Once the game is completed, the function which starts the game returns. This allows the input function to return, allowing the device to return to a state with the menu displayed. At this point, the game can be replayed as before by clicking on the appropriate start game button. Figure 5 shows a general overview of how different threads interact with each other.

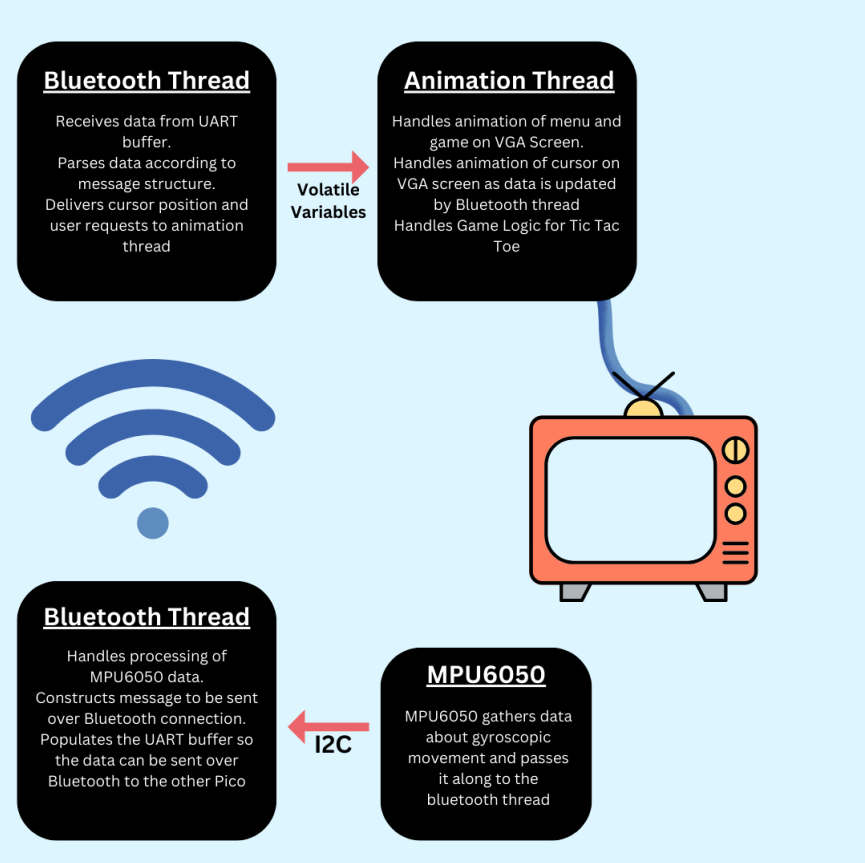

Figure 5: Software Design

Bluetooth

We wanted a wireless motion controller, so we had a few alternatives. Wi-Fi was one option but as we only had four weeks to finish this project, we decided to work with Bluetooth as it is fairly simple to implement, and it met our speed requirements for this project. A baud rate of around 1000 would have been enough for this project but we were able to use a baud rate of 9600 with Bluetooth. The Bluetooth connection would be important for the rest of the project so, our first task was to connect these modules. Our Bluetooth protocol follows a main-secondary implementation. As such, the main Pico is the Pico directly responsible for the outputs. The secondary is the controller that would like to connect to the main Pico and the one processing measurements from the IMU.

Establishing communication between our two HC-05 modules posed a few challenges. We began by connecting the secondary RP2040 to a breadboard, which would also act as our controller. After that, an HC-05 module (transmitter) was connected to this RP2040. The main RP2040 was then placed on a separate breadboard which connected to our second HC-05 module (receiver).

The Bluetooth system works by writing UART messages to a UART channel connected to our Bluetooth module. The Bluetooth module then takes any messages it receives on the UART channel and wirelessly transmits them to the connected secondary Bluetooth module, which subsequently delivers the message to the attached RP2040 via a UART channel. Finally, some basic setup was done to connect the main Bluetooth module to the secondary module on the other Pico, ensuring the two modules only speak to each other.

Two issues arose when we tried to connect the modules. The first revolved around establishing a reliable connection between the Bluetooth modules. Our initial strategy involved using a serial terminal for sending commands to one Bluetooth module through a UART connection. However, we faced one hurdle, the serial terminal added a trailing zero which disrupted communication. We resolved this by hardcoding the commands directly into the Pico.

The second challenge was constructing and deconstructing Bluetooth messages between the two Pico's. We needed a packet structure so that the secondary Pico, which is the handheld controller, could send messages that the Main Pico knew how to interpret. We decided to use one message to send all the information at once. Our implementation uses strtok to tokenize the message based on our chosen tokens, like "|". Although this worked, we continued to face issues where the correct variables were being set more than once. We solved this by adjusting the code to handle recurring parsing errors, which ensured reliable communication.

IMU Sensors

We needed sensors for our motion controller and there were many options, and we knew that our decision on which sensor to pick would be important. As we had limited resources and time, we ended up choosing the MPU6050 sensor which combines both accelerometer and gyroscope, providing fast and comprehensive motion tracking while being easy to work with. It also has a sampling rate of 1KHz, which is more than enough for our project.

The best alternative would be to use both the gyroscope and the accelerometer but with our limited time, we couldn't get the implementation of the accelerometer to work. So, we ended up solely focusing on the gyroscope component for motion tracking. The gyroscope in the MPU6050 is a MEMS (Micro-Electro-Mechanical Systems) gyroscope.

The accelerometer would probably have been better to use than the gyroscope. However, we started implementing the gyroscope and the advantage is that the gyroscope provides smoother and more stable readings for small, quick movements. For this project, this means that the cursor movements on the VGA display will be fluid and responsive to small, quick motions of the controller. This allows players to make precise selections on our screen with minimal latency or overshoot, enhancing the gameplay experience.

However, over time, gyroscopes tend to drift, leading to accumulated errors in angle estimation. This phenomenon, known as gyro drift, is significant in applications requiring precise and long-term angle measurements. The process we are using, where gyro measurements are integrated over time to estimate the angle, while it's effective in the short term, inherently accumulates errors over time. This accumulating error can be mitigated by incorporating information from the accelerometer.

An important part of this section was the fine-tuning of the scale factors for gyroscopic data. Too high of a scale factor could lead to overly sensitive cursor movements,

whereas too low could render the controller unresponsive. Striking this balance was a trial-and-error process requiring multiple iterations. This was a part of developing an intuitive control scheme that felt natural to the users.

The biggest challenge was to gather the readings from the gyroscope and convert them to pixel coordinates. The gyroscope is the centerpiece for detecting and translating physical movements into digital signals. With more time, we would have used data from the accelerometer as well, to get more precise data. The code needed to convert the gyroscope's rotational data into screen coordinates, considering the screen's dimensions and the desired sensitivity.

To achieve this, we needed to mathematically process the gyroscope's readings, which capture the controller’s rotation rate and then map it to the VGA screen accurately. We also had to scale these readings to match the screen's dimensions and responsiveness, making sure that the cursor movements reflect our physical gestures. We also employed a clamping function to limit excessive cursor jumps, enhancing user experience. These calculated movements are then accurately mapped to screen coordinates, ensuring a transition from physical to digital gameplay. See the flow diagram of our implementation in figure 6.

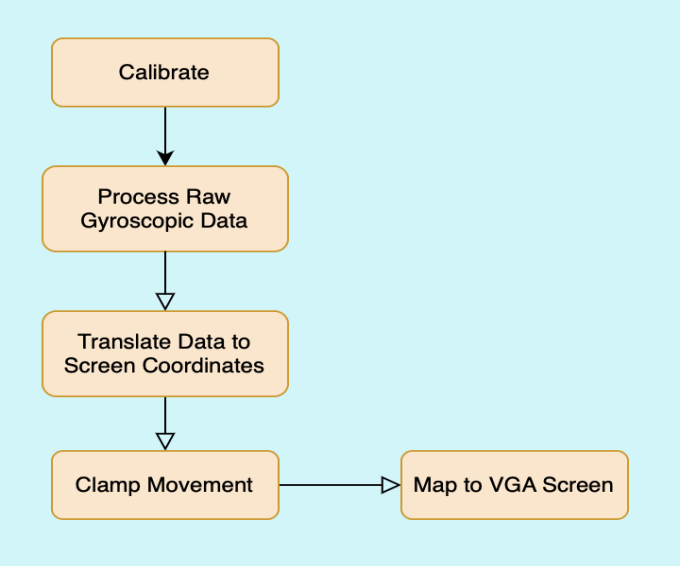

Figure 6: Flow Diagram

For our implementation, we had two important functions that processed raw gyroscopic data, scaling it to derive horizontal and vertical movement values. This represents how much the device has rotated about the x-axis, corresponding to lateral movement, and linear acceleration along the y-axis, corresponding to vertical movement.

Another important function translates the physical movements into screen coordinates, considering the screen's resolution and orientation. This part was particularly challenging, as it required an understanding of how physical movements correspond to pixel displacements on the screen.

Our clamp movement function prevents excessively large movements by setting a maximum limit. If the calculated movement exceeds this limit, it is clamped to this maximum value. This avoids sudden large jumps of the cursor on the screen and helps with the user experience.

We then use these calculations to map it to the VGA screen. The relative movements are scaled according to screen dimensions to map them to screen coordinates. This mapping ensures that the sensor's movements correspond accurately to movements on the screen. We also have a function that ensures that the resulting screen coordinates do not go off-screen.

The data gathering function checks if the device is calibrated so we know it's set to a known position which is used as a reference. If calibrated, it reads the current orientation from the MPU6050 and calculates the relative movement, which is then mapped to screen coordinates. Because of our gyro drift, we needed occasional recalibration or resetting of the controller's position to ensure continued accuracy during gameplay. Therefore, we decided to implement a calibration functionality, which allows the system to establish a reference orientation, essential for accurately tracking movements relative to an initial position.

VGA Screen and Serial Terminal

Since our project focused on displaying real-time sensor measurements on a screen, our goal was to make the software as fast as possible. We had a few options but the most suitable for us, given we only had four weeks, was to use a VGA screen, which is a cost-effective solution and which infrastructure was already in place. Additionally, driving a HDMI screen with our microcontroller would have been a challenge. The VGA screen displayed the game, but it also facilitated the project's visual feedback**,** providing an important interface for real-time monitoring and debugging.

For interactive control and for communication over Bluetooth, a serial terminal was integrated, utilizing the UART protocol for communication with the RP2040. This interface lets us handle the game, by indicating that a player confirms a placement of a character. It was also important when setting up and debugging Bluetooth communication.

Menu Design

We designed an extensible menu for when a player starts the game. The menu is contained to itself, except for any drawing elements that were imported from a VGA graphics library. The menu is composed of a bounding box in which menu items are drawn and it has a set center pointer that is decided on the creation of the menu struct. Each menu could contain at most four menu items, which were stored in an array of menu item structs. The Menu Item struct contains information about the title of the menu item and what action would be taken if the menu item was clicked. This allowed menu items to “store” functions that could be used to call other functionality. Figure 7 displays the appearance of the menu.

Figure 7: Menu

Results

A tic tac toe game was able to be played on the VGA screen between two players, each of which used the same controller by taking turns. Our controller worked as intended and the device was able to successfully move the cursor around the screen by sensing movement in the controller. We planned to use buttons for the controller but as we only had 5 weeks we ran out of time. Instead, we sent commands to the controller through a serial interface accessed through a serial terminal.

The speed of execution for the device was dependent on the speed at which the Pico could access readings from the Bluetooth module and subsequently how fast it could process the readings so they could be displayed on the VGA screen. Thus, this meant the speed of execution was somewhat hindered by the Bluetooth modules used to communicate between Picos.

The cursor location extrapolated from the controller movement was fairly accurate. The main issue with accuracy came from only using gyroscopic measurements to determine the movement of the player and thus where the cursor should be placed. It resulted

in slight cursor drifting as the gyroscope measures slight “ghost” movement. This required the group to calibrate the cursor by placing it in the center of the screen. However, no drifting was seen in the x direction when moving the cursor, only in the y direction.

Conclusion

We are pleased with the results of our project, even though there are aspects we would refine if we were to do it again. Our project successfully translated physical motions into digital commands, allowing for a successful interactive and engaging gaming experience. The overall usability of the device is moderate, but more updates and iterations would be needed to get the device to be more user-friendly from a commercial standpoint. However, with some improvements to the system, we believe that it one day could offer a playful mechanism for people in need and become an inclusive digital tool. A faster communication protocol between the Pico and the Bluetooth module and the use of accelerometer data would be two areas of focus to allow a better user experience.

Related projects

https://circuitcellar.com/research-design-hub/projects/barometer/
https://circuitcellar.com/research-design-hub/projects/blood-pressure-record-keeping/
https://circuitcellar.com/research-design-hub/projects/interfacing-with-video-game-controllers/
https://circuitcellar.com/research-design-hub/projects/build-a-speech-controlled-sudoku-solving-robot/