Constrained Computing:
A Pocket-Sized Pong Game on Arduino

CSIS2810: Computer Architecture

Tommy Collier (tcollie8@bruinmail.slcc.edu)
Alex Adams (aadam143@bruinmail.slcc.edu)
Ty GreenBurg (tgreenb2@bruinmail.slcc.edu)
Jacob Gridley (jgridle1@bruinmail.slcc.edu)

Abstract

This project explores the feasibility of implementing the classic arcade game Pong on a .96-inch OLED display controlled by an Arduino Uno with two push buttons. Working within the Arduino's limited resources and the compact nature of the display, this exercise in constrained resource programming demonstrates what can be achieved with hardware costing under $50. Inspired by a basic tutorial, we expanded the concept to include a custom title screen, a score tracker, and adaptive difficulty where the player's paddle shrinks after every four points.

The project highlights the potential of resource-constrained systems to deliver dynamic interactions on small, affordable platforms. It opens avenues for further exploration in embedded systems and cost-effective applications, such as educational tools, mini-games, and real-time monitoring. This work underscores the value of creativity and optimization in developing innovative, low-cost interactive systems.

1.0 Introduction

In an age of high-powered computing and sophisticated graphics, creating a simple, functional game like Pong on a low-processing power microcontroller offers a unique challenge. This project focuses on designing and implementing Pong on a .96-inch OLED display controlled by an Arduino Uno, using only two push buttons for interaction. By working within the strict limitations of Arduino's memory and processing power, our team explored the principles of constrained resource programming to build a dynamic, pocket-sized game experience.

Our goal was to achieve a visually engaging and fully playable version of Pong, incorporating basic gameplay mechanics and additional features, all for under $50. This project demonstrates the potential of minimal hardware to deliver meaningful interactive experiences, demonstrating the adaptability and versatility of Arduino systems in digital interaction.

2.0 Problem Statement

Creating engaging interactive experiences on minimal, low-cost hardware is a significant challenge in embedded systems, portable electronics, and digital entertainment. Such applications typically require substantial processing power, memory, and input/output capabilities, all of which increase cost and complexity. However, in educational, hobbyist, and other resource-limited contexts, there is a growing need for systems that provide meaningful interactions within constrained resources.

This project addresses the challenge of building a responsive digital game, like Pong, on a platform with limited memory, processing power, and display capabilities. It reflects broader industry trends in designing compact, cost-effective systems, with applications in portable medical monitors, control interfaces, and IoT (Internet of Things) devices that need to operate reliably with minimal hardware.

3.0 Proposed Solution

Our solution to creating a playable version of Pong on minimal hardware combines software optimization with efficient design. Using an Arduino Uno as the controller, the game is displayed on a .96-inch OLED screen and operated with two push buttons. The game's logic and visual rendering were tailored to fit within the Arduino's memory and processing limits, ensuring smooth gameplay.

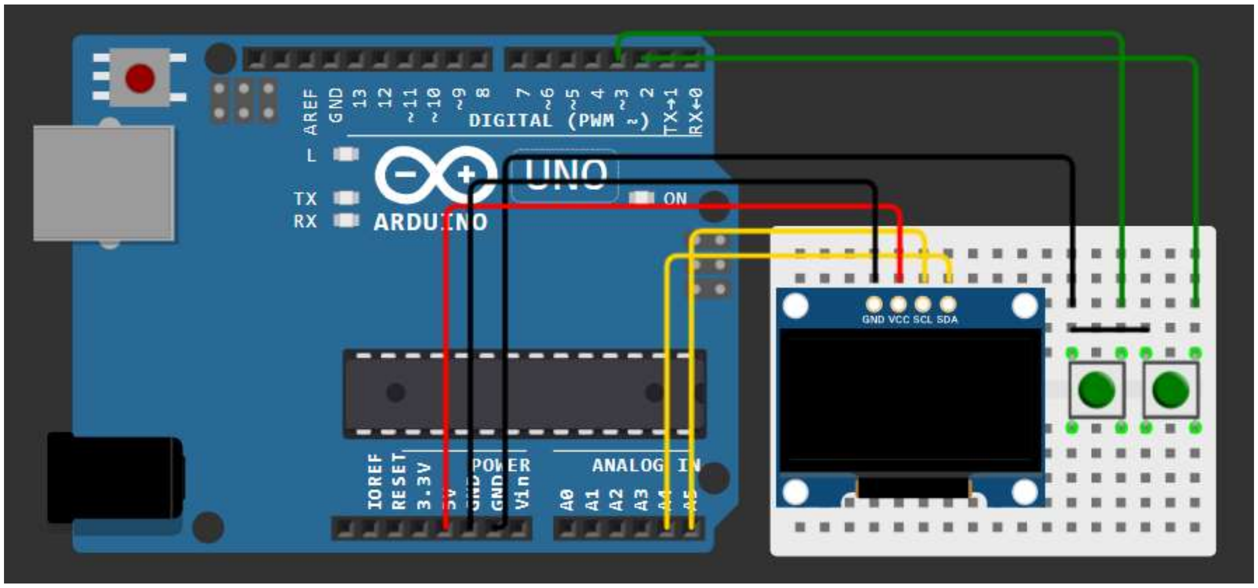


Figure 1: Wokwi Simulation of Hardware Configuration

The player controls a paddle to hit a ball that bounces off the paddle and screen edges. Simple yet engaging mechanics highlight the Arduino Uno's ability to manage real-time input and deliver responsive visuals. Optimized code ensures fluid gameplay, precise collision detection, and an intuitive interface.

3.1 Hardware Selection and Configuration

The hardware used to create the game consisted of nine wires, a two-tone 0.96-inch LED display, a breadboard, two push buttons, and an Arduino UNO R3 microcontroller. The total estimated cost of the project is $50.54 ($61.41 including Utah sales tax). However, this estimate includes unused spare parts such as additional wires, breadboards, and displays. A more accurate breakdown of the components directly used in the project is as follows:

- **OLED display**: ~$2.00
- **Push buttons**: ~$0.10 (two buttons at ~$0.05 each)
- **Breadboard**: ~$0.50–$1.00, depending on size
- **Wires**: ~$0.28 (nine wires at ~$0.03 each)
- **Arduino UNO R3**: $29.00 (retail price)

This brings the adjusted project cost to approximately $32.38 ($39.35 including Utah sales tax) with plenty of spare parts remaining for future use.

The hardware was configured as shown in **Figure 1**, using all listed components. The button wiring allows flexibility, as the power connections can be swapped to assign "up" and "down" controls to the buttons as desired. The port assignments were chosen based on the required functionality and the most convenient layout for gameplay.

The Arduino UNO R3 microcontroller was selected for its flexibility, ease of use, and compatibility with the project's requirements. According to the official Arduino documentation, the UNO R3 "is the most used and documented board of the whole Arduino family" (Arduino). Its user-friendly IDE made uploading and managing code straightforward, while its sustained memory capabilities and optional 9V battery compatibility added versatility. Additionally, the replaceable chip design allows for easy repairs or upgrades, making it an ideal choice for both this project and future applications.

3.2 Memory Constraints

To address the memory and processing limitations of the Arduino Uno, we used optimization techniques such as efficient data structures and streamlined game mechanics. These efforts minimized memory usage, ensured smooth gameplay, and enabled precise collision detection on the OLED screen, demonstrating the potential of resource-constrained programming for compact, low-cost hardware.

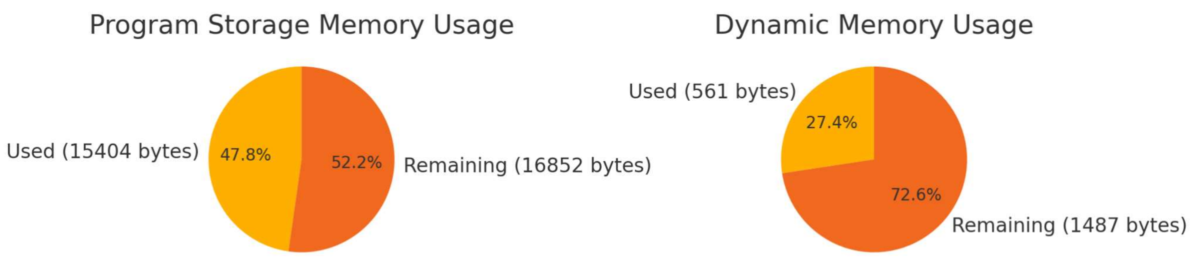
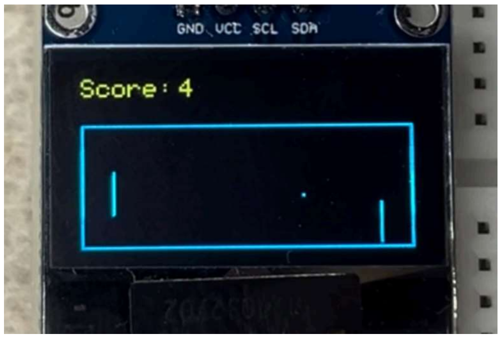


Figure 2: Memory Usage of the Arduino UNO R3

Using the uint8_t (unsigned 8-bit integer) data type for key variables ensured memory-efficient storage while maintaining precision for accurate calculations. This choice also enabled bitwise operations, such as right shift (>>), to efficiently calculate the paddle's midpoint without additional memory overhead.

As noted by EEWeb, "modern C/C++ compilers support types uint8_t, uint16_t, uint32_t, and uint64_t, which allow us to declare unsigned integer variables with widths of exactly 8, 16, 32, and 64 bits, respectively" (Maxfield). Deliberate data type choices such as precise control over variable size are crucial for maximizing performance in memory-constrained systems like the Arduino Uno.

3.3 Integrating a Score Display



During the component selection process, we found that the SSD1306 0.96-inch OLED display featured a two-tone layout, with the top third in yellow and the bottom two-thirds in blue. Leveraging this unique characteristic, we resized the game to occupy only the blue section, reserving the yellow area for a score tracker.

To optimize memory usage, we chose not to implement a dedicated "Game Over" screen. Instead, if the ball's x-coordinate moved past the player's paddle indicating a missed return, the score automatically reset to zero. This approach maintains seamless and continuous gameplay without interruptions.

3.4 Increasing Game Difficulty

During development, we found the initial gameplay lacked challenge, as the ball's speed was too slow to create meaningful difficulty. Attempts to increase the ball's speed revealed a hardware limitation: the ball, represented by a single pixel, could only move one pixel per frame to maintain collision accuracy with the screen edges and paddle. This limitation stemmed from the Arduino Uno's processing capabilities and the OLED display's resolution constraints.

To address this, we pivoted to a different approach for increasing difficulty: gradually shrinking the player's paddle. This mechanism was implemented as a score-based progression system, where the paddle size decreases after every four points scored. This approach adds difficulty while ensuring fairness, as the paddle resets to its original size when the score resets. This solution balances hardware limitations with a dynamic and engaging gameplay experience.

3.5 Replacing the Title Screen

As we refined the game, we noticed that the Adafruit library, used for rendering graphics on the OLED display, by default displays its logo at the start of programs. To implement a custom title screen while minimizing delay for the player, we explored ways to bypass this default behavior. By clearing the display immediately after initializing it, we successfully suppressed the Adafruit logo.

In its place, we designed a custom title screen featuring the class name, "CSIS2810," and the game title, "PONG." This personalized the game and provided a cohesive introduction without adding unnecessary overhead to the program.

4.0 Conclusion

This project demonstrates the feasibility of creating an engaging version of Pong using minimal hardware. By addressing the challenges of constrained resource programming, we delivered a dynamic and intuitive gaming experience on a compact, low-power device. Features such as adaptive difficulty, seamless gameplay, and the innovative use of the OLED's two-tone layout highlight the creativity and precision of this implementation.

The success of this project underscores its educational and practical value, showcasing how affordable components can be optimized to create meaningful interactive experiences. It not only highlights the adaptability of Arduino systems but also provides a clear example of how resource-constrained programming can overcome hardware limitations. This implementation serves as a foundation for future innovations in embedded systems, small-scale computing, and interactive design, proving that even simple devices can deliver dynamic and enjoyable results.

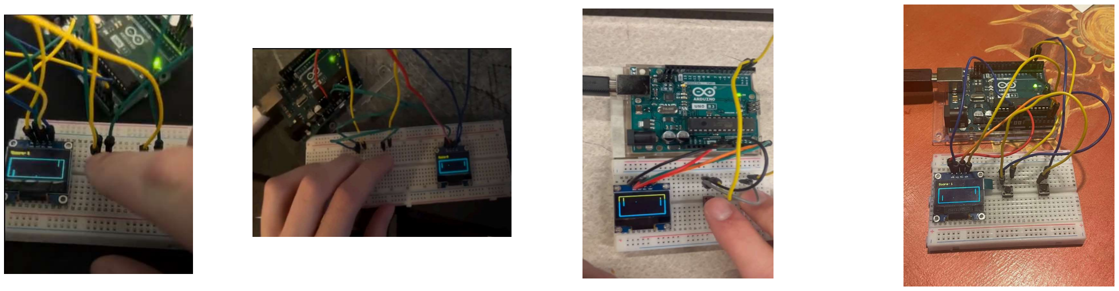


Figure 3: The group's hardware configurations for the project

From left to right: Jacob Gridley's, Ty GreenBurg's, Tommy Collier's, and Alex Adams' setups.

Appendix

- Code Repository: https://github.com/groundctrl2/CSIS2810_FinalProj
- Wokwi Implementation: https://wokwi.com/projects/415194213699900417
  - Wokwi Implementation of the tutorial's version: https://wokwi.com/projects/415223373937591297
- Arduino Serial Peripheral Interface (SPI) Library (reference): https://reference.arduino.cc/reference/en/language/functions/communication/spi/
- Arduino Wire Library (reference): https://reference.arduino.cc/reference/en/language/functions/communication/wire/
- Adafruit GFX Library: https://github.com/adafruit/Adafruit-GFX-Library
- Adafruit SSD1306 OLED Library: https://github.com/adafruit/Adafruit_SSD1306

References

- Arduino. “UNO R3." *Arduino Documentation*, https://docs.arduino.cc/hardware/uno-rev3/. Accessed 20 Nov. 2024
- Adafruit. “Adafruit GFX Graphics Library.” *Adafruit Learning System*, Adafruit Industries, https://learn.adafruit.com/adafruit-gfx-graphics-library/overview. Accessed 20 Nov. 2024.
- Maxfield, Max. "How the C/C++ Shift Operators Work." *EEWeb*, 24 May 2019, www.eeweb.com/how-the-c-c-shift-operators-work/.
- Utsource. “Play a PONG Game with Arduino Uno and OLED 0.96 SSD1306.” *Instructables*, Autodesk, https://www.instructables.com/Play-a-PONG-Game-With-Arduino-Uno-and-OLED-096-SSD/. Accessed 20 Nov. 2024.