PONG

Overview

The history of technology is a subject that has always been very interesting to me, so for this personal project I wanted to try and recreate one of the first video games ever made - Pong. Pong kick-started video gaming as an entertainment genre, and was a key step forwards in computer graphics. For my first step into the world of electronics & computer hardware, this seems like a fitting task to try and tackle!

Aims

My chief aim is to learn the basics of creating electrical circuits, a key skill necessary to many types of engineering. This is a skill I have, currently, no experience with, and I hope by the end of the project to have mastered simple, multi-component circuits. My aims are to:

- Create a basic, functioning circuit
- Further programming skills
- Enhance understanding of electronics and electrical systems
- Create functioning product under a time limit

Timeline

I have allotted two weeks for this project, and I aim to complete 10-20 hours of work during this time, including the time it will take to teach myself some basic circuitry.

Week 1 24th - 31st December

Research & Development - 2 - 5 hours

Key principles of electronics:
- Voltage, current and resistance
- Components: which should I use? Input, output, computation.
- Techniques: soldering, basic circuitry

Prototyping - 2 - 5 hours

- Assemble complete hardware configuration
- Run basic code using the Arduino IDE and Arduino C

Week 2 31st - 7th January

Programming - 5 - 10 hours

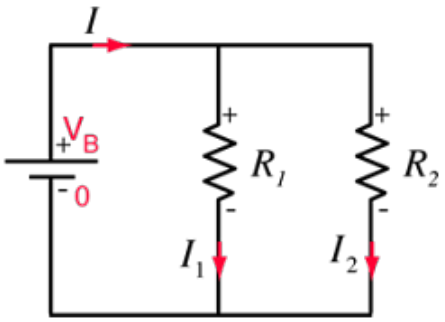
Features:
- Draw to display
- Ball movement & collisions
- Points system
- Win conditions & end screen

RESEARCH

V = IR and types of circuits

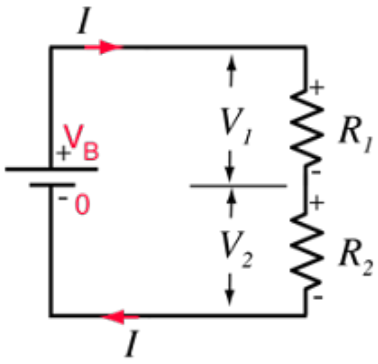
- Resistance = the opposition to the flow of charge (current)
- Each component in a circuit will have its own resistance, as well as the resistance of the wires. To find the total resistance of a circuit, apply the equations below:

- Voltage (volts) = current (Amps) × total resistance (Ohms)



Parallel resistors

$$\frac{1}{R_{equivalent}} = \frac{1}{R_1} + \frac{1}{R_2}$$



Series resistors

$$R_{equivalent} = R_1 + R_2$$

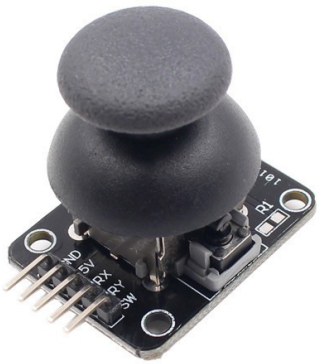
Series circuit - whole current flows through each component.

Parallel circuit - circuit branches, so the current divides, and varies across branches. Voltage however, stays constant.

Components

The IPO (input, process, output) model effectively describes the approach used widely in system analysis and software engineering to describe the function of a program. I needed to carefully consider my input, processing, and output devices for my use case, as they all have different properties and functions.

Input



My input device would dictate the control of the moving bar, the most important, and only, game play feature. As the movement is continuous, I decided an analogue input device would be the best option. The obvious choice is a joystick, as it is both easy to wire, easy to use, and effective.

Processing

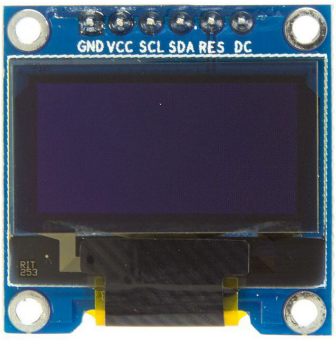
For my processing device, I had a range of choices. There are many microcontrollers with varieties of specifications, but I decided to settle for a middle of the road option - the Arduino Uno. It has a nice 32KB of memory, and more than enough input and output pins for my needs. It also has a handy IDE built just for it, and plenty of online resources to help if you get stuck. It seems like the perfect beginner choice, considering the vast range that is out there.

RESEARCH

Components Cont.

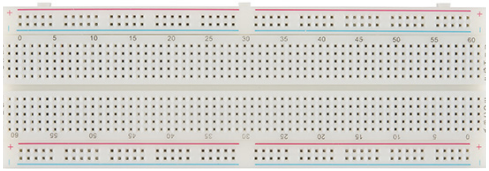
Output

Once again, I had a range of choices for my output device. I knew wanted a screen display, but there were many different types: LED, OLED, LCD, TFT, and in many different resolutions with different pin configurations. In the end I opted for OLED, for the crisp display (relatively) high resolution, and low power cost. My one regret with my choice of output devices was the size, being just over 1 inch across. While on one hand, this is useful, as it saves memory (less pixels to draw to the screen) & power, it can make things slightly uncomfortable to view.

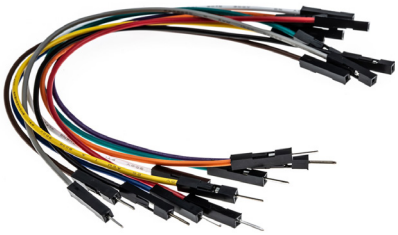


Misc

Finally, some general prototyping components, necessary for the project.



Breadboard



Jumper wires

Solid core prototyping wire

USB A to USB B cable, to connect the Arduino to my computer

Software

To write my code, I will be using the Arduino IDE, due to its compatibility with the Arduino Uno. The language I will be using is Arduino C, which is based on C and C++, but has some small tweaks for ease of use.

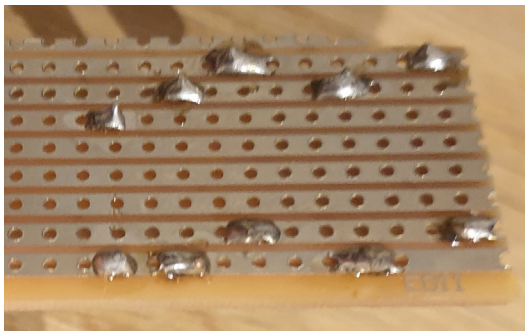
Finally, to display to my OLED display, I will be using the u8g2 library, an open sourced, mono-chrome graphics library for embedded devices. The GitHub link is below:

https://github.com/olikraus/u8g2

TECHNIQUE

Soldering

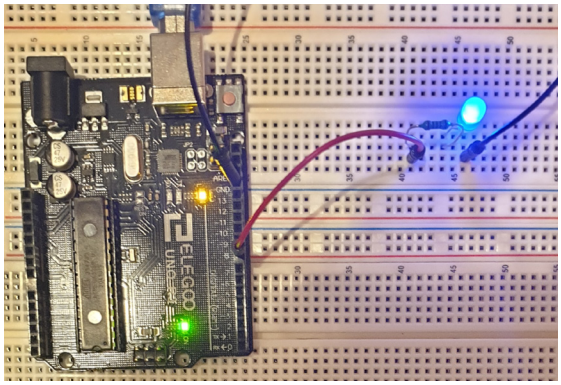
While at this point, I was still unsure on whether I would keep my project restricted to breadboard or not, I figured that soldering was still a valuable skill that I should take the time to learn anyway. I bought myself a soldering iron and some solder wire, and had a go using a piece of prototyping board and some spare components.



First go! Still clearly lots of room for improvement!

Practice circuits

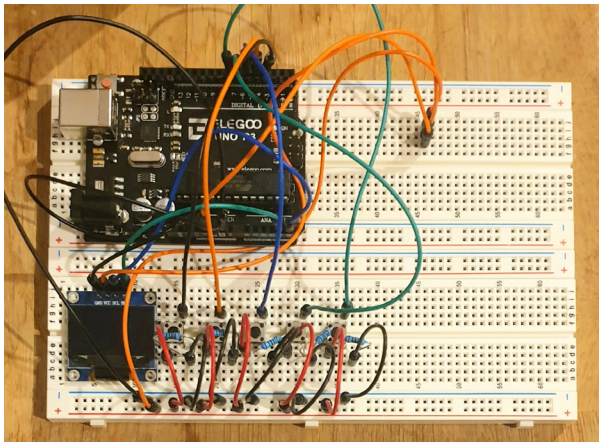
Blinking LED



```
void setup() {
  pinMode(8, OUTPUT);

}

void loop() {
  digitalWrite(8, HIGH);
  delay(1000);
  digitalWrite(8, LOW);
  delay(1000);
```

A simple series circuit consisting only of the Arduino, some wires, a resistor, and an LED. Blinks indefinitely!

Multibutton input to display

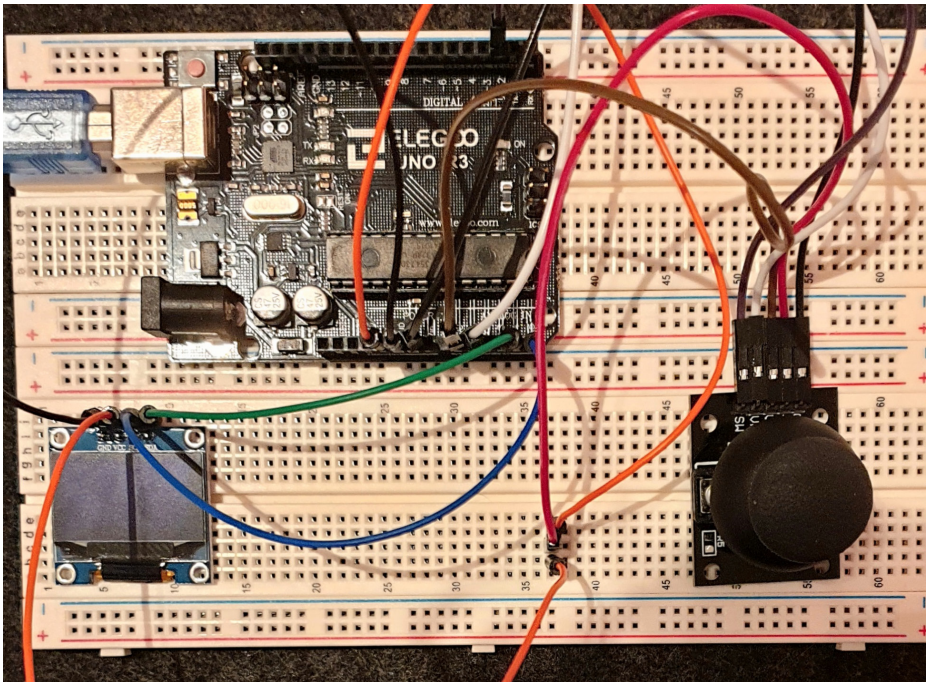


Screen, when button is pressed, will display the number of the pin corresponding to the button input, using the u8g2 library.

```
if (buttonState6 == HIGH) {
  u8g2.drawStr(0,15,"6");
}
```

(Code only partially shown)

PROTOTYPING

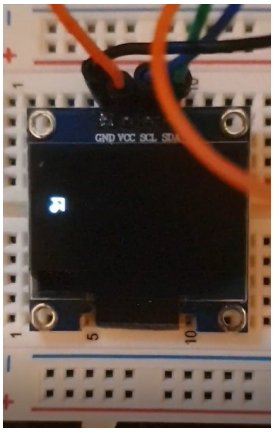


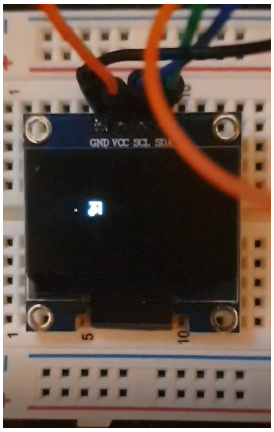
The first thing I did was assemble my circuit, and try running some code taking input from the joystick and displaying output to the screen.

```
void loop() {

  xValue = analogRead(A0);
  yValue = analogRead(A1);
  bValue = digitalRead(2); //default is 1

  xMap = map(xValue, 0, 1023, 0 , 120); //joystick input to pixel coordinates on screen
  yMap = map(yValue,0, 1023, 0, 57);

  u8g2.clearBuffer();

  if (bValue == 0) { //if joystick clicked - draw clicked mouse
    u8g2.drawXBM(xMap, yMap, clicked_width, clicked_height, clicked_bits);
   }
   else {
    u8g2.drawXBM(xMap, yMap, image_width, image_height, image_bits);
   }


  u8g2.sendBuffer();

}
```

I mapped the analogue input from the joystick to a range of 0 - 128 in the x axis, and a range of 0 - 64 in the y axis. This allowed me to use the mapped values to draw the 'mouse' on the screen, which is simply a bitmap of lit and unlit pixels.

(Code only partially shown)





The full video of & the full code for the mouse demonstration can be found in my GitHub repository for this project, linked below. There is also a click animation for when you press down on the joystick!

https://github.com/crabcakes6/awkright

PROGRAMMING

With my circuit and joystick input done, I could now move onto the rest of the programming. I started on getting the player's paddle moving, as well as the ball.

```
u8g2.drawRFrame(xMap,58,paddleLength,5,1);
```

One line of code was enough for the paddle, although the ball would be a little more complicated.

Not only did it need to move, it needed to bounce off the walls and the paddles. To do this I used conditional statements that check the X and Y position of the ball, and reverse the direction of movement if the X/Y positions are out of bounds. Examples shown below:

```
//bounce off the top & bottom walls
  if (ballY <= 3){
    ballDirY = -ballDirY;
    ballY = 4;
  }
  if (ballY >= 62){
    ballDirY = -ballDirY;
    ballY = 61;
  }
//check if hit player paddle
   if (ballX <= 5 && ballY >= yMap && ballY  <= yMap + paddleLength){
     ballDirX = -ballDirX;
     ballX = 6;
     ballSpd += 0.16;
```

Next, I wrote the AI for the CPU paddle, which was surprisingly simple, consisting only of some conditional statements that would move the paddle to match the Y position of the ball.

```
//cpu ai
  if (cpuY + paddleLength > ballY){
    cpuY -= 1;
  }
  else if (cpuY + paddleLength < ballY){
    cpuY += 1;
  }
  if (cpuY < 1) {
    cpuY = 1;
  }
  if (cpuY > 64-paddleLength){
    cpuY = 64-paddleLength-2;
  }
```

Now the game was playable, I decided on the end conditions, and made a point system. If, for example, the player's paddle misses the ball, the CPU gains a point.

```
else if (ballX <= 5){
  cpuScore += 1;
  restart();
}
```

PROGRAMMING

Once either players reach 5 points, the game ends. I pass a parameter to the gameEnd function, which dictates which player won.

```
//game end conditions
  if (playerScore == 5){
    gameEnd(0);
  }
  else if (cpuScore == 5){
    gameEnd(1);
  }
```

In the gameEnd function, I first convert my player & CPU scores (integer data types), to strings using the built in C function "itoa". I then use the C function "strcat" to concatenate my two strings, so I can use the u8g2 function "drawStr" to display the scores on screen as well as some text.

```
char a[15] = "Player score: ";
char b[2];
itoa(playerScore, b, 10);
strcat(a,b);
u8g2.drawStr(2,47,a);
```

Finally, I display which player won (according to the passed parameter), and allow the player to restart (while loop within gameEnd function is broken) by clicking the joystick.

```
if (x == 0){
  u8g2.drawStr(2,10,"Player wins!");
  u8g2.drawStr(2,24,"Click to restart.");
}
else {
  u8g2.drawStr(2,10,"CPU wins!");
  u8g2.drawStr(2,24,"Click to restart.");
}
u8g2.sendBuffer();

if (bValue == 0){
  cpuScore = 0;
  playerScore = 0;
  break;
}
```

I have left the explanation of some features out (as they are not very interesting!), but the entirety of my code can be found in my GitHub repository for this project, linked below.

https://github.com/crabcakes6/awkright

EVALUATION

Problems

- Needed to power both the joystick and the display, but the Arduino only had 1 5V pin
- No buttons for input
- Could not use drawStr function with an integer

Solutions

- Connect the joystick and the display to each other, before connecting them to the 5V pin
- Use a joystick with a built in button, so no need to connect another one
- Cast integer value as a string, and concatenate it to the other string I wanted to display

Successes

- Finished project!
- Finished within allotted time
- Fit program into 32KB of memory
- Advanced understanding of electronics
- Advanced understanding of C
- Used an Arduino for the first time

Failures

- I had wanted to implement a high score system as well difficulty levels, but I found that there wasn't really an elegant way to offer choice between levels considering the lack of input variety and the memory limit of the Nano. If I had chosen the Arduino Mega and added a few buttons, this may not have been a problem, al though this would have increased the footprint and complexity of the device significantly.
- Display quite small

Learning Experiences

I'm very happy to have done this project, as I feel that I have definitely built on skills in programming, as well as having learnt the more novel skills of basic circuitry and electronics.

The mechanics behind embedded systems have always interested me, and building something with limited components has given me a slight insight towards how they are made, even if the parts they would use are far more specialised, and the code they would run on would likely be in a far more low level language, like Assembly or plain C.

The area of this project that I found the most interesting was learning about the operation of the Arduino, specifically what each pin does and how to connect them to other components. It's great to now have a more solid understanding of more simple computer systems, and I feel like this has helped me solidify my understanding of more complex ones, like the laptop I am typing this on right now!

In engineering, you need to understand both the most simple and the most complex versions of the systems you are working on, so putting together a more basic one has been very useful to me.

EVALUATION

I also learnt about optimisation, and how to prioritise certain features over others. Considering the relatively low memory of the Uno, I couldn't include as many features as I wanted, and had to trade, say, a high score system for a smaller footprint. This is important, as in engineering compromises often have to be made in order to make a product the best it can be. I also learnt how to improve the efficiency of my code, which was again important due to the memory constraints. I made sure to not repeat blocks of code, and use more simple solutions when possible.

Going Forwards

I feel like, in this project, I have slightly relied on the Arduino and my own programming skills as a crutch, as in a project where I set out to further my understanding of electrical systems, I have spent more time coding than actually assembling anything physical.

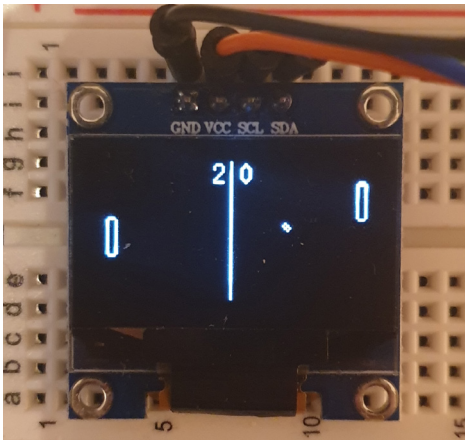
I want to learn more about how the Arduino itself works, and I plan on learning about the design and production of PCBs and motherboards, as well as more about NAND/NOR technology in both computation and storage.

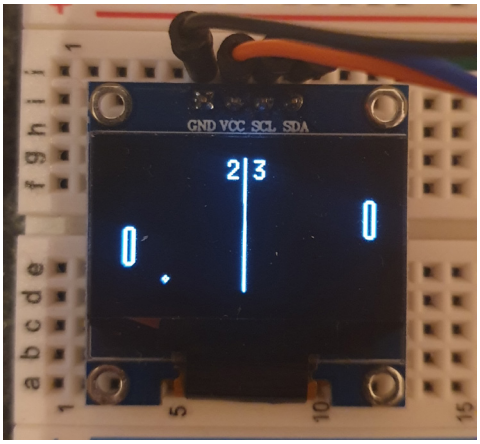
I also want to try put together some systems not using the Arduino, perhaps to monitor water levels, or to control a collection of LEDs, or to turn on my fan when it gets too hot.

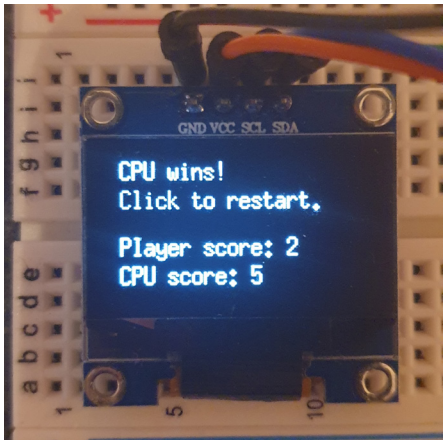
Generally, I think that there is a lot to discover, and I am excited to go further and learn more!

Final Product

I have included some images of my the display below, as well as the link to my GitHub repository for this project, including a video of the game.







https://github.com/crabcakes6/awkright