# Critical Design Review: Puzzle Me Chess

February 22, 2021

*EN.525.743.8VL.SP21*

**Mr. Joseph Vitale**

# Contents

| Capability | Description below |
|---|---|
| Indicator | LED light indicator to show user hint/show answer feature |
| User Direction | OLED to describe to the user where to put pieces |
| SD Card | Read .csv standardized format to quickly import puzzles |
| User Direction | User input switch to show user answer, indicated by LEDs |
| User Direction | User input Button to show user a hint |
| Location Detection | Check board spots are correct for puzzle that was selected |

Table 1: Shows the Capability's for Puzzle me Chess

| Assumptions | Description below |
|---|---|
| User Implication | User puts right pieces on each spot |
| Physical | Chess Board will be lit evenly with light |
| Text File | Each file added to the SDcard will be in a standardized format |

Table 2: Shows the Assumptions that were made for Puzzle me Chess

## Project description

In chess there are many different ways to play the game. After each player moves three times there are 121 million different moves that can end the game. Many people need to practice chess in order to get better and there are many ways to do this. You can play over the internet and play computer which range from (400 - 3200) EIO or you can play puzzle's in chess. Puzzle's in chess allow you to study the board in a given state and make the next several moves. This project is dedicated to solving puzzles in the physical world.

The physical chess puzzle allows the user to practice looking at a physical board to solve each puzzle. This project is titled "Puzzle me Chess" and it will be equipped to have 3 different puzzle's. The chess board will allow the user to place each piece in the right location that is displayed on the OLED screen. After each piece is on the board the chess puzzle will then display to the user which color he/she will play. After the user makes his first move the board will check to see if the move was correct. If not the user will be asked to reset the piece and try again until puzzle is complete.

### Capabilities

Capabilites for the Puzzle me Chess project range from Indicators, User Direction, SD card capability, and Location Detection. See below for more details. More detailed explanation can be found in table 1.

### Assumptions

Table 2 shows the assumptions that are being made for this project. The assume were made to reduce overall cost of the project and difficulty to lower risk.

| Limitations | Description below |
|---|---|
| Location Detection | Not knowing which piece is on the spot |
| Auto Movement | Not being able to move the piece on correct spot |
| Puzzle Selection | Not having 3+ different puzzles to choose from |
| Physical | Not being able to light each square along the perimeter |
| Physical Parts | Not having multiple colors to indicate wrong or right answer |
| Physical | Needing light to illuminate the chess board |

Table 3: Shows the Limitation's for Puzzle me Chess

### Limitations

Limitations for the Puzzle me Chess project range from Location Detection, Auto Movement, Puzzle Selection, Physical and Physical parts. See table 3 for more details.

## Functional description

The Puzzle me Chess project will feature a microcontroller, one 21x21 wooden chess board, 8 Multiplexers, 1 switch, 1 Potentiometer, 1 button, and A Display. Figure 1 shows the current circuit layout as of 02/16/2021, this circuit doesn't show the chess board. Explanation of the System Block Diagram to follow.

### System Block Diagram

Figure 2 shows the sudo hand drawn block diagram. The project consist of One Microcontroller which will feature the teensy 3.6, 8 Multiplexer's from Texas Instruments part # CD4051BE, 1 basic breadboard switch, 1 breadboard Potentiometer, 1 basic breadbaord button, LED's, Photoresister's, 1 Chess Board and one Display.

The Chess board will have 1 LED and 1 photoresister per block. The LED will be drilled and placed on the top left side of each block and will be used to show the user where to place each piece. The photoresister will be drilled in the middle of each block and will be used as a voltage drop to tell if there is a piece there or not. See section multiplexer circuit diagram for the circuit layout.

### Multiplexer Circuit Diagram

Figure 3 shows the sudo circuit for Muxltiplexer A, the puzzle me chess project will consist of 8 Muxltiplexer one for each column A-H. Each of the Muxltiplexers will have 8 outgoing lines that will read Voltages levels from the rows # 1-8. Each row will have one resister which will is set to be $2M\Omega$ and one Photoresister in series. Each Photoresister has two modes a light and a dark mode. Equation (1) shows Mux input voltage at 3.29VDC and equation (2) shows Mux input voltage at 2.2VDC.

Assume Light; R = $500\Omega$

Assume Dark; R = $1M\Omega$

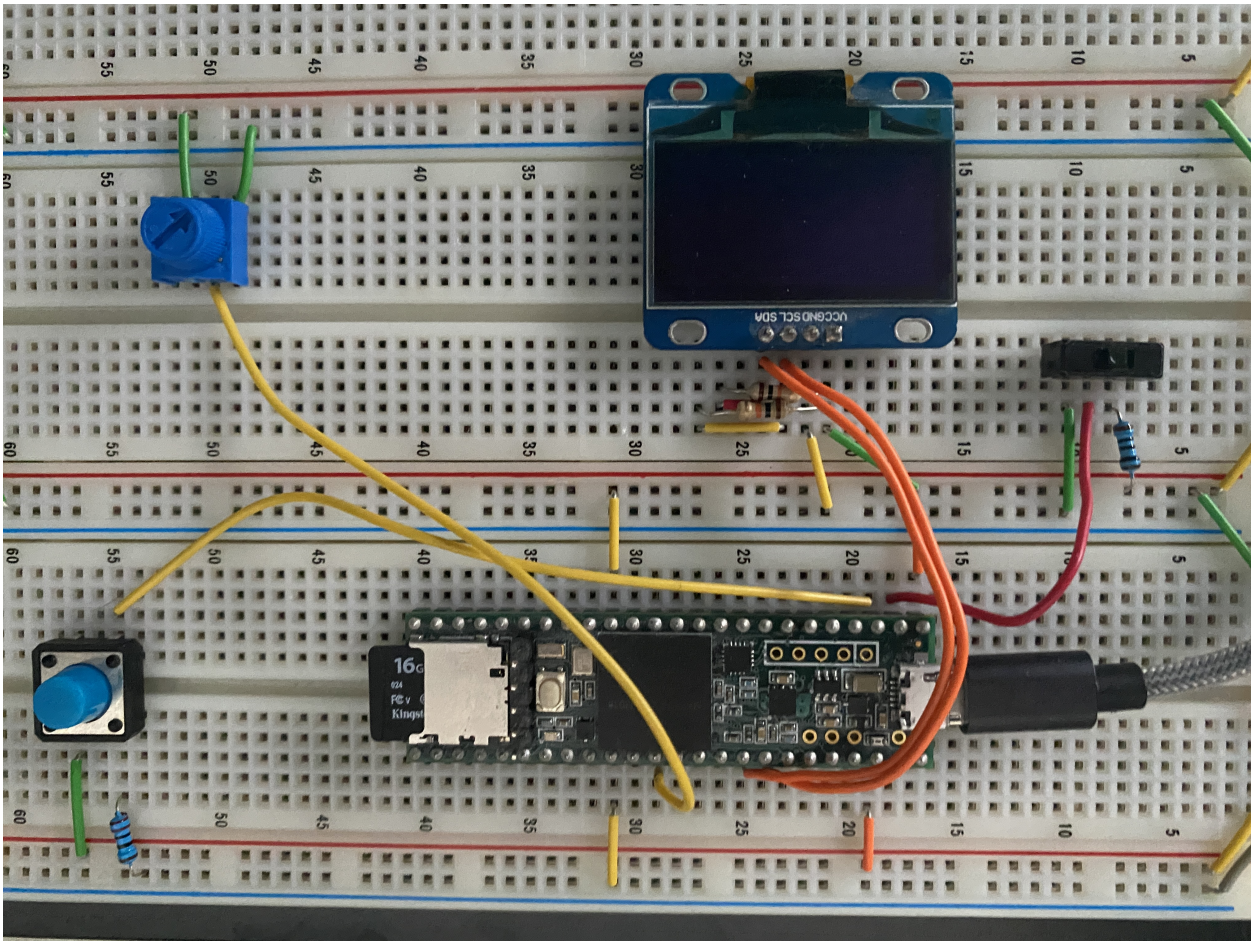


Figure 1: Circuit layout as of 02/16/2021

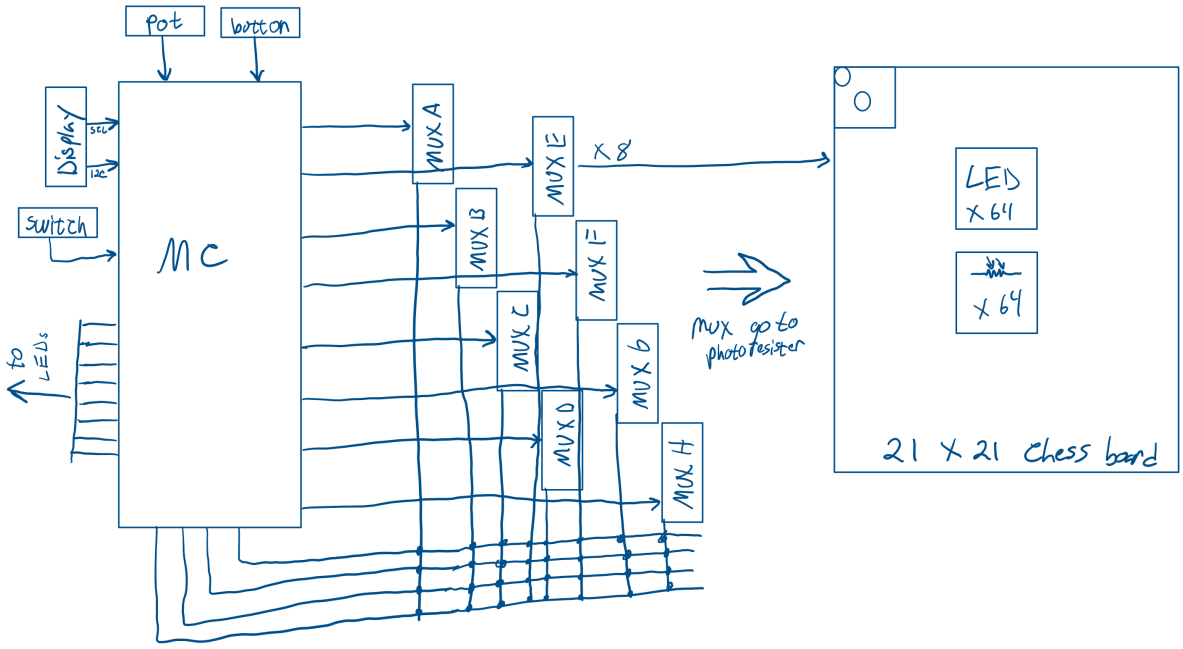


Figure 2: System Block Diagram

*Light*

V = IR, 3.3VDC = I*(2M + 500)Ω

I = 1.65$\mu$A, Vlight = 825$\mu$VDC, Mux input would see 3.299VDC if no piece on top of photoresister

(1)

*Dark*

V = IR, 3.3VDC = I*3.3MΩ

I = 1.1$\mu$A, Vdark = 1.1VDC, Mux input would see 2.2VD if piece was placed on block

(2)

## Code Flow Diagram

Figure 4 shows the code flow diagram. The code starts with the display showing the user to pick a puzzle which the user will use the potentiometer to select puzzle 1-3. After the Puzzle has been selected the display will show which puzzle the user selected and open that .csv file. The code will then mapp each space to a 2x2 matrix with a cell name attached. The code will then display to the user which to place each piece on which block. LEDs will also light up to quickly help the user

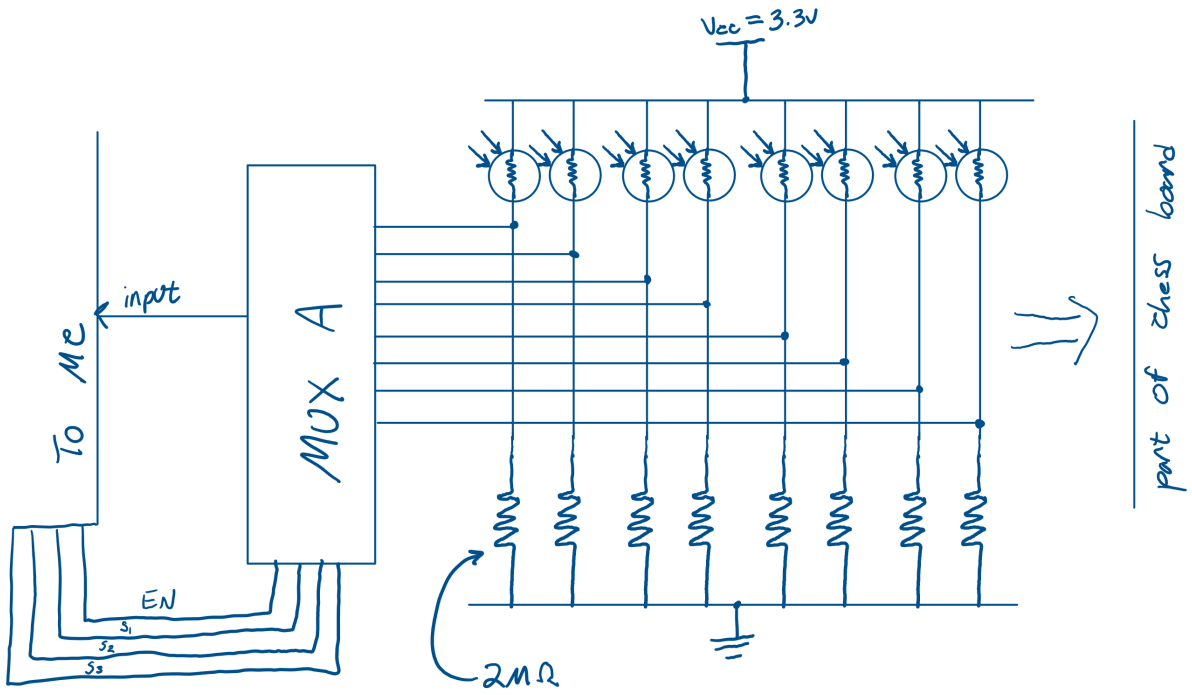


Figure 3: Multiplexer Sudo Circuit Diagram

where to place each piece. After each piece has been placed the code will shut off each LED light and display to the user which color to go first.

The code will have the ability to have the user use a button to light up LEDs to show the user the hint, and a switch to show the user the answer if they get stuck. If the wrong move is made on the chess board the code will have the ability to check each move and display if the move is correct or not. If the move is correct the display will show where the user needs to move the opponent piece so the user can continue the puzzle until completion. If the move is incorrect the user will be asked to place the piece back and try again.

## Interface description

The internal interface description consist of a display, switch, button, mulitiplexer, Potentiometer, and a SD Card. More information can be found in table 4. Charlieplexing is the method that will be used to turn on and off each LED that is needed to show the user where to put each piece, show hint and or show the answer. This equation to determine how many digital I/O pins needed to do this can be found in equation (3).

*Charlieplexing*

Pins $= \frac{1}{2} * (1 + \sqrt{1 + 4 * L}$; L=LED=64, Pins = 8.5

(3)

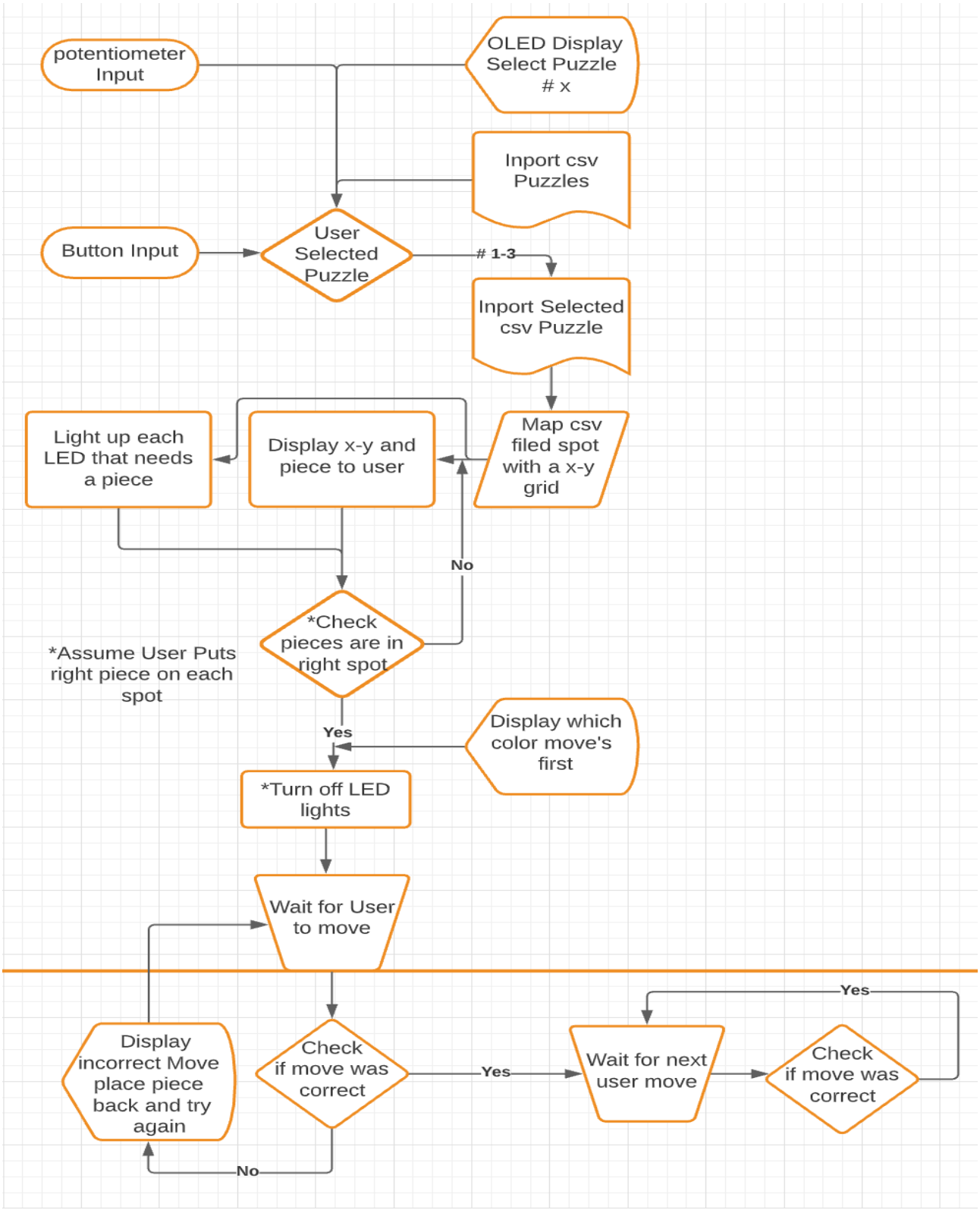


Figure 4: Code Flow Diagram

| Interface Description | Description below |
|---|---|
| Display | one SCL 600MHz, one I2C |
| Mulitiplexer | 3 select digital I/0 pin, 1 enable digital I/O Pin, 8 Input digital I/O Pins |
| Switch | one digital I/O input pin |
| Button | one digital I/O input pin |
| Potentiometer | one analog PWM I/O input pin |
| SD Card | I2C interface 4bit |
| LEDs | 9 Digital I/O pins to light up 64 LEDS |

Table 4: Shows the interface for Puzzle me Chess

## Explanation of code

The following sub sections will describe different parts of the current code as it stands on 02/18/2021. There will be a brief explanation of the code layout which uses reference [1]. Other code explanations will be outlined in Main, Display, SDcard,Button,and Potentiometer, see sections below for more information.

### Code layout

Figure 5 shows the modulatory of the Puzzle me Chess code. Each component of the circuit has its function and for that it will get its own .h and .cpp files. As you can see in figure 5. This code modulatory is talked about in reference [1] and it states laying out the .h files to have a method that is used for one function. An example of this would be the Display having a method for initialization of the display, a print or draw function, and a clear function. This modulatory was done for each component a in the code and is clearly show in figure 5. For developing each method a .cpp file is used and these files will be discussed below in figures 6, 7, 8, 9, and 10. All code below is current as of 02/18/2021 and is not finalized.

### Main

The main.cpp file in Arduino houses the main code. Shown in figure 6 is the current main.cpp file code for Puzzle me Chess. At the top houses all the header files. Each header file is tied to its individual methods which are used in the individual .cpp files. Looking at figure 6 you have the header files and then come the declares. In the declare section there are method callers which grab the latest value of that method at any given time.

After the declares you call a function void setup() which generally would run one time and call most start up methods and initialization processes. Once the setup() function completes its run it then starts the main loop. In Arduino we use void loop() which loops every time it gets to the end but doesn't return any value. Currently I'm using two while loops which let the program pause as we wait for the user to enter in a value. The first loop has the user select a puzzle and then the second loop tells the user which puzzle has been selected and then will open that .csv file.

### Display

Figure 7 shows the .cpp file that holds all of the methods that are used within the main.cpp file and ran in void setup() and or void main(). Within the Display.cpp file there is the declare area

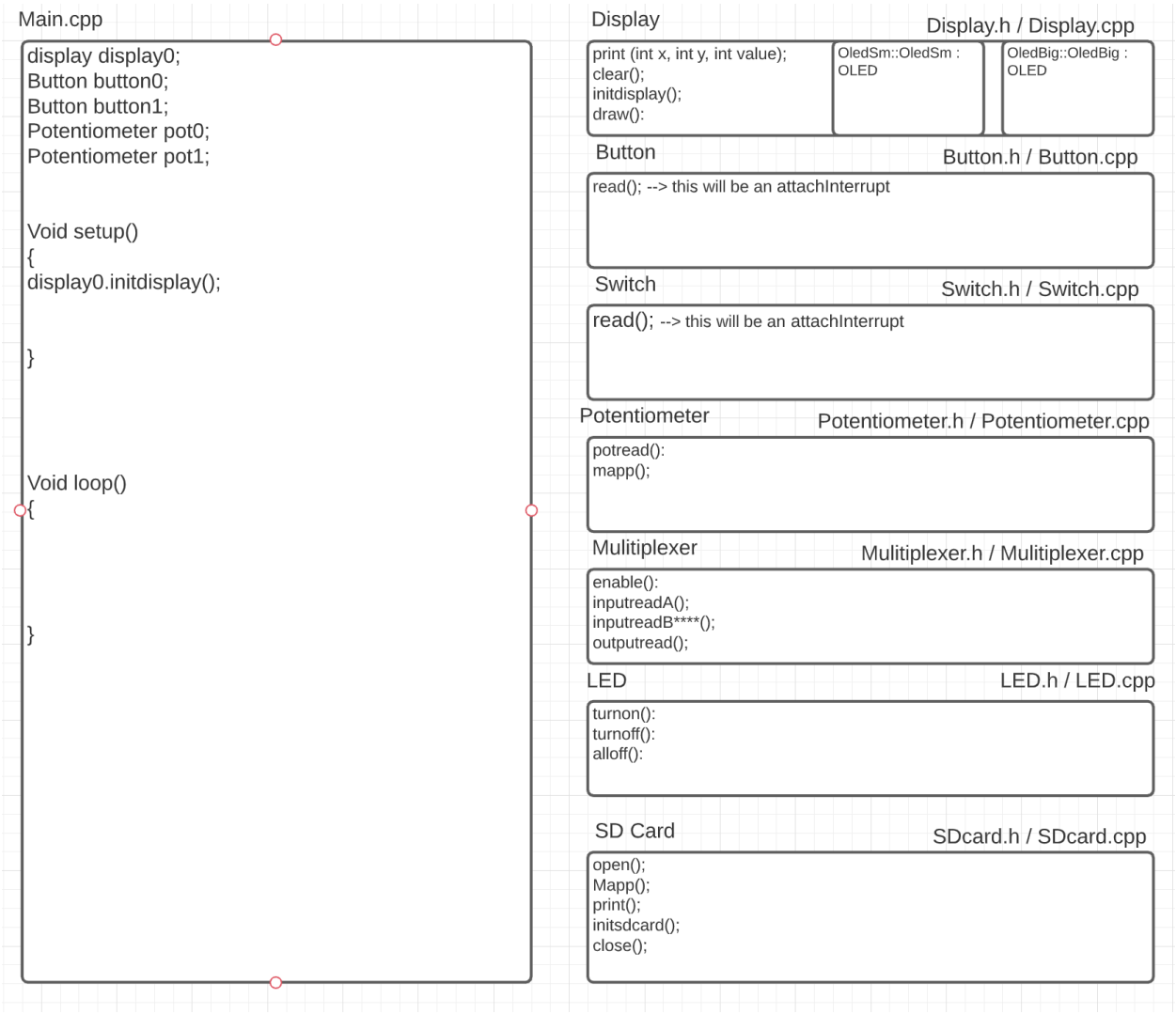


Figure 5: Module Code Layout

```
#include <Arduino.h>
#include <Potentiometer.h>
#include <Display.h>
#include <SDcard.h>
#include <Button.h>

//****************************************Declare****************************//
Display Display0; // Setting Object 0 for Display
SDcard SDcard0;
Button Button0;
Potentiometer Potentiometer0;
//****************************************Setup****************************//
void setup()
{
  Display0.int_display();
  SDcard0.int_SD();
//****************************************Inputs****************************//
  Button0.init_button(); //setting D0 to button
  Potentiometer0.init_pot(); // setting A0 to pot
} // end setup

void loop()
{
//****************************************Declare****************************//
Button Button1;
Potentiometer Potentiometer2;
Display Display1;
//****************************************Start of Code****************************//
while (Button1.r_button() == 0)
{

  Potentiometer Potentiometer1; // moved from Declare b/c I need to pull value continuous
  Display1.print_select_puzzle(45, 30, Potentiometer1.r_pot());
}

delay(1000); //--> to allow for button press

while (Button1.r_button() == 0)
{
  Display1.print_user_puzzle(0,30, Potentiometer2.r_pot());
  SDcard0.open();
  break;
}
```

Figure 6: Puzzle me Chess - main.cpp file

which tells the code which display we are using and how to mapp the pins. After the declare the init.display() function is defined to start the display, followed by print.select.puzzle which shows the user which puzzle to select and the print.user.puzzle which will show the user which puzzle he/she selected.

### SDcard

### Button

### Potentiometer

## Material and resource requirements

### List of Items (BOM)

### Test equipment

### Ording from

include date and times and websites

## Development Plan and Schedule

### Order of Development

### Milestones and Schedule

### Risk

## References

[1] Brain Stone. *C++ Functions Modular Program Design.* (English) *Lecture presented at W 3.1, 3.2* . Dublin City University Glasnevin, Dublin, 2001.

```cpp
#include <Display.h>
#include <Potentiometer.h>

//****************************************Declare*****************************//
U8G2_SH1106_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, /* reset=*/U8X8_PIN_NONE);
//****************************************Setup*******************************//
void Display::int_display()
{
    u8g2.begin(); // Start the Library code for the Display
} // end void int_display
//***************************************Functions****************************//
void Display::print_select_puzzle(int x, int y, int value)
{
    u8g2.clearBuffer();              // clear the internal memory
    u8g2.setFlipMode(0);             // Flips display 180 (1) = True
    u8g2.setFont(u8g2_font_9x18_tf); // choose a suitable font
    u8g2.drawStr(0, 12, "Select Puzzle");
    u8g2.drawStr(x, y, "#");
    u8g2.setCursor(60, 30); // set cursor location
    u8g2.print(value);
    u8g2.sendBuffer(); // transfer internal memory to the display
} // end void print_select_puzzle

void Display::print_user_puzzle(int a, int b, int value1)
{
u8g2.clearBuffer();
u8g2.setFlipMode(0);
u8g2.setFont(u8g2_font_9x18_tf);
u8g2.drawStr(0, 12, "User Selected");
u8g2.drawStr(a, b, "#");
u8g2.setCursor(15, 30);
u8g2.print(value1);
u8g2.drawStr(30, 30, "Puzzle");
u8g2.sendBuffer();
} // end void print_user_puzzle
```

Figure 7: Puzzle me Chess - display.cpp file

```
void SDcard::open()
{
    if (!SD.begin(chipSelect))
    {
        while (true);
    }
    File dataFile = SD.open("1015704.CSV" ); //opening File T015704.csv

    // if the file is available, write to it:

    if (dataFile)
    {
        while (dataFile.available())
        {

            Serial.write(dataFile.read());
        }
        dataFile.close();
    }

    // if the file isn't open, pop up an error:

    else
    {
        Serial.println("error opening 1015704.CSV");
    }
}
```

Figure 8: Puzzle me Chess - SDcard.cpp file

```
#include <Arduino.h>
#include <Button.h>

void Button::init_button()
{
pinMode(2, INPUT); // sets the digital pin 0 as input
} // end init_button

int Button::r_button()
{
int buttonstate1 = 1;
buttonstate1 = digitalRead(button1);
return (buttonstate1);
} // end r_button
```

Figure 9: Puzzle me Chess - button.cpp file

```
#include <Arduino.h>
#include <Potentiometer.h>

//*****************************************Functions*************************//
int Potentiometer::r_pot()
{
  int potmap1 = map(pot1, 0, 1023, 1, 3); // map values 1-3
  return (potmap1);
} // end r_pot

void Potentiometer::init_pot()
{
  pinMode(0, INPUT); // pin A0 mapped to an INPUT --> r_pot
} // end init_pot
```

Figure 10: Puzzle me Chess - Photentiometer.cpp file