# Arkanoid on Arduino

SI project 2019

Rotuna Răzvan Harald
Soreg Andra Marina

# Project description

A simplified version of the Arkanoid game.

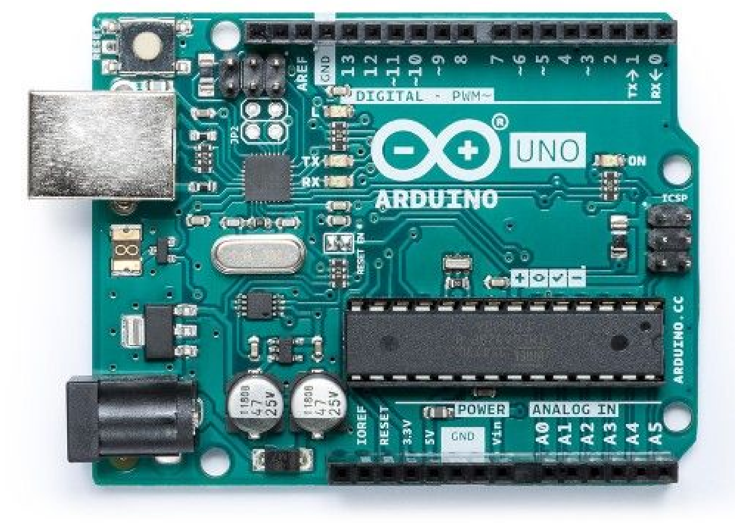
Characteristics:

- there will be defined 3 types of blocks depending on their hardness;
- for displaying the game pieces, one of the following will be used: a LED matrix (with a dimension of 8x8 or more), OLED display, TFT display etc;
- the player will be represented by a block positioned at the bottom of the screen;
- the left or right movement of the player will be done with a button corresponding to each direction (a button press will correspond to a movement with one position);
- optionally, bonuses can be implemented (extra life, stronger ball etc).

# Development board description

The chosen development board is **Arduino Uno**.

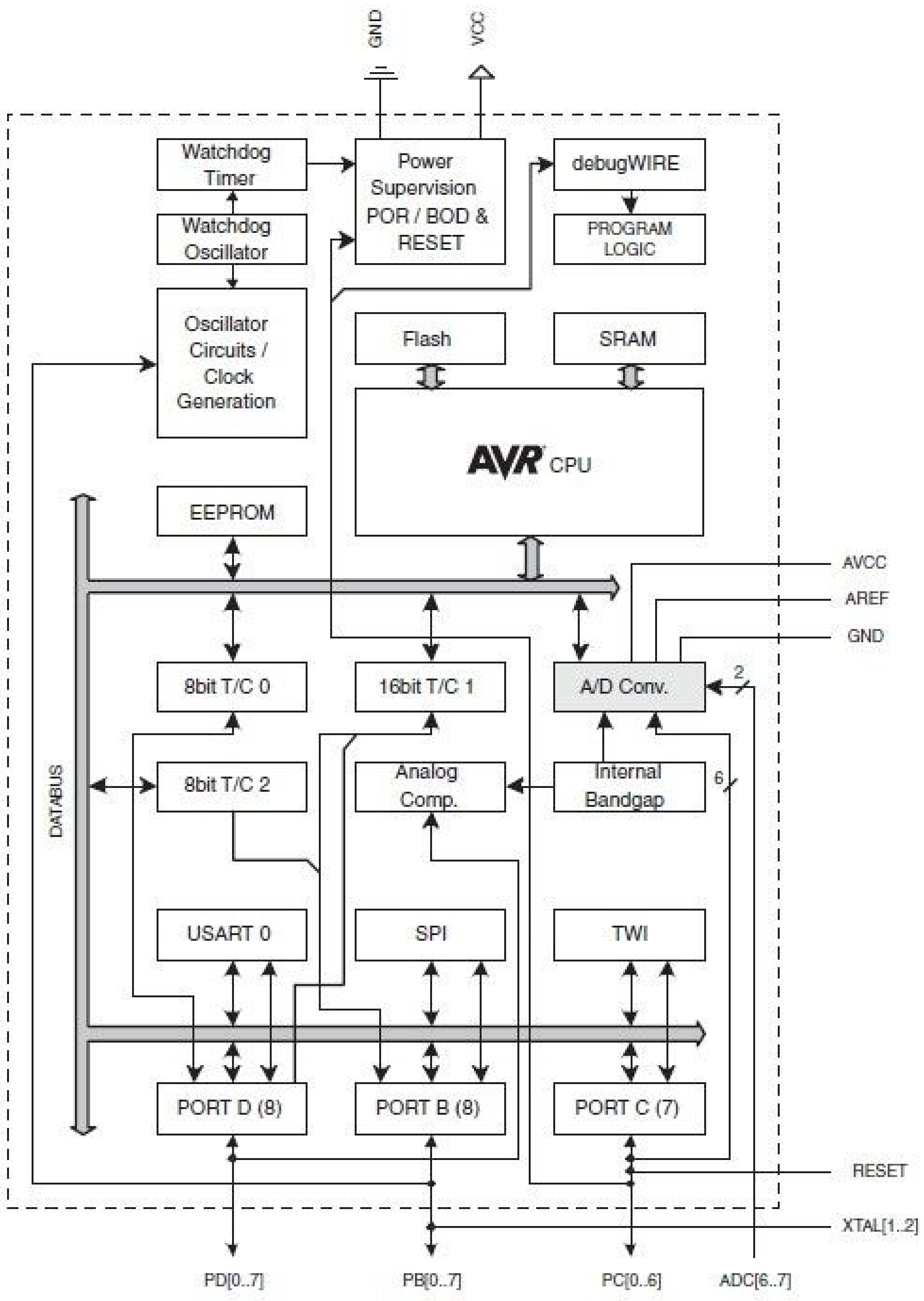
It is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button.



## ATmega328P

ATmega328 is a single-chip microcontroller created by Atmel.

It is an 8-bit AVR RISC-based microcontroller that combines 32 kB ISP flash memory with read-while-write capabilities, 1 kB EEPROM, 2 kB SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible timer/counters with compare modes, internal and external interrupts, serial programmable USART, a byte-oriented 2-wire serial interface, SPI serial port, 6-channel 10-bit A/D converter (8-channels in TQFP and QFN/MLF packages), programmable watchdog timer with internal oscillator, and five software selectable power saving modes. The device operates between 1.8-5.5 volts. The device achieves throughput approaching 1 MIPS per MHz.



*Block diagram for ATmega328P*

# Used microcontroller modules

## SPI (Serial Peripheral Interface)

The SPI module was used for communicating with the display.

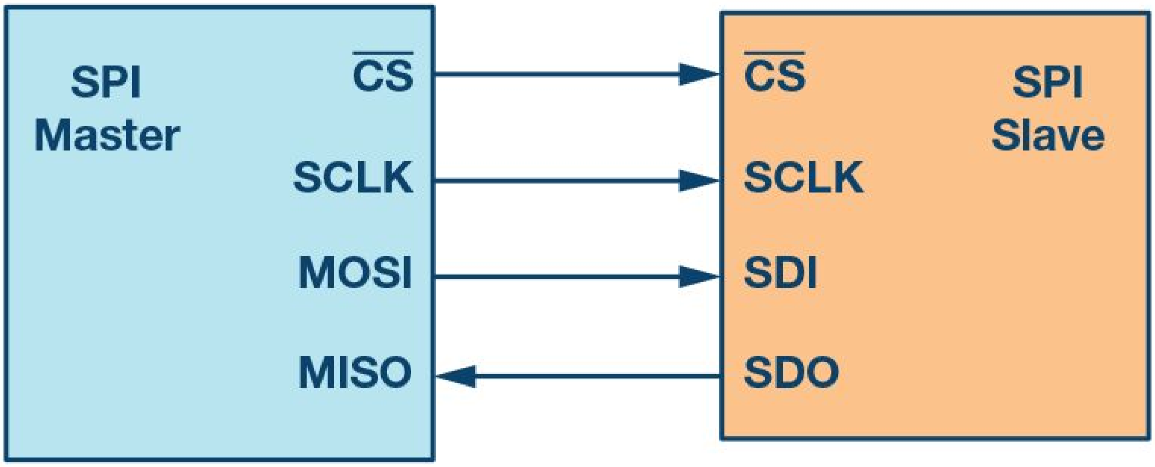
The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems.

SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device originates the frame for reading and writing. Multiple slave-devices are supported through selection with individual slave select (SS) lines.

The data from the master or the slave is synchronized on the rising or falling clock edge. Both master and slave can transmit data at the same time. The SPI interface can be either 3-wire or 4-wire.

4-wire SPI devices have four signals:

- Clock (SPI CLK, SCLK)
- Chip select (CS)
- Master out, slave in (MOSI)
- Master in, slave out (MISO)



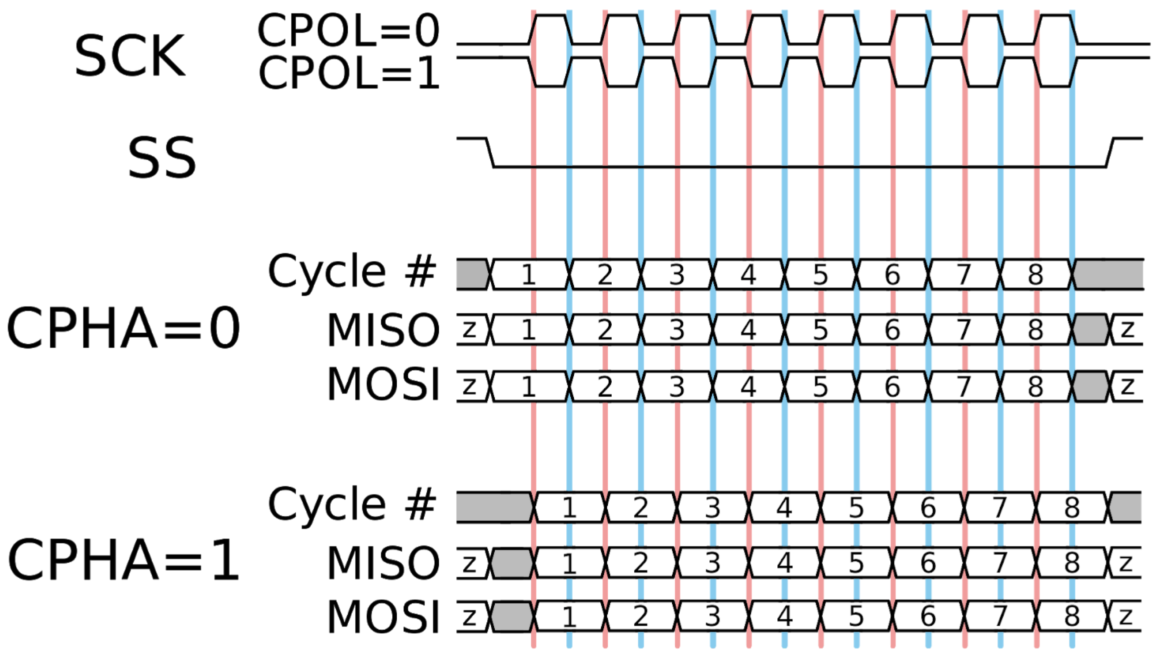
SPI is a full-duplex interface; both master and slave can send data at the same time via the MOSI and MISO lines respectively. In the case of our project, the MISO port is unused, as the display does not need to send any signals to the microcontroller.

## Data transmission

To begin communication, the bus master configures the clock, using a frequency supported by the slave device, typically up to a few MHz. The master then selects the slave device with a logic level 0 on the select line. If a waiting period is required, such as for an analog-to-digital conversion, the master must wait for at least that period of time before issuing clock cycles.

During each SPI clock cycle, a full-duplex data transmission occurs. The master sends a bit on the MOSI line and the slave reads it, while the slave sends a bit on the MISO line and the master reads it. This sequence is maintained even when only one-directional data transfer is intended.

## Timing diagram



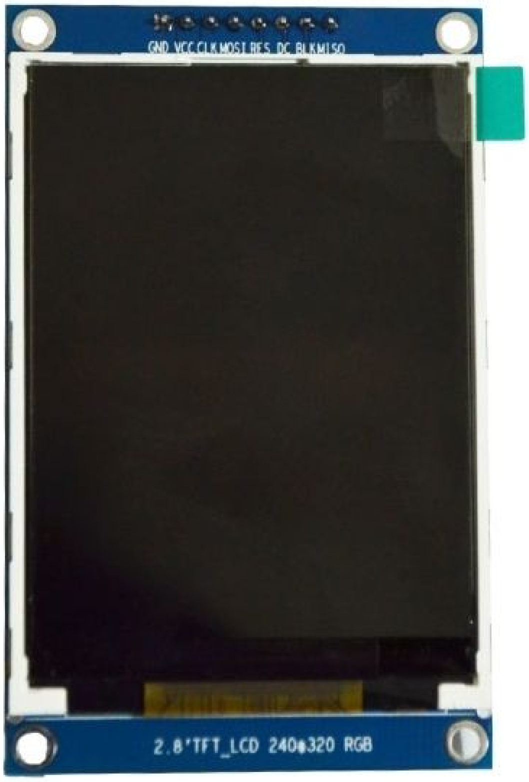
The timing diagram above applies to both the master and and the slave device.

- CPOL - the polarity of the clock
  - CPOL=0, the clock idles at 0 - the leading edge is a rising edge and the trailing edge is a falling edge;
  - CPOL=1, the clock idles at 1 - the leading edge is a falling edge and the trailing edge is a rising edge;
- SS - slave select - slave enabled at 0;
- CPHA - the phase of the clock

- CPHA=0, the "out" side changes the data on the *trailing* edge of the preceding clock cycle, while the "in" side captures the data on (or shortly after) the *leading* edge of the clock cycle;
- CPHA=1, the "out" side changes the data on the *leading* edge of the current clock cycle, while the "in" side captures the data on (or shortly after) the *trailing* edge of the clock cycle.

## Display

The chosen display for this project is a 2.8" LCD display with an ILI9341 controller. Its resolution is 240x320px.



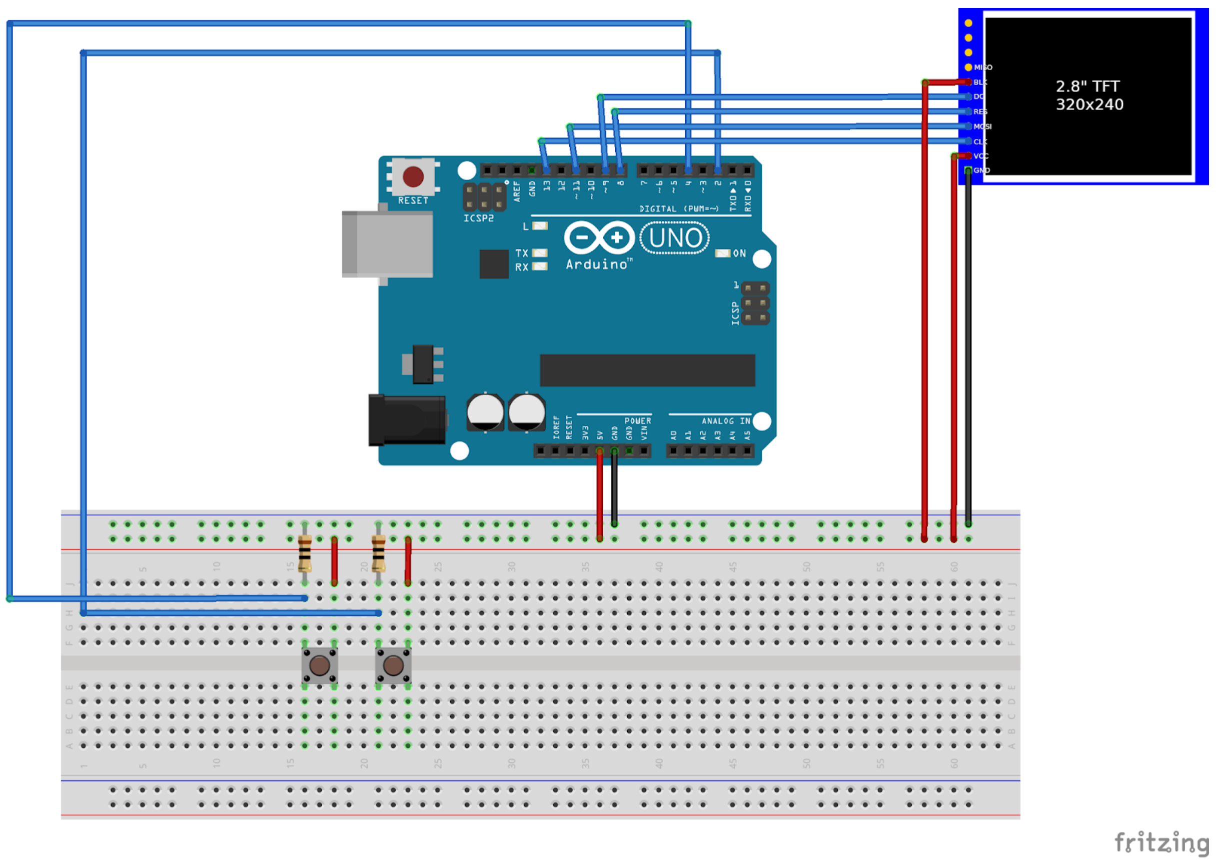
The display communicates with the board through the SPI interface. As such, it has the following ports:

- GND - ground;
- VCC - power;
- CLK - clock;
- MOSI - Master Out Slave In;
- RES - reset;
- DC - Data or Command;
- BLK;
- MISO - Master In Slave Out;

The Ucglib library has been used for communicating with the display.

## Diagrams

### Breadboard



The project is comprised of an Arduino Uno development board, an ILI9341-based LCD display and two pushbuttons.

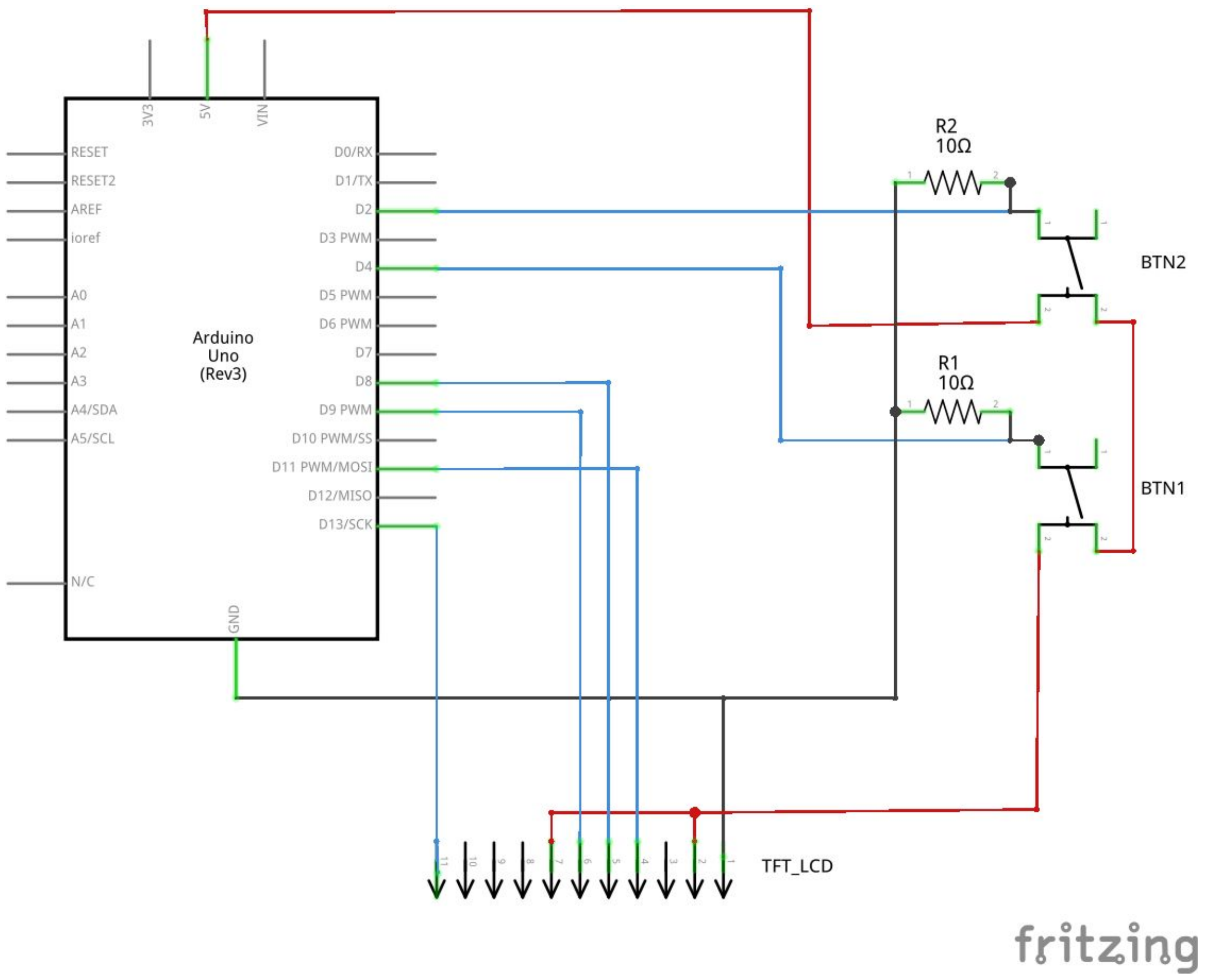
The display has the following connections:

- **GND**: connected to the board's **GND** signal;
- **VCC**: connected to the board's **5V** port;
- **CLK**: connected to port **D13**;
- **MOSI**: connected to port **D11**;
- **RES**: connected to port **D8**;
- **DC**: connected to port **D9**;
- **BLK**: connected to the board's **5V** port;
- **MISO**: disconnected;

The buttons have the following connections:

- **BTN1**: connected to port **D4**;
- **BTN2**: connected to port **D2**;

## Schematic



# Program description

The application starts with a welcome screen which contains the name of the game. The player is then immediately taken to the main game screen, where the game has begun. Using the two hardware buttons, the player can move the position of the paddle to the left and right, in order to prevent the ball from hitting the bottom screen.

At the top of the screen are present three rows of blocks, each row having its own strength:

- top row - blue - requires 3 hits;
- middle row - yellow - requires 2 hits;
- bottom row - red - requires 1 hit.

Once all three rows are cleared, the player is presented with a winning message and the game is restarted. If the ball hits the bottom part of the screen, a game over message displayed and the game is reset.

# Code

## Arkanoid_Arduino.ino

```
#include <SPI.h>
#include "Ucglib.h"
#include "values.h"

// States of the buttons
int readBtnL = 0;
int readBtnR = 0;
int stateBtnL = 0;
int stateBtnR = 0;
int lastStateBtnL = LOW;
int lastStateBtnR = LOW;

// Debounce time for buttons
unsigned long lastDebounceTimeBtnL = 0;
unsigned long lastDebounceTimeBtnR = 0;

Ucglib_ILI9341_18x240x320_HWSPI ucg(/*cd=*/ _pinDisplayCD, /*cs=*/
_pinDisplayCS, /*reset=*/ _pinDisplayRST);

void setup() {
  delay(100);
  Serial.begin(9600);

  // Setup the LCD
  ucg.begin(UCG_FONT_MODE_TRANSPARENT);
  ucg.setFont(ucg_font_ncenR14_hr);
  ucg.clearScreen();

  // Display the welcome message
  reset();
}
// Resets all variables, blocks, ball and paddle to restart game play
void reset() {
  ballX = _initBallX;
  ballY = _initBallY;
  oldBallX = _initBallX;
  oldBallY = _initBallY;
  xincrement = _initXIncrement;
  yincrement = _initXIncrement;
  ballInBlock = false;

  paddleX = _initPaddleX;
  paddleXOld = _initPaddleX;
  score = 0;
  continueGame = true;

  resetBlocks();
  welcome();
}
// Resets all the blocks
void resetBlocks() {
  for(byte i = 0; i < _numBlocks; i++) {
    for(byte param = 0; param < _blockParams; param++) {
      blocks[i][param] = _initBlocks[i][param];
    }
  }
}
// Game play loop
void  loop() {
  readBtnL = digitalRead(_pinBtnL);
  readBtnR = digitalRead(_pinBtnR);

  if (continueGame) {              // If the game is still in play
    drawFrame();                  // Draw the frame
    movePaddle();                 // Update the location of the paddle
    // Determine if the ball has hit the paddle or block
    boolean paddleCollision = checkPaddleCollision();
    ballInBlock = checkBlockCollision();

    // If the score is equivalent to the number of blocks, game is over
    if (score == _numBlocks)
      winner();                   // Display message to user
    else {                        // The game is still in play
      // Redraw screen to draw over any collisions
      if (paddleCollision || ballInBlock)
        drawFrame();
      delay(50);                   // Slight delay
      continueGame = updatePos(); // Update the position of the ball
    }
  }
  else {          // The game is over, the ball fell off the screen.
    gameOver();
  }
}
// Draw the frame with the ball, paddle and blocks
void drawFrame() {
  ucg.setColor(0, 0, 0);
  // Draw over the old ball
  ucg.drawDisc(oldBallX, oldBallY, _ballRad, UCG_DRAW_ALL);
  ucg.setColor(230, 0, 126);
  // Draw the new ball
  ucg.drawDisc(ballX, ballY, _ballRad, UCG_DRAW_ALL);
  ucg.setColor(0, 0, 0);
  // Draw over old paddle
  ucg.drawRBox(paddleXOld, _paddleY, _paddleWidth, _paddleHeight,
_paddleRoundness);

  ucg.setColor(0xff, 0xff, 0xff);
  // Draw new paddle
  ucg.drawRBox(paddleX, _paddleY, _paddleWidth, _paddleHeight,
_paddleRoundness);

  for(int i = 0; i < _numBlocks; i++) {          // Draw the blocks
    if(blocks[i][0] == 1)                        // If still in play
      drawBlock(i);                              // Draw the block
  }
}
// Move the ball
boolean updatePos() {
  // If the ball hits the right or left of the screen
  if (ballX > 240 - _ballRad*2 || ballX < 0 + _ballRad*2)
    xincrement = -xincrement;

  // Ball has hit the bottom. GAME OVER
  if(ballY > 320)
    return false;
  // If the ball is at top of the screen
  else if (ballY < 0 + _ballRad*2)
    yincrement = -yincrement;

  oldBallX = ballX;      // Save the ball's current position
  oldBallY = ballY;

  ballX += xincrement;  // Update the ball's location
  ballY += yincrement;

  return true;
}
// Checks if the ball has hit the paddle
boolean checkPaddleCollision() {
  int ballTop    = ballY - _ballRad;    // Define some values for easy reference
  int ballBottom = ballY + _ballRad;
  int ballLeft   = ballX - _ballRad;
  int ballRight  = ballX + _ballRad;
  int paddleX1   = paddleX + _paddleWidth - 1;
  int middleOffset = 5;

  // If the ball is hitting the TOP of the paddle
  if (ballBottom >= _paddleY) {
    // If the ball has hit in between the left and right edge of paddle
    if (ballX >= paddleX && ballX <= paddleX1) {
      // Determine paddle midpt to determine whether to flip x dir
      float paddleMidPt = (paddleX+_paddleWidth-1)/2;
      yincrement = -yincrement;     // Flip y increment

      // If the ball's xloc is less than the paddle's mid point, flip the x
direction
      if (ballX <= paddleMidPt-middleOffset)
        xincrement = -xincrement;
      return true;
    }
    if (ballRight >= paddleX && ballLeft < paddleX) {
      if (xincrement > 0)           // If it's coming from the left, flip x
        xincrement = -xincrement;
      yincrement = -yincrement;
      return true;
    }
    // If the ball is hitting the RIGHT of the paddle
    else if (ballLeft <= paddleX1 && ballRight > paddleX1) {
      if (xincrement < 0)           // If it's coming from the right, flip x
        xincrement = -xincrement;
      yincrement = -yincrement;
      return true;
    }
  }
  return false;
}
// Checks if the ball has collided with a block
boolean checkBlockCollision() {
  int ballTop = ballY-_ballRad;     // Values for easy reference
  int ballBottom = ballY+_ballRad;
  int ballLeft = ballX-_ballRad;
  int ballRight = ballX+_ballRad;

  for(int i=0; i<_numBlocks; i++) { // Loop through the blocks
       if (blocks[i][0] == 1) {     // If the block hasn't been eliminated
       int blockX = blocks[i][1];   // Grab x and y location
       int blockY = blocks[i][2];

      // If hitting BLOCK
       if (ballBottom >= blockY && ballTop <= blockY+_blockHeight) {
         if (ballRight >= blockX && ballLeft <= blockX+_blockWidth) {
           // Only change y direction if horizontal collision
           if (ballBottom >= blockY && ballTop < blockY)
             yincrement = -yincrement;
          else if (ballTop <= blockY+_blockHeight && ballBottom >
blockY+_blockHeight)
            yincrement = -yincrement;

          // Only change x direction if lateral collision
          if (ballRight >= blockX && ballLeft < blockX && ballX > oldBallX)
            xincrement = -xincrement;
          else if (ballLeft <= blockX+_blockWidth && ballRight >
blockX+_blockWidth && ballX < oldBallX)
            xincrement = -xincrement;

          if (ballInBlock) return true;
          --blocks[i][3];
          drawBlock(i);
          return true;
        }
      }
    }
  }
  return false;  // No collision detected
}

// Draw blocks with the color corresponding to their strength
void drawBlock(int i) {
  switch (blocks[i][3]) {
    case 0:
      removeBlock(i);
      break;
    case 1:
      ucg.setColor(blockColors[0][0], blockColors[0][1], blockColors[0][2]);
      ucg.drawBox(blocks[i][1], blocks[i][2], _blockWidth, _blockHeight);
      break;
    case 2:
      ucg.setColor(blockColors[1][0], blockColors[1][1], blockColors[1][2]);
      ucg.drawBox(blocks[i][1], blocks[i][2], _blockWidth, _blockHeight);
      break;
    case 3:
      ucg.setColor(blockColors[2][0], blockColors[2][1], blockColors[2][2]);
      ucg.drawBox(blocks[i][1], blocks[i][2], _blockWidth, _blockHeight);
      break;
  }
}
// Removes a block from game play
void removeBlock(int index) {
  ucg.setColor(0, 0, 0);
  // Draw the block
  ucg.drawBox(blocks[index][1], blocks[index][2], _blockWidth, _blockHeight);
  blocks[index][0] = 0;        // Mark it as out of play
  score++;                     // Increment score
  adjustSpeed();               // Increment the speed of the ball
}
// Move the paddle
void movePaddle() {
  paddleXOld = paddleX;                 // Save the paddle's old location

  if (readBtnL != lastStateBtnL)        // Reset the debouncing timers
    lastDebounceTimeBtnL = millis();
  if (readBtnR != lastStateBtnR)
```

```
    lastDebounceTimeBtnR = millis();

  if ((millis() - lastDebounceTimeBtnL) > _debounceDelay) {
    if (readBtnL != stateBtnL)  // Check if the left button's state has changed
      stateBtnL = readBtnL;
    if (stateBtnL == HIGH)      // Move the paddle if the new button state is
HIGH
      paddleX = max(0, paddleX - _paddleMove);
  }

  if ((millis() - lastDebounceTimeBtnR) > _debounceDelay) {
    if (readBtnR != stateBtnR)  // Check if the left button's state has changed
    stateBtnR = readBtnR;
    if (stateBtnR == HIGH)      // Move the paddle if the new button state is
HIGH
      paddleX = min(240 - _paddleWidth, paddleX + _paddleMove);
  }

  lastStateBtnL = readBtnL;
  lastStateBtnR = readBtnR;
}
// Increments the speed of the ball
void adjustSpeed() {
  if (yincrement < 0)
    yincrement -= _ballSpeedUp;
  else
    yincrement += _ballSpeedUp;

  if (xincrement < 0)
    xincrement -= _ballSpeedUp;
  else
    xincrement += _ballSpeedUp;
}
// Draws a welcome message before game play begins
void welcome() {
  ucg.setFont(ucg_font_courB14_tf);
  ucg.setColor(255, 255, 255);
  ucg.setPrintPos(15,30);
  ucg.setPrintDir(0);
  ucg.print(_welcomeMessage);
  delay(2000);

  ucg.clearScreen();
  ucg.setFont(ucg_font_helvB18_tr);
  ucg.setColor(230, 0, 126);
  ucg.setPrintPos(55,160);
  ucg.setPrintDir(0);
  ucg.print(_gameTitle);
  delay(1500);
  ucg.clearScreen();
```

```
}
// Tell the player they won
void winner() {
  ucg.clearScreen();
  ucg.setFont(ucg_font_courB14_tf);
  ucg.setColor(255, 255, 255);
  ucg.setPrintPos(80,160);
  ucg.setPrintDir(0);
  ucg.print(_winnerMessage);
  delay(3000);

  ucg.clearScreen();
  reset();
}
// Start a new game
void gameOver() {
  ucg.clearScreen();
  ucg.setFont(ucg_font_courB14_tf);
  ucg.setColor(255, 255, 255);
  ucg.setPrintPos(70,160);
  ucg.setPrintDir(0);
  ucg.print(_gameOverMessage);
  delay(3000);

  ucg.clearScreen();
  reset();
}
```

## values.h

```
#ifndef _H_VALUES
#define _H_VALUES

// Constants
const int _pinBtnL = 4;
const int _pinBtnR = 2;

const int _pinDisplayRST = 8;
const int _pinDisplayCD = 9;
const int _pinDisplayCS = 10;

const unsigned long _debounceDelay = 50; // The debounce time for the buttons

const char *_gameTitle     = "ARKANOID";
const char *_welcomeMessage  = "Get ready to play...";
const char *_winnerMessage    = "You won!";
const char *_gameOverMessage = "Game over!";
const char *_scoreMessage = "Score: ";
const int _ballRad = 6;
const float _ballSpeedUp = 0.09;

const float _initBallX = 20;
const float _initBallY = 250;
const float _initXIncrement = 5;
const float _initYIncrement = 5;
const int _initPaddleX = 64;
const int _paddleY = 310;
const int _paddleWidth = 40;
const int _paddleHeight = 10;
const int _paddleMove = 10;
const int _paddleRoundness = 4;

const int _numBlocks = 12;
const int _blockParams = 4;

const int _blockWidth = 40;
const int _blockHeight = 10;

const short _initBlocks[_numBlocks][_blockParams] = {
//   en,  xloc,   yloc, strength
    {1,   10,      20,    3},
    {1,   70,      20,    3},
    {1,   130,     20,    3},
    {1,   190,     20,    3},
    {1,   10,      40,    2},
    {1,   70,      40,    2},
    {1,   130,     40,    2},
    {1,   190,     40,    2},
    {1,   10,      60,    1},
    {1,   70,      60,    1},
    {1,   130,     60,    1},
    {1,   190,     60,    1}
};

// Variables
float ballX;
float ballY;
float oldBallX;
float oldBallY;
boolean ballInBlock;

float xincrement;
float yincrement;

int paddleX;
int paddleXOld;
int score;
boolean continueGame = true;

unsigned char blockColors[3][3] = {
    {255, 0, 0},   // strength 1
    {255, 255, 0}, // strength 2
    {0, 0, 255}    // strength 3
};

short blocks[_numBlocks][_blockParams] = {
//   en,  xloc,   yloc, strength
    {1,   10,       20,    3},
    {1,   70,       20,    3},
    {1,   130,      20,    3},
    {1,   190,      20,    3},
    {1,   10,       40,    2},
    {1,   70,       40,    2},
    {1,   130,      40,    2},
    {1,   190,      40,    2},
    {1,   10,       60,    1},
    {1,   70,       60,    1},
    {1,   130,      60,    1},
    {1,   190,      60,    1}
};

#endif
```

# Bibliography

1. https://store.arduino.cc/arduino-uno-rev3
2. https://en.wikipedia.org/wiki/ATmega328
3. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
4. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface
5. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface#/media/File:SPI_timing_diagram2.svg
6. https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html
7. https://cdn-shop.adafruit.com/datasheets/ILI9341.pdf
8. https://www.sparkfun.com/datasheets/Components/SMD/ATMega328.pdf
9. https://github.com/olikraus/ucglib