Saving User Settings.

The Arduboy has three different types of memory –  32Kb of memory to store actual programs (PROGMEM) which can only be updated when downloading a new program; 2.5Kb of RAM to hold variables at run time.  This memory is volatile and is lost when you turn the power off; and 1Kb of EEPROM which can be updated within a program and is retained even when the power is turned off.

EEPROM is ideal for saving user settings, high scores and other information between sessions.  EEPROM stands for *El*ectrically Erasable Programmable Read-*O*nly Memory but this is a misnomer as the memory can actually be updated.  EEPROMs have a limited life and will eventually fail after they have been erased and rewritten too many times – this number may be in the millions of operations but a poorly written program that attempts to use it as working memory could easily reach that.

The EEPROM class provides three basic functions to read and write a byte of memory, as shown below.  The memory location can be anywhere in the 1Kb and equates to a value between 0 and 1023.  The update() function differs from the write() function in that it checks the value to be written against what is already stored in order to minimize the number of updates thus prolonging the life of the EEPROM.

EEPROM.read(memory\_location);

EEPROM.update(memory\_location, value);

EEPROM.write(memory\_location, value);

An example application might use functions such as those below:

#define SAVED\_LEVEL 200

byte getSavedLevel() { return EEPROM.read(SAVED\_LEVEL); }

void updateSavedLevel(byte levelNo) { EEPROM.update(SAVED\_LEVEL, levelNo);

The library also offers two other functions that can save and retrieve datatypes other than a byte, such as a float, int or even a structure.

EEPROM.put(memory\_location, value);

EEPROM.get(memory\_location, value);

An example of application might save the current game status using the following functions :

#define SAVED\_STATUS 201

struct GamePlay {

  byte level;

  int score;

  byte board[5][5];

}

gamePlay;

void saveGameStatus(GamePlay gamePlay) {

  EEPROM.put(SAVED\_STATUS, gamePlay);

}

void getGamePlay(GamePlay \*gamePlay) {

  EEPROM.get(SAVED\_STATUS, \*gamePlay);

}

A final note:  the Arduboy2 library uses the first 16 bytes of EEPROM to store settings such as the unit’s name and the current audio settings.  The library also contains a constant, EEPROM\_STORAGE\_SPACE\_START, which defines the starting memory location for user applications to use.  You are free to use the memory above this location anyway you like however to ensure compatibility with this and future versions of the library, you should define the constants used when saving and retrieving EEPROM values relative to the  EEPROM\_STORAGE\_SPACE\_START constant.  An example of this is shown below.

#define SAVED\_LEVEL      EEPROM\_STORAGE\_SPACE\_START

#define SAVED\_STATUS     EEPROM\_STORAGE\_SPACE\_START + 1

More information about the EEPROM functions can be found at the Arduino site (

<https://www.arduino.cc/en/Reference/EEPROM)>

Modulus in C / C++

A modulus (modulo, mod) is simply the remainder when dividing one number by another. When we divide 7 by 2 the result is 3 with 1 remainder. In C / C++ the operand % returns the modulus between two numbers, for example:

Serial.println(7 % 2); >> Outputs ‘1’.

One ‘gotcha’ with the modulus operation is that it returns the modulus using the same sign as the dividend. The sign of the divisor is irrelevant. The examples below shows this in action:

Serial.println(7 % 2); >> Outputs ‘1’.

Serial.println(-7 % 2); >> Outputs ‘-1’.

Serial.println(7 % -2); >> Outputs ‘1’.

Serial.println(-7 % -2); >> Outputs ‘-1’.

Consider the function below. It tests to see if a number is odd by checking the modulus is one. Of course, this will fail if the numberToTest is negative.

bool isAnOddNumber(int numberToTest) {

return numberToTest % 2 == 1;

}

Changing the function to test for a non-zero number - a one or a negative one – accounts for both positive or negative inputs.

bool isAnOddNumber(int numberToTest) {

return numberToTest % 2 != 0;

}