comments inline in red. This file won’t compile and hasn’t been debugged, but should illustrate some concepts that have helped me with LED animations.

General note: you might want to take a look at debouncing as a concept with button reading.

You’re not really up against a memory limit here, this is a small sketch, but if you have a number *variable* and you know it’s between 0-255, you should switch to byte to save.

But this is a special case, as these aren’t really variables, so in theory if you want to prevent trouble later, you might want to assign them as constants, like:

const byte dice2 = 2;

But of you really really want to save on variable space, they are sequential so really you can probably not use variables at all here and just pass the pin numbers. So I’d cut it entirely and use a comment.

# dice pins run from 2-12 and correspond to the value of the roll.

byte buttonPin = 13;

I’d probably change to a Boolean type here, but if nothing else, use byte

byte buttonState = 1;

byte diceroll;

int sleeptime = 1000;

byte rolltime = 30;

byte readtime = 10;

byte curpin;

void setup()

{

# set the dice pin modes with a for loop instead of line after line. I like to use the real start and stop numbers (2-12) so you can see what the code is doing without interpreting weird < edge cases.

for (byte i=2; i<=12; i++) {

pinMode(i,OUTPUT);

}

pinMode(buttonPin, INPUT);

# use randomseed here to properly randomize the generator. https://www.arduino.cc/reference/en/language/functions/random-numbers/randomseed/

randomSeed(analogRead(0));

}

void loop()

{

buttonState = digitalRead(buttonPin);

if (buttonState == LOW)

{

for (byte i = 0; i <= 3; i++) {

#Use a for loop here like above. Have to use new loop var j since you’re inside a loop already using i.

for (byte j=2; i<=12; i++) {

digitalWrite(j, HIGH);

delay(rolltime\*(i+1));

digitalWrite(j, LOW);

}

}

# I’d probably just use a single line here, and remove the randnumber variables altogether:

diceroll = random(1,6) + random(1,6);

#temp var for the roll

if(diceroll != 7) {

# a roll of seven has a special animation that I am not going to touch, so I’ll just ignore it here 😉

# set the current pin at 2.

curpin = 2;

while (curpin <= diceroll) {

flashpin(curpin);

curpin++;

}

# now light up the actual dice roll

digitalWrite(diceroll, HIGH);

}

else {

#dice roll is 7 now… You have a pretty interesting animation here that I don’t feel like rewriting 😉

# if I were to do it I would use the pin%2 trick to get odds and evens, you can pretty easily tell it to light up and turn off only odd/evens using for a modulo math (%2).

digitalWrite(2, HIGH);

digitalWrite(4, HIGH);

digitalWrite(6, HIGH);

digitalWrite(8, HIGH);

digitalWrite(10, HIGH);

digitalWrite(12, HIGH);

delay(sleeptime\*.5);

digitalWrite(2, LOW);

digitalWrite(4, LOW);

digitalWrite(6, LOW);

digitalWrite(8, LOW);

digitalWrite(10, LOW);

digitalWrite(12, LOW);

digitalWrite(3, HIGH);

digitalWrite(5, HIGH);

digitalWrite(7, HIGH);

digitalWrite(9, HIGH);

digitalWrite(11, HIGH);

delay(sleeptime\*.5);

digitalWrite(2, HIGH);

digitalWrite(4, HIGH);

digitalWrite(6, HIGH);

digitalWrite(8, HIGH);

digitalWrite(10, HIGH);

digitalWrite(12, HIGH);

digitalWrite(3, LOW);

digitalWrite(5, LOW);

digitalWrite(7, LOW);

digitalWrite(9, LOW);

digitalWrite(11, LOW);

delay(sleeptime\*.5);

digitalWrite(2, LOW);

digitalWrite(4, LOW);

digitalWrite(6, LOW);

digitalWrite(8, LOW);

digitalWrite(10, LOW);

digitalWrite(12, LOW);

delay(sleeptime\*.1);

digitalWrite(2, HIGH);

delay(sleeptime\*.1);

digitalWrite(3, HIGH);

delay(sleeptime\*.1);

digitalWrite(4, HIGH);

delay(sleeptime\*.1);

digitalWrite(5, HIGH);

delay(sleeptime\*.1);

digitalWrite(6, HIGH);

delay(sleeptime\*.1);

digitalWrite(8, HIGH);

delay(sleeptime\*.1);

digitalWrite(9, HIGH);

delay(sleeptime\*.1);

digitalWrite(10, HIGH);

delay(sleeptime\*.1);

digitalWrite(11, HIGH);

delay(sleeptime\*.1);

digitalWrite(12, HIGH);

delay(sleeptime\*.1);

digitalWrite(7, HIGH);

delay(sleeptime\*.1);

digitalWrite(7, LOW);

delay(sleeptime\*.1);

digitalWrite(7, HIGH);

delay(sleeptime\*.1);

digitalWrite(7, LOW);

delay(sleeptime\*.1);

digitalWrite(7, HIGH);

delay(sleeptime\*.1);

digitalWrite(7, LOW);

delay(sleeptime\*.1);

digitalWrite(7, HIGH);

delay(sleeptime);

digitalWrite(2, LOW);

digitalWrite(3, LOW);

digitalWrite(4, LOW);

digitalWrite(5, LOW);

digitalWrite(6, LOW);

digitalWrite(8, LOW);

digitalWrite(9, LOW);

digitalWrite(10, LOW);

digitalWrite(11, LOW);

digitalWrite(12, LOW);

delay(sleeptime);

digitalWrite(2, HIGH);

digitalWrite(3, HIGH);

digitalWrite(4, HIGH);

digitalWrite(5, HIGH);

digitalWrite(6, HIGH);

digitalWrite(8, HIGH);

digitalWrite(9, HIGH);

digitalWrite(10, HIGH);

digitalWrite(11, HIGH);

digitalWrite(12, HIGH);

delay(sleeptime);

digitalWrite(2, LOW);

digitalWrite(3, LOW);

digitalWrite(4, LOW);

digitalWrite(5, LOW);

digitalWrite(6, LOW);

digitalWrite(8, LOW);

digitalWrite(9, LOW);

digitalWrite(10, LOW);

digitalWrite(11, LOW);

digitalWrite(12, LOW);

delay(sleeptime);

digitalWrite(2, HIGH);

digitalWrite(3, HIGH);

digitalWrite(4, HIGH);

digitalWrite(5, HIGH);

digitalWrite(6, HIGH);

digitalWrite(8, HIGH);

digitalWrite(9, HIGH);

digitalWrite(10, HIGH);

digitalWrite(11, HIGH);

digitalWrite(12, HIGH);

}

// delay(sleeptime);

}

else

{

delay(sleeptime/100);

}

buttonState = digitalRead(buttonPin);

int j = 0;

# this next section appears to be a timer. I would consider using millis() to get an elapsed amount and thus tie it to a real amount of time. This works too, but millis() will be a bit more manageable. Granted, Arduino timer isn’t super accurate, but generally it’s accurate enough for most purposes. I’m not going to change any of it.

while ((buttonState == HIGH) && (j < 400)) {

delay(readtime);

j += 1;

buttonState = digitalRead(buttonPin);

}

#turn all the lights off.

alloff();

}

# two utility functions to turn all the lights off and on.

void alloff() {

for (byte i=2; i<=12; i++) {

digitalWrite(i, LOW);

}

}

void allon() {

for (byte i=2; i<=12; i++) {

digitalWrite(i, HIGH);

}

}

#flashpin is good for flashing one pin at a time. Useful in loops above.

void flashpin(byte whichpin) {

digitalWrite(whichpin, HIGH);

delay(sleeptime);

digitalWrite(whichpin, LOW);

delay(sleeptime);

}

/\*

digitalWrite(2, HIGH);

digitalWrite(3, HIGH);

digitalWrite(4, HIGH);

digitalWrite(5, HIGH);

digitalWrite(6, HIGH);

digitalWrite(7, HIGH);

digitalWrite(8, HIGH);

digitalWrite(9, HIGH);

digitalWrite(10, HIGH);

digitalWrite(11, HIGH);

digitalWrite(12, HIGH);

delay(sleeptime\*100);

\*/