Lab 1

Student name: Dawood Nahhas

Student ID: 2240495

1. Chosen language: Python 3
2. LFSR parameter:
   1. Length: 8
   2. Feedback Polynomial:
   3. Initially seed value: 1469876
3. LFSR Algorithm:
   1. Distribute seed over the registers (S0 ,S1 ,S2 ,…)
   2. Shift registers to the left by the size of the registers
   3. S0i = S3 + S7
   4. Repeat from b.
4. Analyzing the LFSR Algorithm output
   1. The Output: 0100100110
   2. The main output before it gets back to the same output: Couldn't reach the same seed again duo to not have enough memory space
   3. It achieves the the maximum possible period: more than 2^n -1
5. The Algorithm generate a close number for the output and the seed
6. The code:

import sys

# use this library to take arguments

# try and check if the user gave the right input

try:

seed = int(sys.argv[1])

bits = int(sys.argv[2])

taps = [int(sys.argv[3]),int(sys.argv[4])]

repeats = int(sys.argv[5])

# if the bits length is less than 3 it would be fully expected algorithm

if bits < 3:

raise Exception("can't have a bits length less than 3")

except:

sys.exit("please, it can't be less than 3 bits\n[+] usage: lab1.py seed bits\_size s1 s2 repeats")

stat = seed

"""

this variable is for the checking when the algorithm will reach the same seed

to calculate the maximum possible period

"""

for \_ in range(repeats):

output = seed & 1 # take the first bit from the left

print(output,end="")

str\_seed = str(bin(seed))[2:].zfill(bits) # we can't reach the binaries as a list

s1 = int(str\_seed[taps[0]-1])

s2 = int(str\_seed[taps[1]-1])

feedback = s1 ^ s2 # Xor the registers

seed = seed >> 1 # shift the seed with 1 to left

seed = seed | (feedback << bits -1) # add the feedback value ot the first bit from the right

if seed == stat: # check if the algorithm reached the same seed

sys.exit("reached the same seed")

print('\n'+str(seed)) # show teh seed reached

* 1. A screenshot of a video game

     Description automatically generated
  2. The generated sequence:
     1. Using the command line {python lab1.py 1469876 8 7 3 10} the output is : 0010110110
  3. The Algorithm couldn't reach the same seed after repeating for 100000 times
  4. The randomness is not will implemented since the changes in the appearance of 0 and 1 is still expected and the solution to it is to change the seed and make the bits size much higher

1. The new redesigned LFSR will have two Lines that make the output

import sys

def startup():

    try:

        seed = int(sys.argv[1])

        length = int(sys.argv[2])

        taps = [int(sys.argv[3]),int(sys.argv[4])]

        repeats = int(sys.argv[5])

        if length < 3 or taps[0] < 1 or taps[1] >= length:

            sys.exit("length must be more then 3 and the positions must be in range 0 to length")

    except:

        sys.exit("[+] usage: lab1-2.py seed length position1 position2 repeats")

    return seed,length,taps,repeats

class lfsr():

    def \_\_init\_\_(self,seed,length,taps):

        self.seed = seed

        self.length = length

        self.taps = taps

    def lfsr(self,repeats):

        for \_ in range(repeats):

            output = self.seed & 1 # take the first bit from the left

            str\_seed = str(bin(self.seed))[2:].zfill(self.length) # we can't reach the binaries as a list

            s1 = int(str\_seed[self.taps[0]-1])

            s2 = int(str\_seed[self.taps[1]-1])

            feedback = s1 ^ s2 # Xor the registers

            self.seed = self.seed >> 1 # shift the seed with 1 to left

            self.seed = self.seed | (feedback << self.length -1) # add the feedback value ot the first bit from the right

            #print(output,end='')

def generate():

    seed,length,taps,repeats = startup()

    lfsr1 = lfsr(seed,length,taps)

    lfsr1.lfsr(repeats)

    taps = [taps[0]-1,taps[1]+1]

    lfsr2 = lfsr((lfsr1.seed-1)\*2,length,taps)

    for \_ in range(repeats):

        print(bin(lfsr1.seed^lfsr2.seed)[2:],end='')

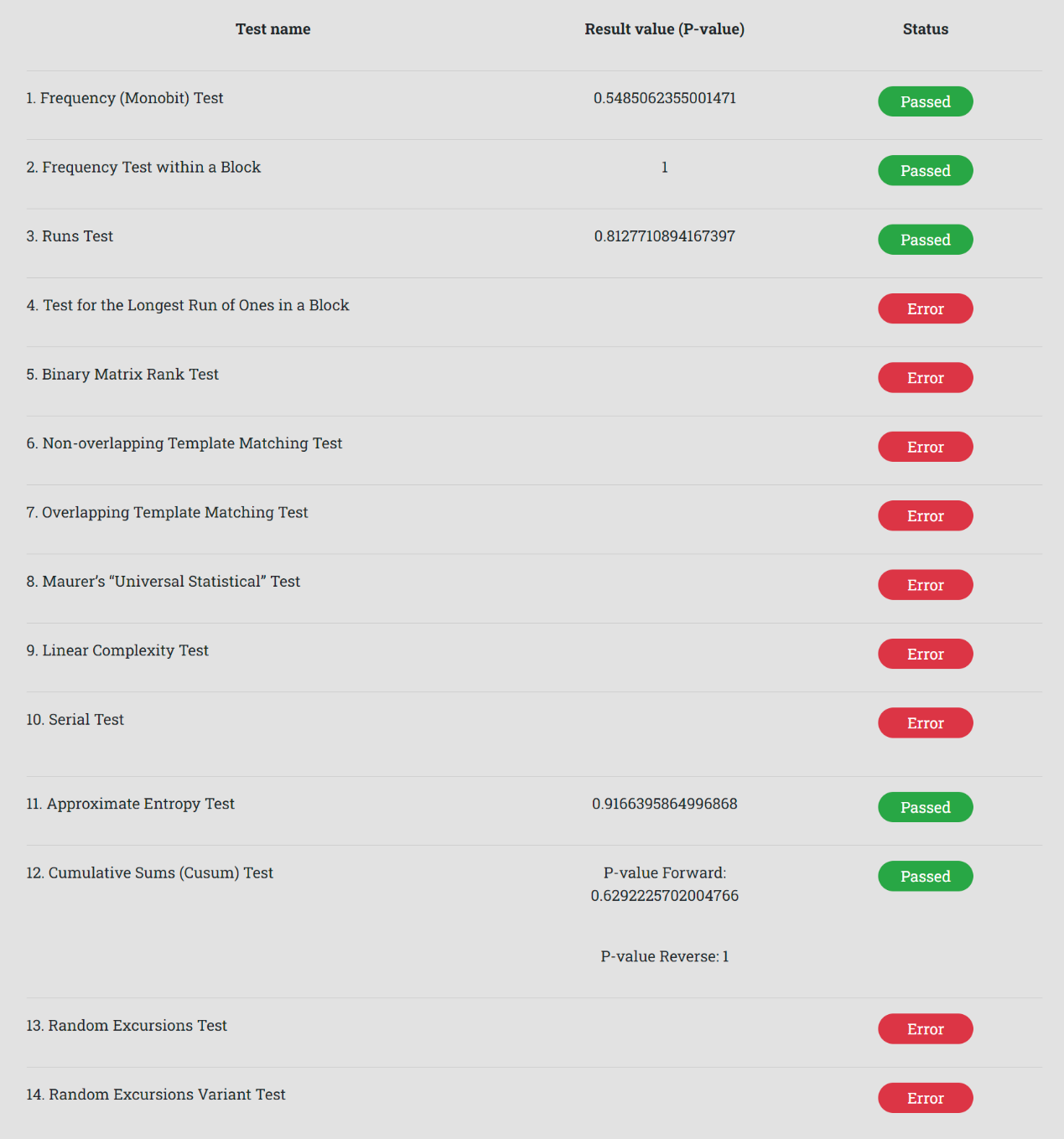
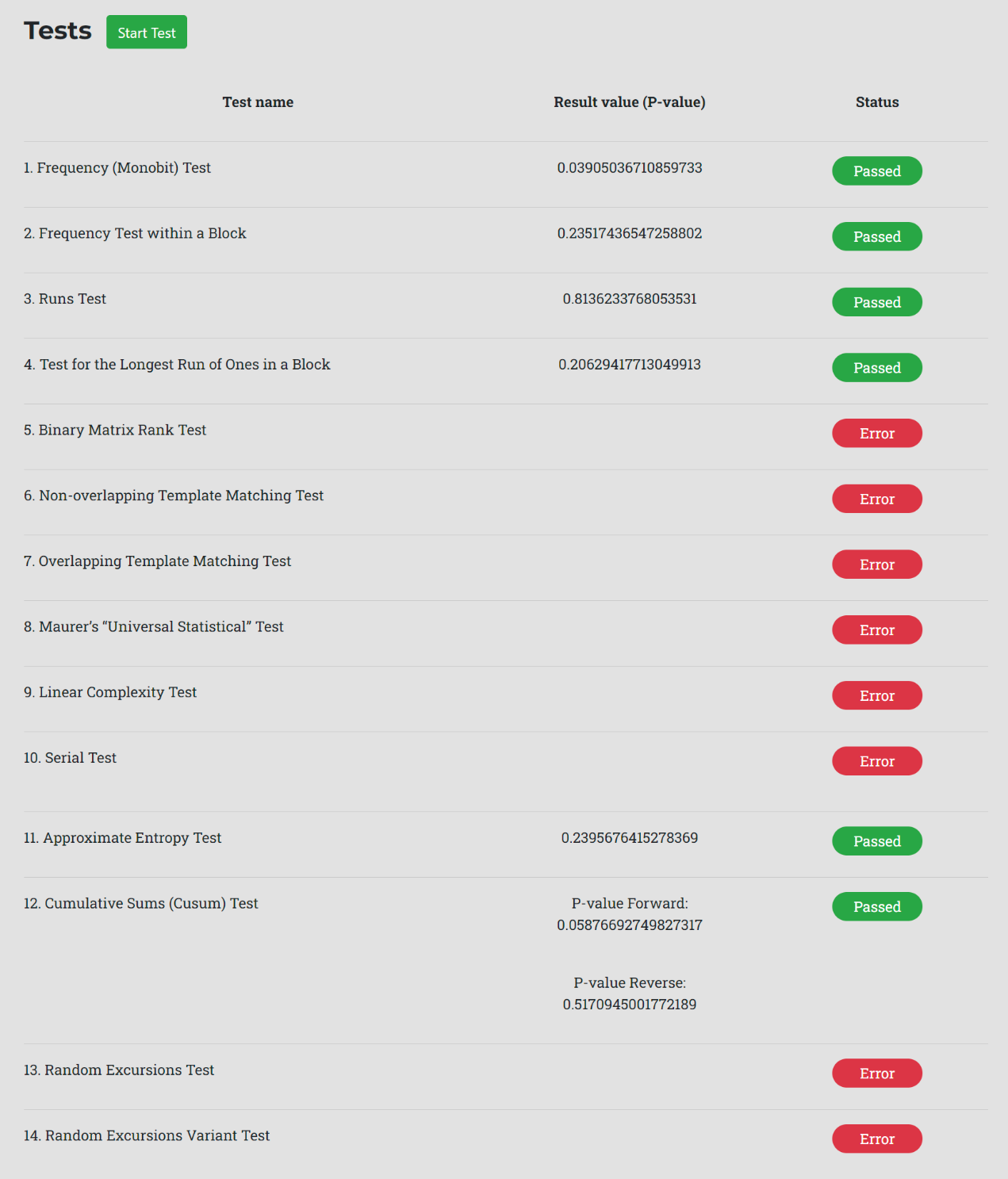
        #print(bin(lfsr1.seed&lfsr2.seed)[2:],end='')

        lfsr1.lfsr(length)

        lfsr2.lfsr(length)

generate()

1. A black and white screen with white squares

   Description automatically generated
2. The equations :
3. Using an online tool called "RANDOM BITSTREAM TESTER"(<https://mzsoltmolnar.github.io/random-bitstream-tester/>), we can analyze the output of both algorithms
   1. The Command line and the arguments: `python .\[file\_name] 146 8 3 7 100`
   2. 100111111101100110000101101100011101100110000110011001100011111100001010000110011011111001111010010111011111000001100100100111010010110110000101111011011000111101011000010111101101011111001111111001111111101101010111100111111000111110110111001111111011110101010100010111001110000011010111001101001101011101011110100100100111111011000101100101111111001011100111001010111110111010110110100000100011011010001011101011011011011001011001111101111101010111001011111011011111010001101011100000011001111111110111001011000111011000111111000011001001111101001000110011011001111010111011011001111101011001001110110011010100111011101101100000011000111000110111101101011010011110101001110110110001111100010110000011111110111000001
   3. The output shows that there's no repetition before the 15 bit
   4. The LFSR test result: 
   5. The NLFSR test result: 
4. The first algorithm depends on one seed but the second algorithm depends on a seed and the result of the seed in a function, this leads to the fact that the second algorithm will be more random than the first algorithm since it depends on two lines of produced numbers