LAB #1 AED

Antonio José Cadavid– A00354484
Juan David Hernández - A00356210
Alejandro Suarez – A00359653

ENGIENEERING METHOD PROYECT REPORT

1. Problem identification

Since the company will improved its security system, cryptography will have a fundamental role in the process, since it is a very secure way to protect data.
In order to implement an effective cryptography-based security system, the company needs a piece of software that generates prime numbers.
They need a software module that generates (n) prime numbers (being ‘n’ the desired amount).

The prime numbers must be displayed on the screen, arranged in a matrix.

Specifications

Name:	R. #1 Generate prime numbers.
Description:	The program must be able to generate (n) prime numbers. It must have three algorithms that can perform this task.
Input:	Amount (n) of prime numbers
Output:	A table bidimensional with de first (n) prime numbers

Name:	R. #2 Get input

LAB #1 AED

Antonio José Cadavid– A00354484
Juan David Hernández - A00356210
Alejandro Suarez – A00359653

Description:	The program must be able to receive the input from the user via a GUI. This input must be an integer (n) >0 and will be used to generate the prime numbers and create a matrix containing them in order to display them on the screen.
Input:	An input (n) that represents the maximum amount of prime numbers that must be generated.
Output:	<None>

Name:	R. #3 Generate Matrix
Description:	The program must generate a matrix containing all the integers from 0 to (n) where (n) is an input given by the user.
Input:	Input (n)
Output:	A matrix containing all the numbers from 0 to (n)

Name:	R. #4 Differeance the primes numbers
Description:	as the algorithm finds that the number is or is not a prime, that is, that allows to show in real time the process performed by the algorithm to find these prime numbers.

LAB #1 AED

Antonio José Cadavid– A00354484
Juan David Hernández - A00356210
Alejandro Suarez – A00359653

Input:	<None>
Output:	Green: Prime numbers
	Red: Numbers not prime

2. Relevant information compilation

Fuente:
https://whatis.techtarget.com/definition/prime-number

https://crypto.stackexchange.com/questions/20867/why-are-primes-important-for-encryption

Prime number:
A prime number is a whole number greater than 1 whose only factors are 1 and itself. A factor is a whole number that can be divided evenly into another number. Numbers that have more than two factors are called composite numbers. The number 1 is neither prime nor composite.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

There are 25 prime numbers between 1 and 100.

Prime numbers and cryptography:
Encryption always follows a fundamental rule: the algorithm or the actual procedure being used doesn't need to be kept secret, but the key does. Even the most sophisticated hacker in the world will be unable to decrypt data as long as the key remains secret and prime numbers are very useful for creating keys. For example, the strength of public/private key encryption lies in the fact that it's easy to calculate the product of two randomly chosen prime numbers, but it can be very difficult and time consuming to determine which two prime numbers were used to create an extremely large product, when only the product is known. This problem is called prime factorization and finding an algorithm which does it fast is one of the unsolved problems of computer science.

LAB #1 AED

Antonio José Cadavid– A00354484
Juan David Hernández - A00356210
Alejandro Suarez – A00359653

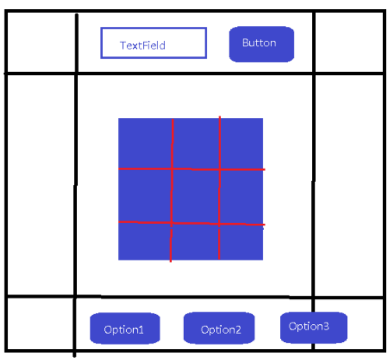
3. Creative Solutions search

We will tackle the problem of generating prime numbers using different approaches.

We will tackle the problem of generating prime numbers and displaying them using different approaches.

Interface of the program:

Alternative 1:
In this approach a single all-in-one interface program was thought.
In the window the scene would be displayed in a BorderPane layout in which on the top part one would have a TextField to introduce the input and generate the matrix, in the bottom part one would have three buttons, each one with a different algorithm to find the prime numbers in the matrix and in the center the matrix would be displayed.



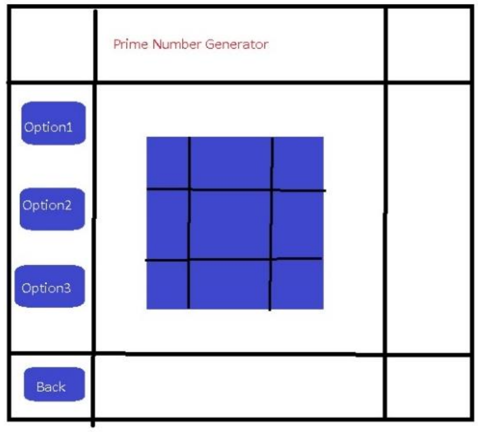
Mockup of the interface

Alternative2:
In this approach we thought about a two scene interface.
In the first scene there is a Textfield (to get the user input) and a Button (to generate the matrix).
Once the user generates the matrix the scene changes and now there is a BorderPane layout in wich on the left side there are four butons, three for filling up the matrix with primes and one for going back. In the center of the pane the matrix is displayed.

LAB #1 AED

Antonio José Cadavid– A00354484
Juan David Hernández - A00356210
Alejandro Suarez – A00359653

Mockup scene 1



Mockup scene 2

-Sieve of Atkin:

```java
public ArrayList<Integer> SieveOfAtkin(int limit) {

    ArrayList<Integer> primes = new ArrayList<Integer>();

    if (limit > 2)
        primes.add(2);

    if (limit > 3)
        primes.add(3);

    boolean sieve[] = new boolean[limit];

    for (int i = 0; i < limit; i++)
        sieve[i] = false;

    for (int x = 1; x * x < limit; x++) {
        for (int y = 1; y * y < limit; y++) {

            int n = (4 * x * x) + (y * y);
            if (n <= limit && (n % 12 == 1 || n % 12 == 5))
                sieve[n] ^= true;

            n = (3 * x * x) + (y * y);
            if (n <= limit && n % 12 == 7)
                sieve[n] ^= true;

            n = (3 * x * x) - (y * y);
            if (x > y && n <= limit && n % 12 == 11)
                sieve[n] ^= true;
        }
    }

    // Mark all multiples of squares as
    // non-prime
    for (int r = 5; r * r < limit; r++) {
        if (sieve[r]) {
            for (int i = r * r; i < limit; i += r * r)
                sieve[i] = false;
        }
    }

    // Print primes using sieve[]
    for (int a = 5; a < limit; a++)
        if (sieve[a])
            primes.add(a);

    return primes;
}
```

LAB #1 AED

Antonio José Cadavid– A00354484
Juan David Hernández - A00356210
Alejandro Suarez – A00359653

- This algorithm adds numbers 2 and 3 to the array list if the input is bigger than them.
- Then creates an array of Boolean and put some indexes like true.
- In the square indices, there are indicated as false
- All indexes indicated as true are added to the array list

-Sieve of Betwise:

```java
    public ArrayList<Integer> betwiseSieve(int n) {

        ArrayList<Integer> primes = new ArrayList<Integer>();

        // prime[i] is going to store true if
        // if i*2 + 1 is composite.
        boolean prime[] = new boolean[n / 2];
        Arrays.fill(prime, false);

        // 2 is the only even prime so we can
        // ignore that. Loop starts from 3.
        for (int i = 3; i * i < n; i += 2) {
            // If i is prime, mark all its
            // multiples as composite
            if (prime[i / 2] == false)
                for (int j = i * i; j < n; j += i * 2)
                    prime[j / 2] = true;
        }

        // writing 2 separately
        primes.add(2);

        // Printing other primes
        for (int i = 3; i < n; i += 2)
            if (prime[i / 2] == false)
                primes.add(i);

        return primes;
    }
```

- This method fills an array of Boolean with false
- Add the number 2 to the array
- If find a prime number mark all its multiples as false
- Add primes to an array list

LAB #1 AED

Antonio José Cadavid– A00354484
Juan David Hernández - A00356210
Alejandro Suarez – A00359653

-Sieve of Eratosthenes:

```java
    public ArrayList<Integer> sieveOfEratosthenes(int n) {

        ArrayList<Integer> primes = new ArrayList<Integer>();

        // Create a boolean array "prime[0..n]" and initialize
        // all entries it as true. A value in prime[i] will
        // finally be false if i is Not a prime, else true.
        boolean prime[] = new boolean[n + 1];
        for (int i = 0; i < n; i++)
            prime[i] = true;

        for (int p = 2; p * p <= n; p++) {
            // If prime[p] is not changed, then it is a prime
            if (prime[p] == true) {
                // Update all multiples of p
                for (int i = p * p; i <= n; i += p)
                    prime[i] = false;
            }
        }

        // Print all prime numbers
        for (int i = 2; i <= n; i++) {
            if (prime[i] == true)
                primes.add(i);
        }
        return primes;
    }
```

- This method fills an array of Boolean with true
- Indicates squares of prime numbers as false
- Add all primes to an array list

-Simple methods to find prime numbers: 1.

LAB #1 AED

Antonio José Cadavid– A00354484
Juan David Hernández - A00356210
Alejandro Suarez – A00359653

```java
	public boolean isPrime(int n) {

		if (n <= 1)
			return false;
		if (n <= 3)
			return true;

		if (n % 2 == 0 || n % 3 == 0)
			return false;

		for (int i = 5; i * i <= n; i = i + 6) {
			if (n % i == 0 || n % (i + 2) == 0) {
				return false;
			}
		}
		return true;
	}

	public ArrayList<Integer> primes(int n) {

		ArrayList<Integer> primes = new ArrayList<Integer>();

		for (int i = 2; i <= n; i++) {
			if (isPrime(n))
				primes.add(i);
		}

		return primes;
	}
```

- This algorithm search primes looking each number since 2 to n
- To know if a number is a prime first look if is divisible by 2 and 3 and if it’s the square of another prime number
- If the number is prime its added to an array list

2.

```java
	public boolean isPrime2(int n) {
		// Corner case
		if (n <= 1)
			return false;

		// Check from 2 to n-1
		for (int i = 2; i < n; i++)
			if (n % i == 0)
				return false;

		return true;
	}

	public ArrayList<Integer> printPrime(int n) {

		ArrayList<Integer> prime = new ArrayList<Integer>();

		for (int i = 2; i <= n; i++) {
			if (isPrime2(i))
				prime.add(i);
		}
		return prime;
	}
```

- This algorithm search primes looking each number since 2 to n

LAB #1 AED

Antonio José Cadavid– A00354484
Juan David Hernández - A00356210
Alejandro Suarez – A00359653

- To know if a number is a prime look if the number is divisible by another number since 2 to n
- If the number is prime its added to an array list

-Segmented sieve:

LAB #1 AED

Antonio José Cadavid– A00354484
Juan David Hernández - A00356210
Alejandro Suarez – A00359653

```java
public void simpleSieve(int limit, ArrayList<Integer> prime) {

    boolean mark[] = new boolean[limit + 1];

    for (int i = 0; i < mark.length; i++)
        mark[i] = true;

    for (int p = 2; p * p < limit; p++) {
        // If p is not changed, then it is a prime
        if (mark[p] == true) {
            // Update all multiples of p
            for (int i = p * 2; i < limit; i += p)
                mark[i] = false;
        }
    }

    for (int p = 2; p < limit; p++) {
        if (mark[p] == true) {
            prime.add(p);
        }
    }
}

public ArrayList<Integer> segmentedSieve(int n) {

    int limit = (int) (floor(sqrt(n)) + 1);
    ArrayList<Integer> prime = new ArrayList<Integer>();
    simpleSieve(limit, prime);

    int low = limit;
    int high = 2 * limit;

    while (low < n) {
        if (high >= n)
            high = n;

        boolean mark[] = new boolean[limit + 1];

        for (int i = 0; i < mark.length; i++)
            mark[i] = true;

        for (int i = 0; i < prime.size(); i++) {

            int loLim = (int) (floor(low / prime.get(i)) * prime.get(i));
            if (loLim < low)
                loLim += prime.get(i);

            for (int j = loLim; j < high; j += prime.get(i))
                mark[j - low] = false;
        }

        for (int i = low; i < high; i++)
            if (mark[i - low] == true)
                prime.add(i);

        low = low + limit;
        high = high + limit;
    }
    return prime;
}
```

- This algorithm takes the square root of the input and find all primes since 2 to that number with the simple sieve and add the numbers to an array list
- Then find more prime numbers with the primes found before with an array of Boolean

LAB #1 AED

Antonio José Cadavid– A00354484
Juan David Hernández - A00356210
Alejandro Suarez – A00359653

-Sieve of Sundaram:

```java
    public ArrayList<Integer> SieveOfSundaram(int n) {

        ArrayList<Integer> prime = new ArrayList<Integer>();

        // In general Sieve of Sundaram, produces
        // primes smaller than (2*x + 2) for a number
        // given number x. Since we want primes
        // smaller than n, we reduce n to half
        int nNew = (n - 2) / 2;

        // This array is used to separate numbers of the
        // form i+j+2ij from others where 1 <= i <= j
        boolean marked[] = new boolean[nNew + 1];

        // Initialize all elements as not marked
        Arrays.fill(marked, false);

        // Main logic of Sundaram. Mark all numbers of the
        // form i + j + 2ij as true where 1 <= i <= j
        for (int i = 1; i <= nNew; i++)
            for (int j = i; (i + j + 2 * i * j) <= nNew; j++)
                marked[i + j + 2 * i * j] = true;

        // Since 2 is a prime number
        if (n > 2)
            prime.add(2);

        // Print other primes. Remaining primes are of
        // the form 2*i + 1 such that marked[i] is false.
        for (int i = 1; i <= nNew; i++)
            if (marked[i] == false)
                prime.add(2 * i + 1);
        return prime;
    }
```

- This algorithm reduces the input by 2, then divides that number by 2 and creates an array of Boolean filled with false.
- Then marks some indexes that are not prime with true.
- If find an index with false, add the resulting number of (2 * index + 1) to an array list

4. **Transition of ideas formulation to preliminary designs**

   In this phase we're going to discard some solutions from the previous phase. Solutions that will be discard are the following:

   - Sieve of Atkin
   - Simple method to find prime numbers (1 and 2)
   - Segmented sieve

   The reason why we're going to discard these solutions is because are not very efficient and a bit unstable.

5. **Evaluation and selection of the best solution**

LAB #1 AED

Antonio José Cadavid– A00354484
Juan David Hernández - A00356210
Alejandro Suarez – A00359653

Evaluation criteria #1: Efficiency, that is, the number of lines required to reach the solution.

1) Give a solution by executing a ridiculous amount of lines.
2) Give a solution by running many lines.
3) Give a solution by running few lines

Evaluation criteria #2: Code decoupling.

1) Very coupled, hard to reuse
2) Uncoupled, can be reused in other solutions

	Evaluation criteria #1	Evaluation criteria #2	Total
Sieve of Atkin	1	1	2
Sieve of Betwise	3	2	5
Sieve of Eratosthenes	3	2	5
Simple method #1	2	1	3
Simple method #2	1	1	2
Segmented sieve	1	1	2
Sieve of Sundaram	3	2	5