GENERADOR DE NÚMEROS ALEATORIOS

SIMULACIÓN TP1

GRUPO 3

INTEGRANTES

1. VÍCTOR ZACARÍAS
2. LEONEL GONZALEZ
3. ALEXANDER VILLA
4. KEVIN SPASIUK

CONTENIDO

PASO 1

Partimos de un generador congruente lineal

PASO 2

Xorshift aleatorio para mejorar los bits centrales

PASO 3

Multiplicación para mejorar bits superiores

PASO 4

Xorshift para mejorar los bits inferiores

PCG-RXS-M-XS

Implementamos el generador PCG variante PCG-RXS-M-XS Genera 32 bits de aleatoriedad a partir de 32 bits de estado.

PARÁMETROS DEL GENERADOR

```
#Parametros generales del generador
seed = 10
m = 2**32
a = 1664525
c = 1013904223
n = 1000000
```

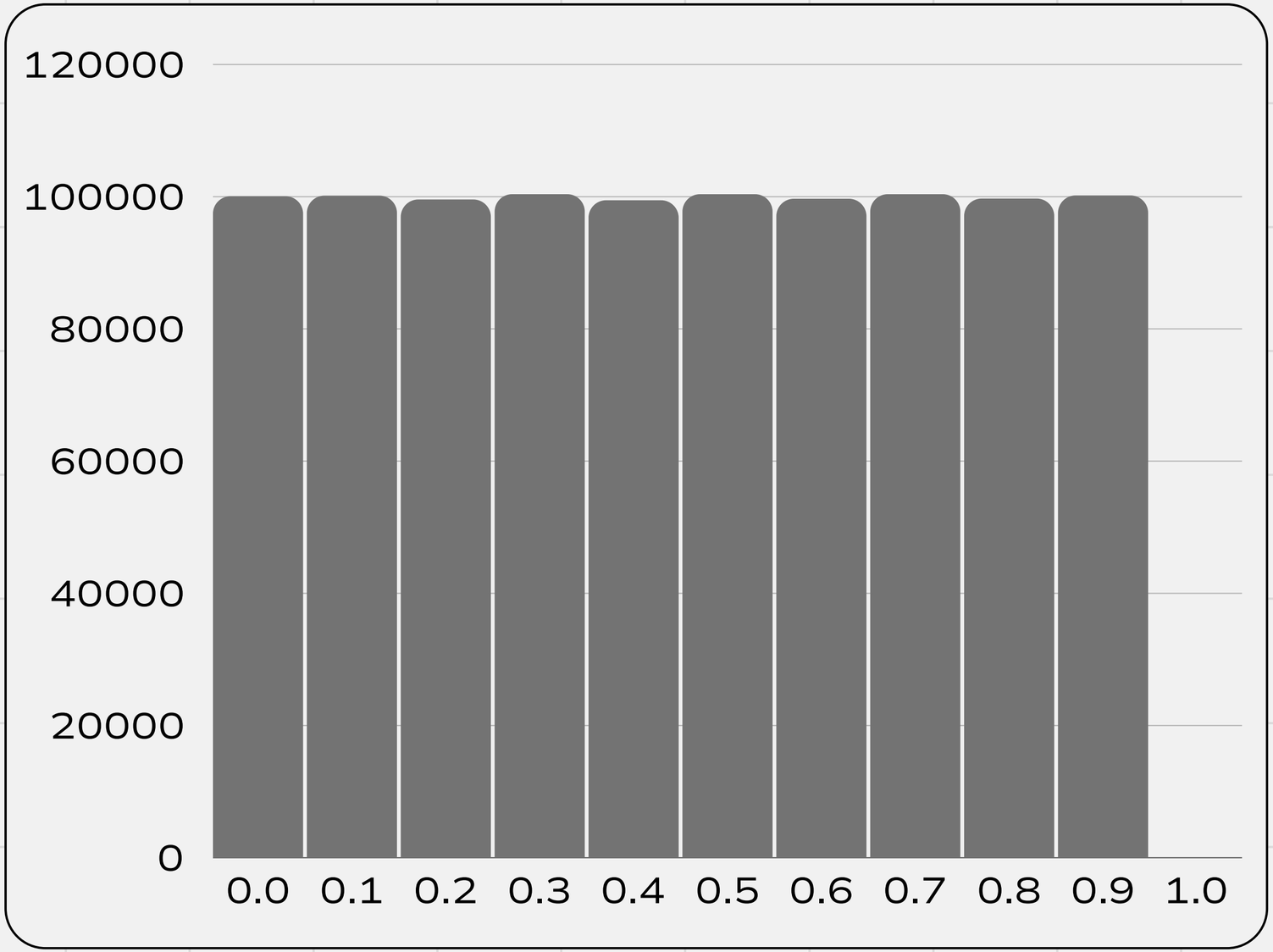
ALGORITMO

```python
def generar_lote_de_numeros_aleatorios(n, seed):
    numeros = []
    x = seed
    for _ in range(n):
        x = (a * x + c) % m
        pcg_num = pcg_rxs_m_xs(x) / float(2**32)  # Escalar a [0, 1) con 2^32
        numeros.append(pcg_num)

    return numeros
```

$$f_c(n) = ((n \twoheadrightarrow c) \boxtimes r) \oplus (c \triangleleft (r - c)),$$

```python
def pcg_rxs_m_xs(n):
    '''
    #Paso 1: random xorshift
    Obtenemos los primeros 3 bits del state:
    Del paper:
    We will use the top three bits to choose members of a family of eight permutations,
    but this time the permutation is an xorshift
    we will use the top t bits two ways at once—we will use them to both define
    how much of an xorshift to apply and also to be part of the xorshift itself
    '''
    c = primeros_tres_bits(n)
    xorshifted = n ^ (n >> c)

    '''
    defino un r = 17 (debe pertenecera 2^32-c)
    '''
    r= 17
    xorshifted ^= (xorshifted << r)

    ''' ultimo xorsfhit de
    (c≪(r-c))
    '''
    xorshifted ^= (xorshifted >> (r - c))     # Tercer XOR-shift

    '''
    #Paso 2: Multiplicación para mejorar los bits superiores
    '''
    multiplied = (xorshifted * a + c) & 0xFFFFFFFF  #nos aseguramos que sea de 32 bits

    '''
    # Paso 3:  xorshift para mejorar los bits inferiores
    '''
    result = multiplied ^ (multiplied >> 15)
    return result
```

HISTOGRAMA


120000
100000
80000
60000
40000
20000
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

TEST ESTADÍSTICOS

CHI^2

KOLMOGOROV SMIRNOV

INDEPENDENCIA

CHI^2 PROPIO

```
sumado= 0
for elem in bin_count:
    sumado += ((elem-100000)**2)/100000

grados_libertad= 9
limite = chi2.ppf(0.95, df=grados_libertad)

print("D^2 = ", sumado)
print("limite = ", limite)
print('No rechazamos H0' if sumado <= limite   else 'Rechazamos H0')
```

```
D^2 =  11.454040000000003
LIMITE =  16.918977604620448
NO RECHAZAMOS H0
```

CHI^2

```python
frecuencias = bin_count
tamanio = len(bin_count)
lanzamientos = bin_count.sum()
grados_libertad= tamanio - 1

D2 = sum([(fO - 1/10 *lanzamientos)**2 for fO in
frecuencias])/(1/10*lanzamientos)
limiteSuperior = chi2.ppf(0.95, df=grados_libertad)

print("D^2 = ", D2)
print("Estadistico: {:.2f} ".format(D2))
print("Limite superior: {:.2f} ".format(limiteSuperior))

if D2 <= limiteSuperior:
    print("No rechazamos H0")
else:
    print("Rechazamos H0")
```

D^2 = 11.45404 ESTADISTICO: 11.45
LIMITE SUPERIOR: 16.92
NO RECHAZAMOS H0

KOLMOGOROV SMIRNOV

```python
ks_stat, p_value = stats.kstest(numeros, 'uniform')

print(f'Estadístico de Kolmogorov-Smirnov: {ks_stat}')
print(f'Valor p: {p_value}')

print('No rechazamos H0' if p_value > 0.01 else 'Rechazamos H0')
```

ESTADÍSTICO DE KOLMOGOROV-SMIRNOV: 0.0005864273765534
VALOR P: 0.8815203789694045
NO RECHAZAMOS H0

INDEPENDENCIA

Resultado chi-cuadrado: 450.0528
Valor p: 1.0000
Degrees of Freedom: 2499
Limit: 2616.4113

NO HAY EVIDENCIA SUFICIENTE PARA RECHAZAR H0

NO HAY UNA RELACIÓN SIGNIFICATIVA ENTRE LAS VARIABLES.

```
#genero 2 lotes de 2500 numeros
lote_x = numeros[0:2500]
lote_y = generar_lote_de_numeros_aleatorios(2500, seed * 25)

numeros_x = [truncar_a_dos_decimales(elem) for elem in lote_x]
numeros_y = [truncar_a_dos_decimales(elem)for elem in lote_y]

chi2v, p_val, dof, ex = chi2_contingency([numeros_x, numeros_y],
correction=True)

print(f'Resultado de prueba de Estadístico chi-cuadrado:
{chi2v:.4f}')
print(f'Valor p: {p_val:.4f}')
print(f'Degrees of Freedom: {dof}')

#usamos un nivel de significacion del 5%
alpha = 0.05

#calculo limite supoerior para aceptar/rechazar hipotesis
limiteSuperior = chi2.ppf(1 - alpha, df=dof)

print(f'Limit: {limiteSuperior:.4f}')

#comparo resultado de estadistico contra limite
if chi2v > limiteSuperior:
    print("Rechazamos H0: Existe una relación significativa entre
las variables, por ende NO son independientes")
else:
    print("No hay evidencia suficiente para rechazar H0, no hay
una relación significativa entre las variables.")
```

TEST ESTADÍSTICOS SOBRE GENERADOR GRUPO 4

	CHI^2	KOLGOMOROV	INDEPENDENCIA
TIRA 1	D^2 = 22.27154 ESTADISTICO: 22.27 LIMITE SUPERIOR: 16.92 **RECHAZAMOS H0**	ESTADÍSTICO: 0.001117 VALOR P: 0.16460 **NO RECHAZAMOS H0**	ESTADÍSTICO CHI-CUADRADO: 18.7171 VALOR P: 1.0000 DEGREES OF FREEDOM: 99 LIMIT: 81.4493 **NO HAY EVIDENCIA SUFICIENTE PARA RECHAZAR H0**
TIRA 2	D^2 = 15.10242 ESTADISTICO: 15.10 LIMITE SUPERIOR: 16.92 **NO RECHAZAMOS H0**	ESTADÍSTICO: 0.001017 VALOR P: 0.251053 **NO RECHAZAMOS H0**	

CONCLUSIONES DE LOS TESTS

- Rechazo del test de chi cuadrado de una tira de números. Motivos: semilla o parámetros internos.

- Test de independencia no rechazan las hipótesis nula. Si aumentamos alfa, y reducimos la cantdad de muestras por tira, se mantiene la tendencia

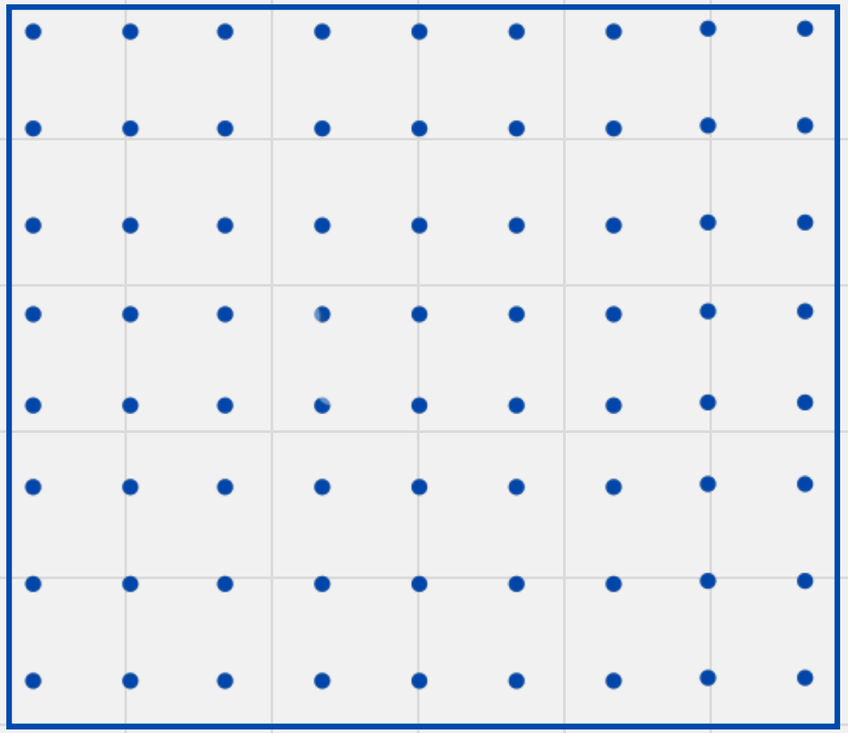
- El test de Kolmogorov-Smirnov no rechaza la hipótesis nula.

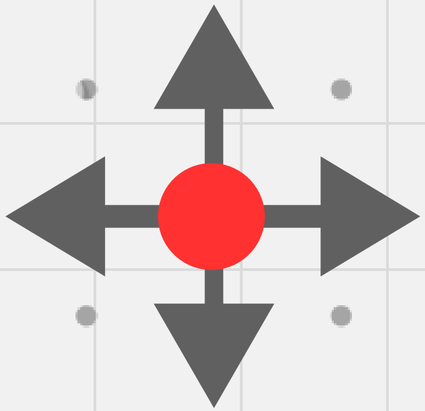
Si comparamos el p valor de los tres test:

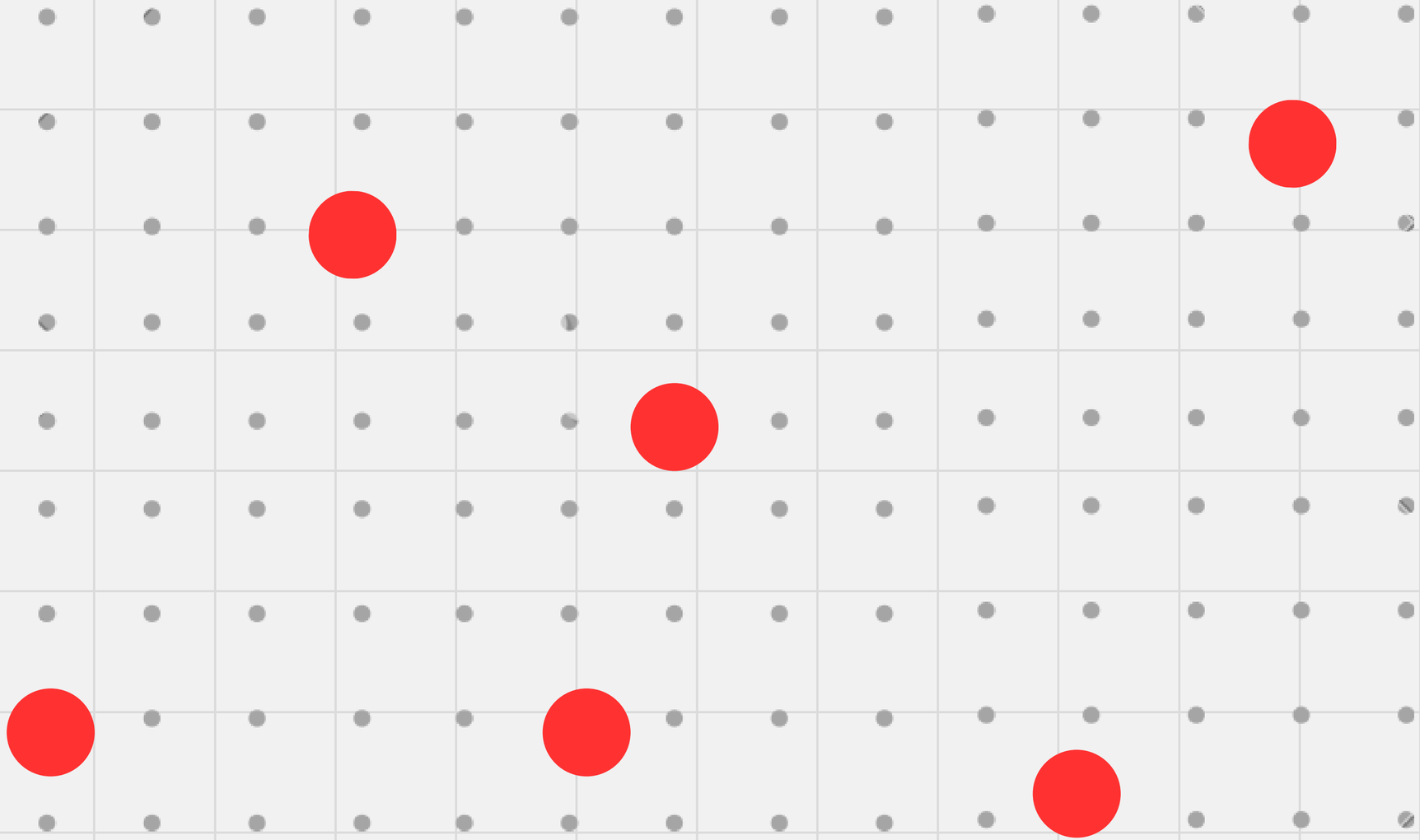
- numeros grupo 3: 0.8815203789694045
- numeros grupo4_1: 0.1646010943503009
- numeros grupo4_2: 0.2510538902300188

Vemos que los números del grupo 4 están más cerca del límite.

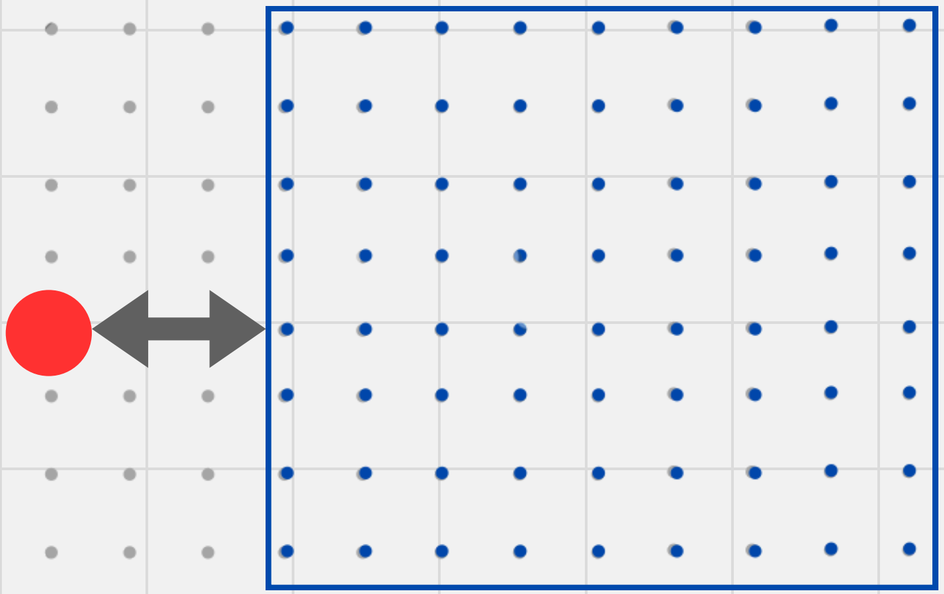
SIMULACIÓN MOVIMIENTO BROWNIANO



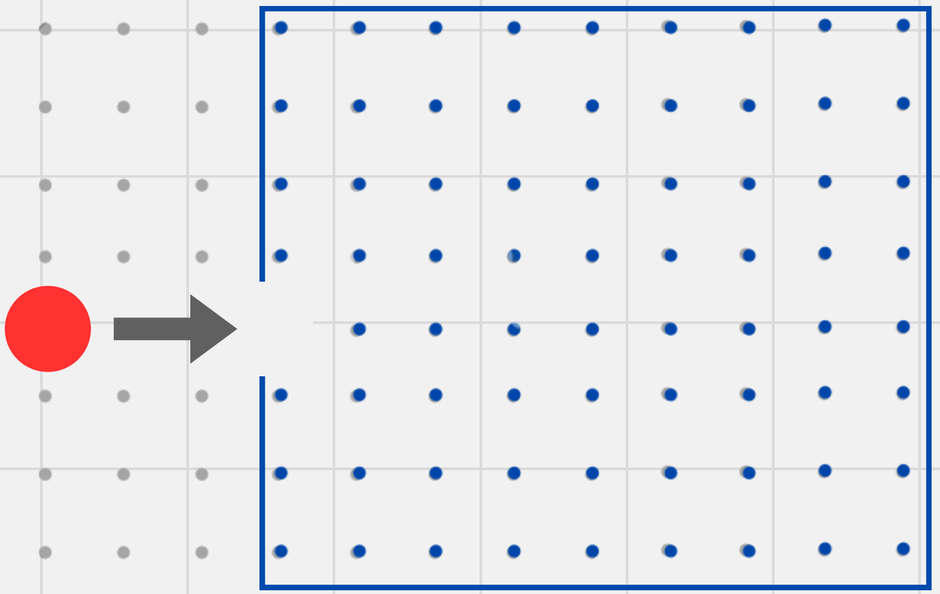




REBOTE



ABSORCIÓN



IMPLEMENTACIÓN

DIRECCIÓN

```
ARRIBA = -1
ABAJO = 1
IZQUIERDA = -1
DERECHA = 1

MOVIMIENTO_HORIZONTAL = "MOVIMIENTO_HORIZONTAL"
MOVIMIENTO_VERTICAL = "MOVIMIENTO_VERTICAL"


def determinar_movimiento_horizontal(semilla):
    numero_aleatorio = generar_lote_de_numeros_aleatorios(1, semilla)[0]

    if numero_aleatorio <= 0.5:
        return IZQUIERDA
    else:
        return DERECHA


def determinar_movimiento_vertical(semilla):
    numero_aleatorio = generar_lote_de_numeros_aleatorios(1, semilla)[0]

    if numero_aleatorio <= 0.5:
        return ARRIBA
    else:
        return ABAJO


def determinar_tipo_de_movimiento(semilla):
    numero_aleatorio = generar_lote_de_numeros_aleatorios(1, semilla)[0]

    if numero_aleatorio <= 0.5:
        return MOVIMIENTO_HORIZONTAL
    else
        return MOVIMIENTO_VERTICAL
```

IMPLEMENTACIÓN
SELECCIÓN PROXIMA POSICIÓN

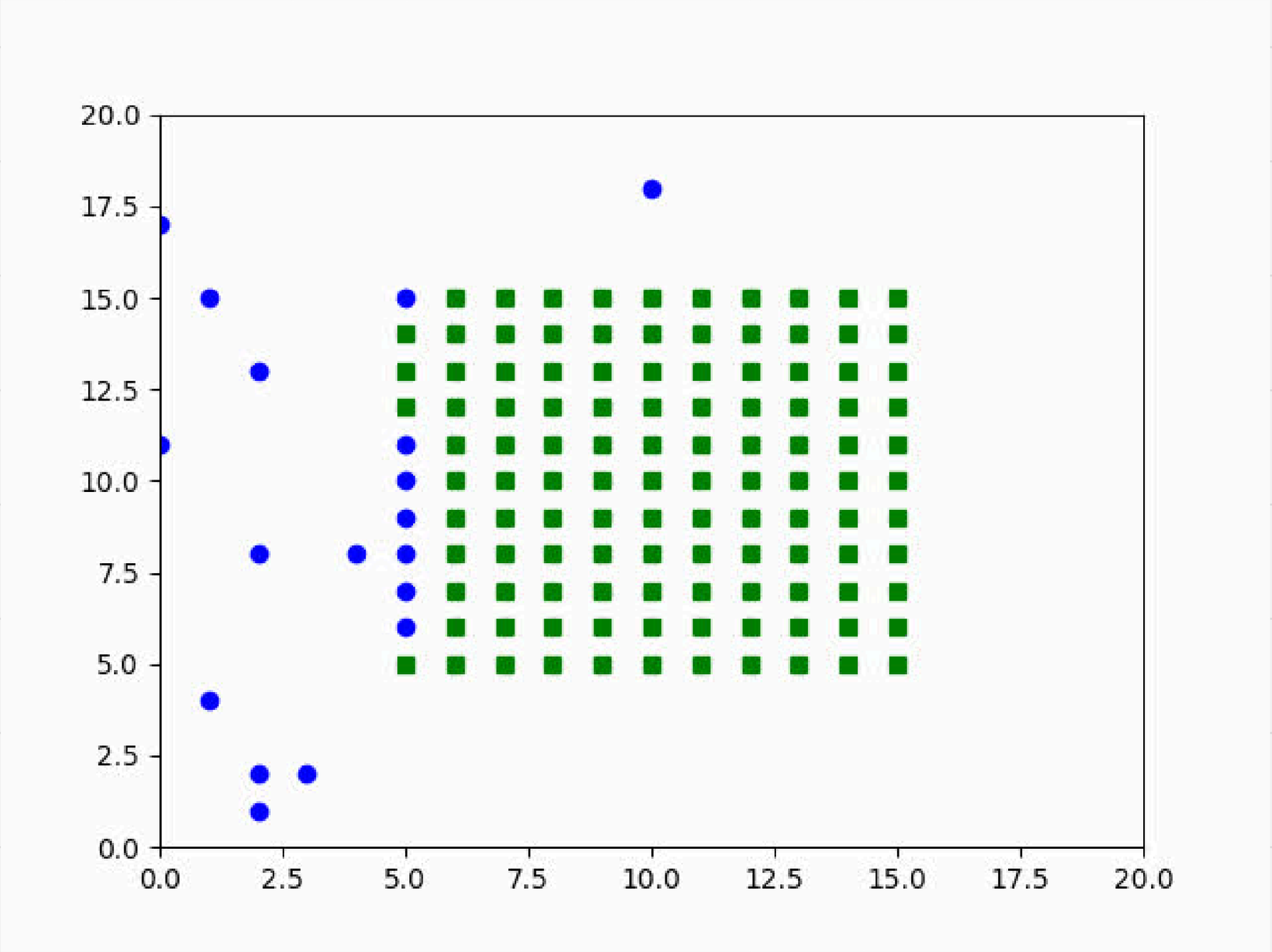
```python
def esta_dentro_de_los_limites(nueva_posicion, ancho, alto):
    return (nueva_posicion[0] >= 0
            and nueva_posicion[0] < ancho)
            and (nueva_posicion[1] >= 0
            and nueva_posicion[1] < alto)
```

```python
proxima_posicion =
    (particula["pos_actual"][0] + movimiento_horizontal,
    particula["pos_actual"][1] + movimiento_vertical)
if esta_dentro_de_los_limites(proxima_posicion, ancho_grilla, alto_grilla)
    and grilla[proxima_posicion[0]][proxima_posicion[1]] != PARTICULA:
        es_pos_valida = True
        grilla[particula["pos_actual"][0]][particula["pos_actual"][1]] = particula["estado_anterior"]
        particula["estado_anterior"] = grilla[proxima_posicion[0]][proxima_posicion[1]]
        particula["pos_actual"] = proxima_posicion
```

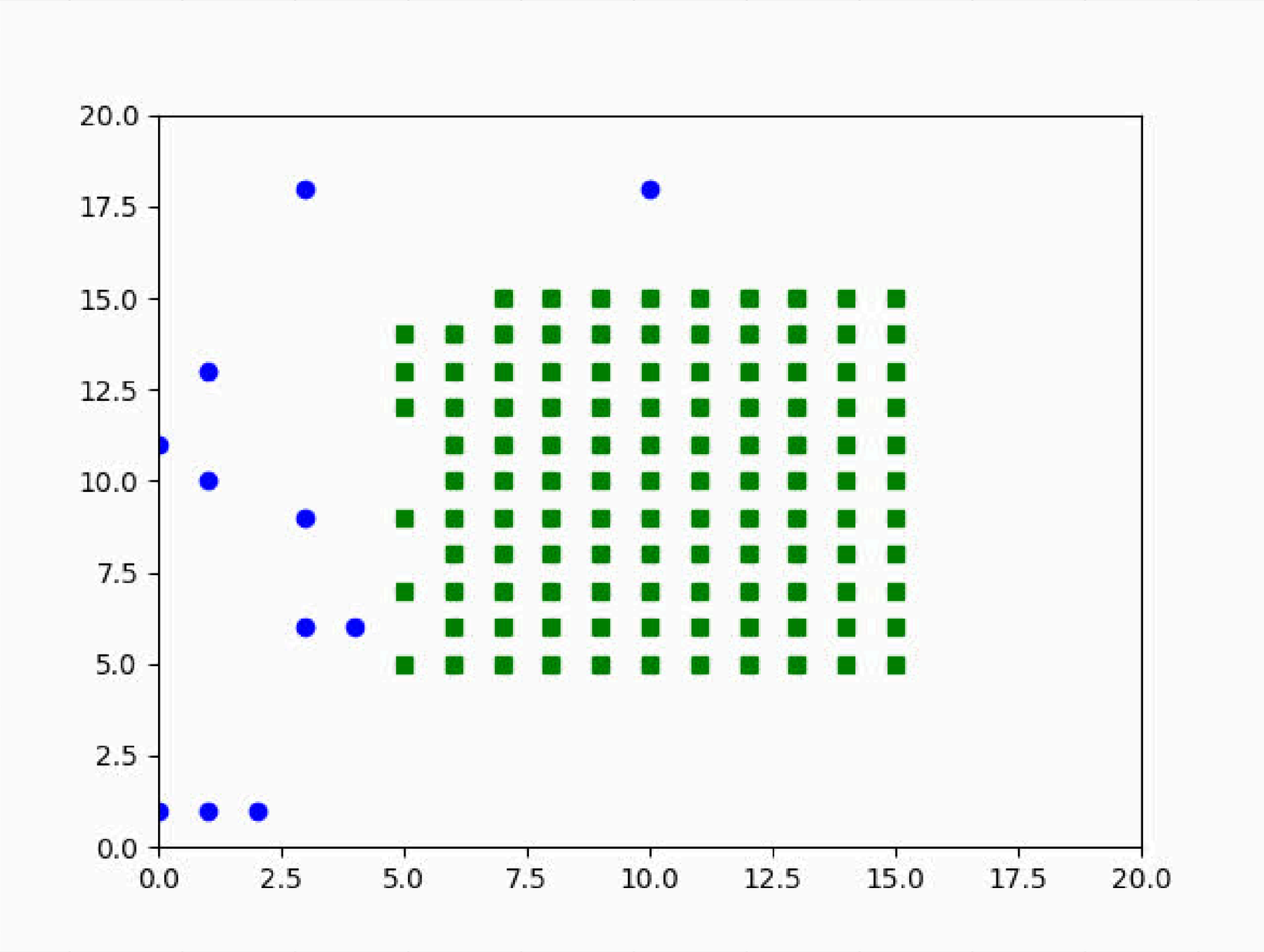
ADHERENCIA

```
if grilla[particula["pos_actual"][0]][particula["pos_actual"][1]] == SOLIDO:

    if hubo_adherencia(particula["semilla_adherencia"]):
        grilla[particula["pos_actual"][0]][particula["pos_actual"][1]] = AIRE
        particula["adherida"] = True
    else:
        # reseto el estado donde estaba la particula
        particula["estado_anterior"] = grilla[pos_anterior[0]][pos_anterior[1]]
        # reseteo la posicion
        particula["pos_actual"] = pos_anterior
        # reboto la particula
        grilla[particula["pos_actual"][0]][particula["pos_actual"][1]] = PARTICULA
        particula["semilla_adherencia"] += 2
```

ANIMACIÓN SIN CONSUMO



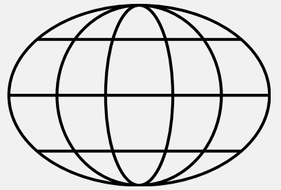
ANIMACIÓN CON CONSUMO



CONCLUSIONES

NOS GUSTÓ

Implementar el generador y realizar la animación



DIFICULTADES

Encontrar un generador
Comprensión de los papers

GRACIAS