

Practica

1. R0 obtenidos de la prediccion del SIR (Trabajo anterior)
2. Predecir que va a ocurrir la proxima semana.
3. El valor 4, el cual representaría el peor de los casos.
4. El valor 1.4 en el mejor de los casos
5. R0 con las medidas realizadas por el Ecuador, obtenemos el R0 solo de los dias sin cuarentena y lo evaluamos con las acciones de la cuarentena.

1.R0 obtenidos de la prediccion del SIR (Trabajo anterior)

In [9]:

```
class Learner(object):
    def __init__(self, country, loss, start_date, predict_range, s_0, i_0, r_0):
        self.country = country
        self.loss = loss
        self.start_date = start_date
        self.predict_range = predict_range
        self.s_0 = s_0
        self.i_0 = i_0
        self.r_0 = r_0
        self.Beta = 0
        self.Gamma = 0

    def load_confirmed(self, country):
        df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv')
        country_df = df[df['Country/Region'] == country]
        return country_df.iloc[0].loc[self.start_date:]

    def extend_index(self, index, new_size):
        values = index.values
        current = datetime.strptime(index[-1], '%m/%d/%y')
        while len(values) < new_size:
            current = current + timedelta(days=1)
            values = np.append(values, datetime.strptime(current, '%m/%d/%y'))
        return values

    def predict(self, beta, gamma, data, country, s_0, i_0, r_0):
        new_index = self.extend_index(data.index, self.predict_range)
        size = len(new_index)
        def SIR(t, y):
            S = y[0]
            I = y[1]
            R = y[2]
            return [-beta*S*I, beta*S*I-gamma*I, gamma*I]
        extended_actual = np.concatenate((data.values, [None] * (size - len(data.values))))
        return new_index, extended_actual, solve_ivp(SIR, [0, size], [s_0, i_0, r_0], t_eval=np.arange(0, size, 1))

    def train(self):
        data = (self.load_confirmed(self.country))
        self.i_0 = data[-1]
        optimal = minimize(loss, [0.001, 0.001], args=(data, self.s_0, self.i_0, self.r_0), method='L-BFGS-B', bounds=[(0.00000001, 1.80), (0.00000001, 0.899)])
        beta, gamma = optimal.x
        self.Beta, self.Gamma = optimal.x
        new_index, extended_actual, prediction = self.predict(beta, gamma, data, self.country, self.s_0, self.i_0, self.r_0)
        df = pd.DataFrame({'Infected': extended_actual, 'Susceptibles': prediction.y[0], 'Recovered': prediction.y[2]}, index=new_index)
        fig, ax = plt.subplots(figsize=(12, 7))
        ax.set_title(self.country)
        df.plot(ax=ax)
        self.r_0 = (beta/gamma)
        #print(f"\n beta={beta:.8f}, gamma={gamma:.8f}, R0: {(beta/gamma):.8f}")
```

```
contadiadosA = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv')
contadiadosA = contadiadosA[contadiadosA['Country/Region'] == 'Ecuador'].iloc[0].loc[:]
```

```
def loss(point, data, s_0, i_0, r_0):
```

```
    size = len(data)
    beta, gamma = point
```

```
    def SIR(t, y):
```

```
        S = y[0]
```

```
        I = y[1]
```

```
        R = y[2]
```

```

return [-beta*S*I, beta*S*I-gamma*I, gamma*I]
solution = solve_ivp(SIR, [0, size], [s_0,i_0,r_0], t_eval=np.arange(0, size, 1), vectorized=True)
return np.sqrt(np.mean((solution.y[1] - data)**2))

```

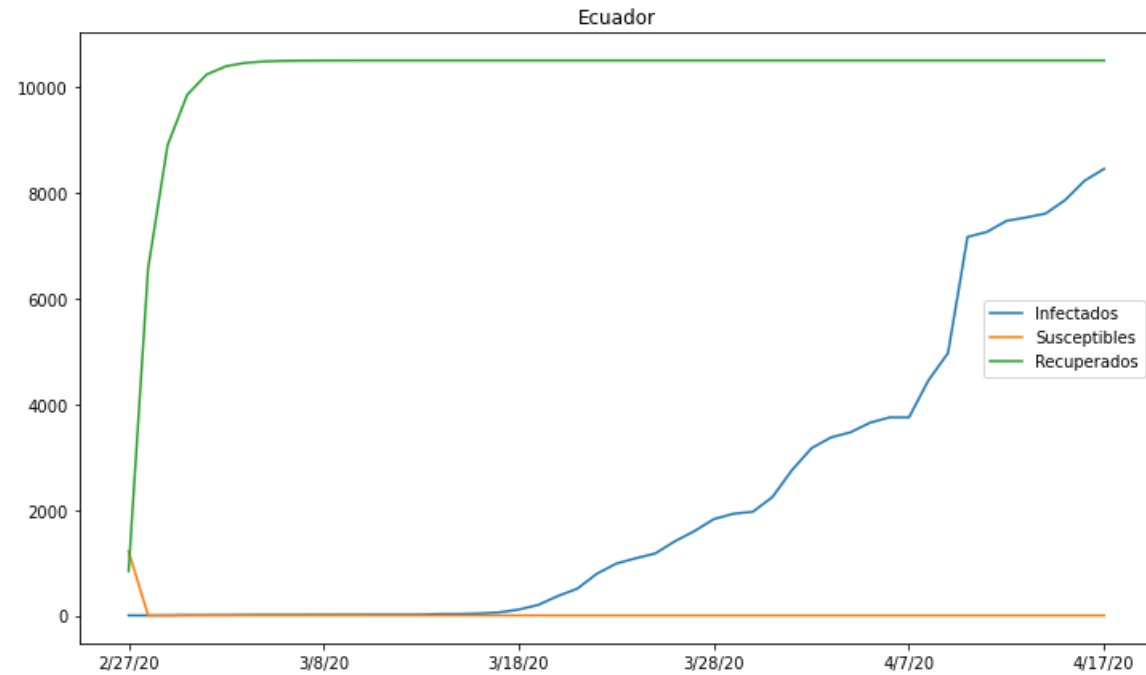
```

N=10500
i0=contadiadosA[-1]
r0=838
s0=N - i0 - r0

print("Actual")
actual = Learner('Ecuador', loss, '2/27/20', 0, s0, i0, r0) #R0 ACTUAL
actual.train()
R0_actual=actual.r_0
print("R0_actual:", R0_actual )

```

Actual
R0_actual: 2.0022246941045605



In []:

```

from random import randrange # Obtener un numero randomico
import pygame

#Parametros de inicio
Dias=7
PROBA_MUERTE = 8.4 # Probabilidad de que la gente muera COVID
CONTAGION_RATE =R0_actual# Factor R0 para la simulacion COVID probabilidad
PROBA_INFECT = CONTAGION_RATE * 10
PROBA_VACU = 0 # Probabilidad de que exista una vacuna, COVID = 0
SIMULACION_SPEED = Dias*25 # Tiempo de un día en milisegundos (Cada 25 es un día)
nb_rows = 45 #Numero de filas
nb_cols = 45 #Numero de columnas

```

```

global display, myfont, states, states_temp #Declaracion de variables globales

```

```

#Declaro colores en formato RGB

```

```

WHITE = (255, 255, 255)
BLUE = (0, 0, 255)
GREEN = (0, 247, 0)
BLACK = (0, 0, 0)

```

```

#Obtiene los vecinos dado un punto x,y

```

```

def get_vecinos(x, y):
    incx = randrange(3)
    incy = randrange(3)
    incx = (incx * 1) - 1
    incy = (incy * 1) - 1
    x2 = x + incx
    y2 = y + incy
    #Validar limites
    if x2 < 0:
        x2 = 0
    if x2 >= nb_cols:
        x2 = nb_cols - 1

```

```

if y2 < 0:
    y2 = 0
if y2 >= nb_rows:
    y2 = nb_rows - 1
return [x2, y2] # Nuevos contagiados

#Genero las personas que cuentan con inmunidad o vacuna
def vacunar():
    for x in range(nb_cols):
        for y in range(nb_rows):
            if randrange(99) < PROBA_VACU:
                states[x][y] = 1

#Funcion que permite contar el numero de muertos de la matriz states == -1
def contar_muertes():
    contador = 0
    for x in range(nb_cols):
        for y in range(nb_rows):
            if states[x][y] == -1:
                contador += 1
    return contador

#Definimos datos de inicio
states = [[0] * nb_cols for i1 in range(nb_rows)]
states_temp = states.copy()
states[randrange(50)][randrange(50)] = 10 # Estado inicial de la simulacion Posicion del Infectado
it = 0 # Variable para contar las iteraciones
total_muerte = 0 # Contabiliza el numero de muertos
vacunar() #Llamar a la funcion vacunar

pygame.init() #Incializo el motor de juegos pygame
pygame.font.init() #Incializo el tipo de letra
display=pygame.display.set_mode((800,750),0,32) #Tamaño de la ventana
pygame.display.set_caption("Simulacion de Epidemia Covid-19 Ecuador")# Titulo
font=pygame.font.SysFont('Calibri', 40) # Tipo de letra
display.fill(WHITE) # Color de fondo

while True:
    pygame.time.delay(SIMULACION_SPEED) # Sleep o pausa
    it = it + 1
    if it <= 10000 and it >= 2:
        states_temp = states.copy() #Copia de la matriz
        #Recorrera la matriz
        for x in range(nb_cols):
            for y in range(nb_rows):
                state = states[x][y]
                if state == -1:
                    pass
                if state >= 10: # Numero de días de contagio
                    states_temp[x][y] = state + 1
                if state >= 20:
                    if randrange(99) < PROBA_MUERTE: # Genero un randomico para verificar si fallece o se recupera
                        states_temp[x][y] = -1 # Muere
                    else:
                        states_temp[x][y] = 1 # Cura o recupera
                if state >= 10 and state <= 20: # Rango de infectado
                    if randrange(99) < PROBA_INFECT: # Infecto a las personas cercanas entre 10 y 20
                        neighbour = get_vecinos(x, y) #Obtenemos los vecinos a contagiar
                        x2 = neighbour[0]
                        y2 = neighbour[1]
                        neigh_state = states[x2][y2]
                        if neigh_state == 0: #Verifico que este sano
                            states_temp[x2][y2] = 10 # Contagia
        states = states_temp.copy()
        total_muerte = contar_muertes() # contar el numero de muertos

    pygame.draw.rect(display, WHITE, (250, 30, 260, 50)) # Grafico el fondo
    textsurface = font.render("Total muertes: "+ str(total_muerte), False, (255,160,122)) #El numero de muertos
    display.blit(textsurface, (250, 30)) # Graficar el texto de muertes
    #Graficar el estado del paciente matriz
    for x in range(nb_cols):
        for y in range(nb_rows):
            if states[x][y] == 0:
                color = BLUE # No infectado
            if states[x][y] == 1:
                color = GREEN # Recupero
            if states[x][y] >= 10:
                color = (states[x][y] * 12, 50, 50) # Infectado - Rojo
            if states[x][y] == -1:
                color = BLACK # Muerto
            pygame.draw.circle(display, color, (100 + x * 12 + 5, 100 + y * 12 + 5), 5)
            pygame.draw.rect(display, WHITE, (100 + x * 12 + 3, 100 + y * 12 + 4, 1, 1))
    #Escuchar los eventos del teclado

```

```

#Escuchar los eventos del teclado
for event in pygame.event.get():
    if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE: #Presiona y Escape
        pygame.quit() #Termino simulacion
    if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE: #Presiona y espacio
        #Reiniciamos valores
        states = [[0] * nb_cols for i1 in range(nb_rows)]
        states_temp = states.copy()
        states[5][5] = 10
        it = 0
        total_muerte = 0
        vacunar() #Llamar a la funcion vacunar

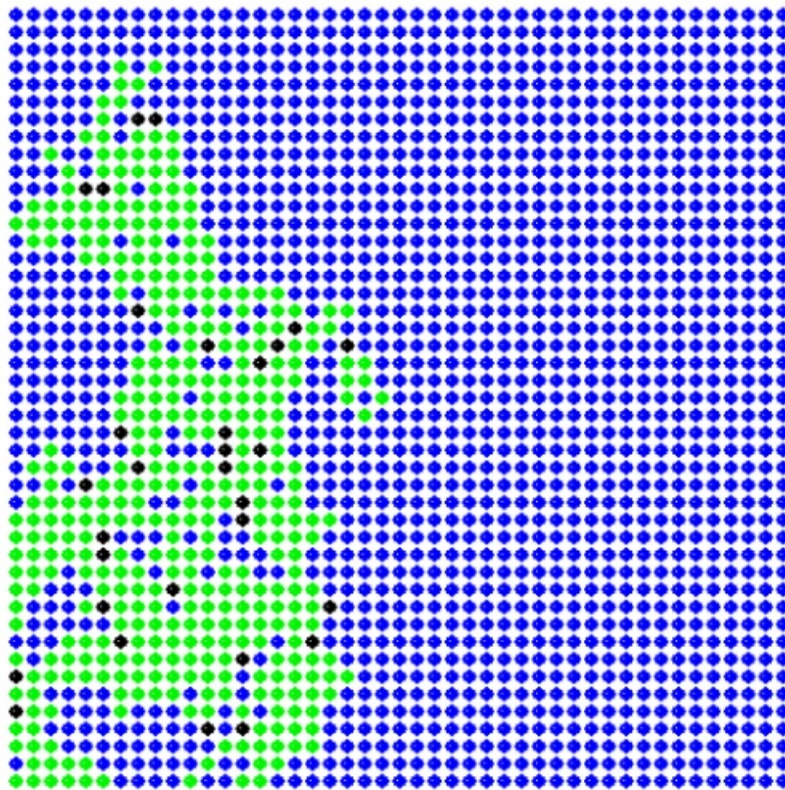
pygame.display.update()# Mandar actualizar la ventana

```

pygame 1.9.6

Hello from the pygame community. <https://www.pygame.org/contribute.html>

Total muertes: 3 🏴‍☠️



2.Predecir que va a ocurrir la proxima semana

In [12]:

```

class Learner(object):
    def __init__(self, country, loss, start_date, predict_range,s_0, i_0, r_0):
        self.country = country
        self.loss = loss
        self.start_date = start_date
        self.predict_range = predict_range
        self.s_0 = s_0
        self.i_0 = i_0
        self.r_0 = r_0
        self.Beta = 0
        self.Gamma = 0

    def load_confirmed(self, country):
        df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv')
        country_df = df[df['Country/Region'] == country]
        return country_df.iloc[0].loc[self.start_date:]

    def extend_index(self, index, new_size):
        values = index.values
        current = datetime.strptime(index[-1], '%m/%d/%y')
        while len(values) < new_size:

```

```

current = current + timedelta(days=1)
values = np.append(values, datetime.strptime(current, '%m/%d/%y'))
return values

def predict(self, beta, gamma, data, country, s_0, i_0, r_0):
    new_index = self.extend_index(data.index, self.predict_range)
    size = len(new_index)
    def SIR(t, y):
        S = y[0]
        I = y[1]
        R = y[2]
        return [-beta*S*I, beta*S*I-gamma*I, gamma*I]
    extended_actual = np.concatenate((data.values, [None] * (size - len(data.values))))
    return new_index, extended_actual, solve_ivp(SIR, [0, size], [s_0,i_0,r_0], t_eval=np.arange(0, size, 1))

def train(self):
    data = (self.load_confirmed(self.country))
    self.i0=data[-1]
    optimal = minimize(loss, [0.001, 0.001], args=(data, self.s_0, self.i_0, self.r_0), method='L-BFGS-B', bounds=[(0.00000001, 1.98125), (0.0000001, 0.911)])
    beta, gamma = optimal.x
    self.Beta, self.Gamma = optimal.x
    new_index, extended_actual, prediction = self.predict(beta, gamma, data, self.country, self.s_0, self.i_0, self.r_0)
    df = pd.DataFrame({'Infectados': extended_actual, 'Susceptibles': prediction.y[0], 'Recuperados': prediction.y[2]}, index=new_index)
    fig, ax = plt.subplots(figsize=(12, 7))
    ax.set_title(self.country)
    df.plot(ax=ax)
    self.r_0=(beta/gamma)
    #print(f"\n beta={beta:.8f}, gamma={gamma:.8f}, R0: {(beta/gamma):.8f}")

contadiadosA = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv')
contadiadosA = contadiadosA[contadiadosA['Country/Region'] == 'Ecuador'].iloc[0].loc[::]

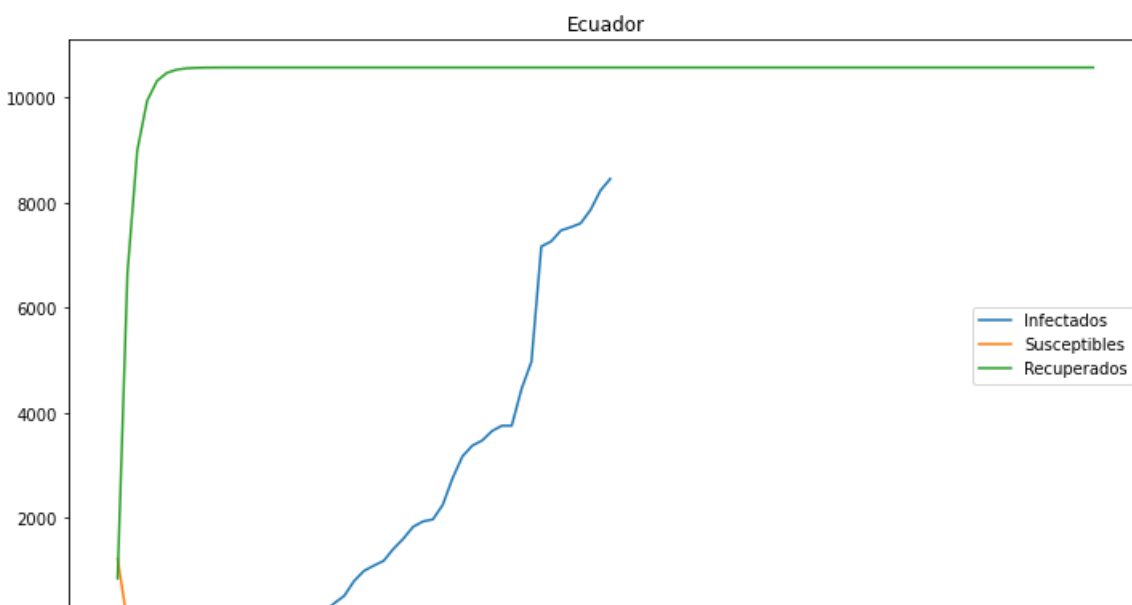
def loss(point, data, s_0, i_0, r_0):
    size = len(data)
    beta, gamma = point
    def SIR(t, y):
        S = y[0]
        I = y[1]
        R = y[2]
        return [-beta*S*I, beta*S*I-gamma*I, gamma*I]
    solution = solve_ivp(SIR, [0, size], [s_0,i_0,r_0], t_eval=np.arange(0, size, 1), vectorized=True)
    return np.sqrt(np.mean((solution.y[1] - data)**2))

N=10500
i0=contadiadosA[-1]
r0=838
s0=N - i0 - r0

print('En una semana')
semana = Learner('Ecuador', loss, '2/27/20', 100, s0, 8520, r0) #RO en una semana
semana.train()
R0_semana=semana.r_0
print('R0_semana:', R0_semana)

```

En una semana
R0_semana: 2.174674515928388



In []:

```
from random import randrange # Obtener un numero randomico
import pygame
```

```
#Parametros de inicio
```

```
Dias=7
PROBA_MUERTE = 8.4 # Probabilidad de que la gente muera COVID
CONTAGION_RATE =R0_semana# Factor R0 para la simulacion COVID probabilidad
PROBA_INFECT = CONTAGION_RATE * 10
PROBA_VACU = 0 # Probabilidad de que exista una vacuna, COVID = 0
SIMULACION_SPEED = Dias*25 # Tiempo de un dia en milisegundos (Cada 25 es un dia)
nb_rows = 45 #Numero de filas
nb_cols = 45 #Numero de columnas
```

```
global display, myfont, states, states_temp #Declaracion de variables globales
```

```
#Declaro colores en formato RGB
```

```
WHITE = (255, 255, 255)
BLUE = (0, 0, 255)
GREEN = (0, 247, 0)
BLACK = (0, 0, 0)
```

```
#Obtiene los vecinos dado un punto x,y
```

```
def get_vecinos(x, y):
    incx = randrange(3)
    incy = randrange(3)
    incx = (incx * 1) - 1
    incy = (incy * 1) - 1
    x2 = x + incx
    y2 = y + incy
    #Validar limites
    if x2 < 0:
        x2 = 0
    if x2 >= nb_cols:
        x2 = nb_cols - 1
    if y2 < 0:
        y2 = 0
    if y2 >= nb_rows:
        y2 = nb_rows - 1
    return [x2, y2] # Nuevos contagiados
```

```
#Genero las personas que cuentan con inmunidad o vacuna
```

```
def vacunar():
    for x in range(nb_cols):
        for y in range(nb_rows):
            if randrange(99) < PROBA_VACU:
                states[x][y] = 1
```

```
#Funcion que permite contar el numero de muertosde la matriz states == -1
```

```
def contar_muertes():
    contador = 0
    for x in range(nb_cols):
        for y in range(nb_rows):
            if states[x][y] == -1:
                contador += 1
    return contador
```

```
#Definimos datos de inicio
```

```
states = [[0] * nb_cols for i1 in range(nb_rows)]
states_temp = states.copy()
states[randrange(50)][randrange(50)] = 10 # Estado inicial de la simulacion Posicion del Infectado
it = 0 # Variable para contar las iteraciones
total_muerte = 0 # Contabiliza el numero de muertos
vacunar() #Llamar a la funcion vacunar
```

```
pygame.init() #Incializo el motor de juegos pygame
pygame.font.init() #Incializo el tipo de letra
display=pygame.display.set_mode((800,750),0,32) #Tamaño de la ventana
pygame.display.set_caption("Simulacion de Epidemia Covid-19 Ecuador")# Titulo
font=pygame.font.SysFont('Calibri', 40) # Tipo de letra
display.fill(WHITE) # Color de fondo
```

```
while True:
    pygame.time.delay(SIMULACION_SPEED) # Sleep o pausa
    it = it + 1
    if it <= 10000 and it >= 2:
        states_temp = states.copy() #Copia de la matriz
```



```

states_temp = states.copy() #Copia de la matriz
#Recorrera la matriz
for x in range(nb_cols):
    for y in range(nb_rows):
        state = states[x][y]
        if state == -1:
            pass
        if state >= 10: # Numero de dias de contagio
            states_temp[x][y] = state + 1
        if state >= 20:
            if randrange(99) < PROBA_MUERTE: # Genero un randomico para verificar si fallece o se recupera
                states_temp[x][y] = -1 # Muere
            else:
                states_temp[x][y] = 1 # Cura o recupera
        if state >= 10 and state <= 20: # Rango de infectado
            if randrange(99) < PROBA_INFECT: # Infecto a las personas cercanas entre 10 y 20
                neighbour = get_vecinos(x, y) #Obtenemos los vecinos a contagiar
                x2 = neighbour[0]
                y2 = neighbour[1]
                neigh_state = states[x2][y2]
                if neigh_state == 0: #Verifico que este sano
                    states_temp[x2][y2] = 10 # Contagia
states = states_temp.copy()
total_muerte = contar_muertes() # contar el numero de muertos

pygame.draw.rect(display, WHITE, (250, 30, 260, 50)) # Grafico el fondo
textsurface = font.render("Total muertes: " + str(total_muerte), False, (255,160,122)) #El numero de muertos
display.blit(textsurface, (250, 30)) # Graficar el texto de muertes
#Graficar el estado del paciente matriz
for x in range(nb_cols):
    for y in range(nb_rows):
        if states[x][y] == 0:
            color = BLUE # No infectado
        if states[x][y] == 1:
            color = GREEN # Recupero
        if states[x][y] >= 10:
            color = (states[x][y] * 12, 50, 50) # Inyectado - Rojo
        if states[x][y] == -1:
            color = BLACK # Muerto
        pygame.draw.circle(display, color, (100 + x * 12 + 5, 100 + y * 12 + 5), 5)
        pygame.draw.rect(display, WHITE, (100 + x * 12 + 3, 100 + y * 12 + 4, 1, 1))
#Escuchar los eventos del teclado
for event in pygame.event.get():
    if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE: #Presiona y Escape
        pygame.quit() #Termino simulacion
    if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE: #Presiona y espacio
        #Reiniciamos valores
        states = [[0] * nb_cols for i1 in range(nb_rows)]
        states_temp = states.copy()
        states[5][5] = 10
        it = 0
        total_muerte = 0
        vacunar() #Llamar a la funcion vacunar

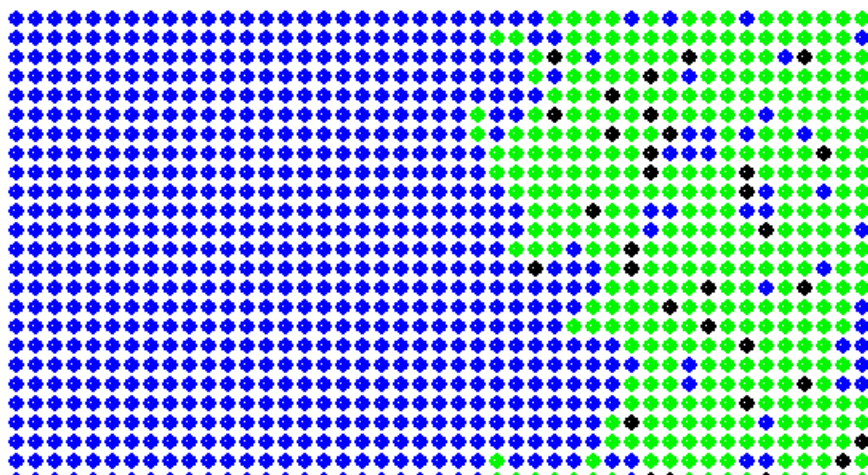
pygame.display.update()# Mandar actualizar la ventana

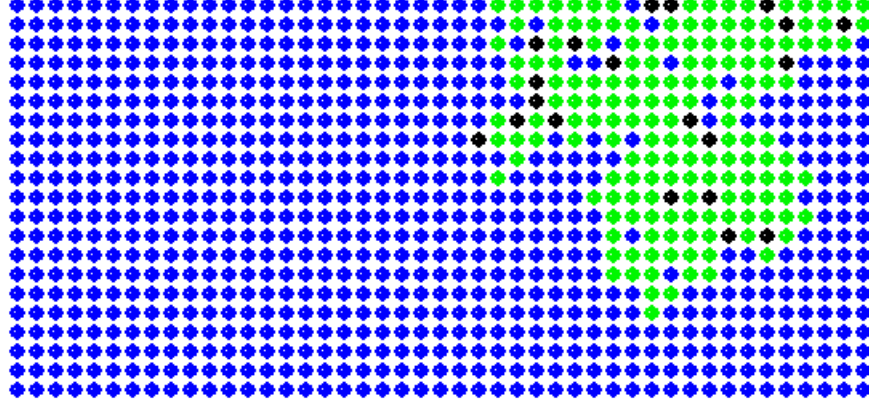
```

pygame 1.9.6

Hello from the pygame community. <https://www.pygame.org/contribute.html>

Total muertes: 4





3.El valor 4, el cual representaría el peor de los casos.

In []:

```
from random import randrange # Obtener un numero randomico
import pygame

#Parametros de inicio
Dias=7
PROBA_MUERTE = 8.4 # Probabilidad de que la gente muera COVID
CONTAGION_RATE =4# Factor R0 para la simulacion COVID probabilidad
PROBA_INFECT = CONTAGION_RATE * 10
PROBA_VACU = 0 # Probabilidad de que exista una vacuna, COVID = 0
SIMULACION_SPEED = Dias*25 # Tiempo de un dia en milisegundos (Cada 25 es un dia)
nb_rows = 45 #Numero de filas
nb_cols = 45 #Numero de columnas

global display, myfont, states, states_temp #Declaracion de variables globales

#Declaro colores en formato RGB
WHITE = (255, 255, 255)
BLUE = (0, 0, 255)
GREEN = (0, 247, 0)
BLACK = (0, 0, 0)

#Obtiene los vecinos dado un punto x,y
def get_vecinos(x, y):
    incx = randrange(3)
    incy = randrange(3)
    incx = (incx * 1) - 1
    incy = (incy * 1) - 1
    x2 = x + incx
    y2 = y + incy
    #Validar limites
    if x2 < 0:
        x2 = 0
    if x2 >= nb_cols:
        x2 = nb_cols - 1
    if y2 < 0:
        y2 = 0
    if y2 >= nb_rows:
        y2 = nb_rows - 1
    return [x2, y2] # Nuevos contagiados

#Genero las personas que cuentan con inmunidad o vacuna
def vacunar():
    for x in range(nb_cols):
        for y in range(nb_rows):
            if randrange(99) < PROBA_VACU:
                states[x][y] = 1

#Funcion que permite contar el numero de muertosde la matriz states == -1
def contar_muertes():
    contador = 0
    for x in range(nb_cols):
        for y in range(nb_rows):
            if states[x][y] == -1:
                contador += 1
    return contador

#Definimos datos de inicio
states = [[0] * nb_cols for i1 in range(nb_rows)]
states_temp = states.copy()
states[randrange(50)][randrange(50)] = 10 # Estado inicial de la simulacion Posicion del Infectado
```



```

it = 0 # Variable para contar las iteraciones
total_muerte = 0 # Contabiliza el numero de muertos
vacunar() #Llamar a la funcion vacunar

pygame.init() #Incializo el motor de juegos pygame
pygame.font.init() #Incializo el tipo de letra
display=pygame.display.set_mode((800,750),0,32) #Tamanio de la ventana
pygame.display.set_caption("Simulacion de Epidemia Covid-19 Ecuador")# Titulo
font=pygame.font.SysFont('Calibri', 40) # Tipo de letra
display.fill(WHITE) # Color de fondo

while True:
    pygame.time.delay(SIMULACION_SPEED) # Sleep o pausa
    it = it + 1
    if it <= 10000 and it >= 2:
        states_temp = states.copy() #Copia de la matriz
        #Recorrera la matriz
        for x in range(nb_cols):
            for y in range(nb_rows):
                state = states[x][y]
                if state == -1:
                    pass
                if state >= 10: # Numero de dias de contagio
                    states_temp[x][y] = state + 1
                if state >= 20:
                    if randrange(99) < PROBA_MUERTE: # Genero un randomico para verificar si fallece o se recupera
                        states_temp[x][y] = -1 # Muere
                    else:
                        states_temp[x][y] = 1 # Cura o recupera
                if state >= 10 and state <= 20: # Rango de infectado
                    if randrange(99) < PROBA_INFECT: # Infecto a las personas cercanas entre 10 y 20
                        neighbour = get_vecinos(x, y) #Obtenemos los vecinos a contagiar
                        x2 = neighbour[0]
                        y2 = neighbour[1]
                        neigh_state = states[x2][y2]
                        if neigh_state == 0: #Verifico que este sano
                            states_temp[x2][y2] = 10 # Contagia
            states = states_temp.copy()
            total_muerte = contar_muertes() # contar el numero de muertos

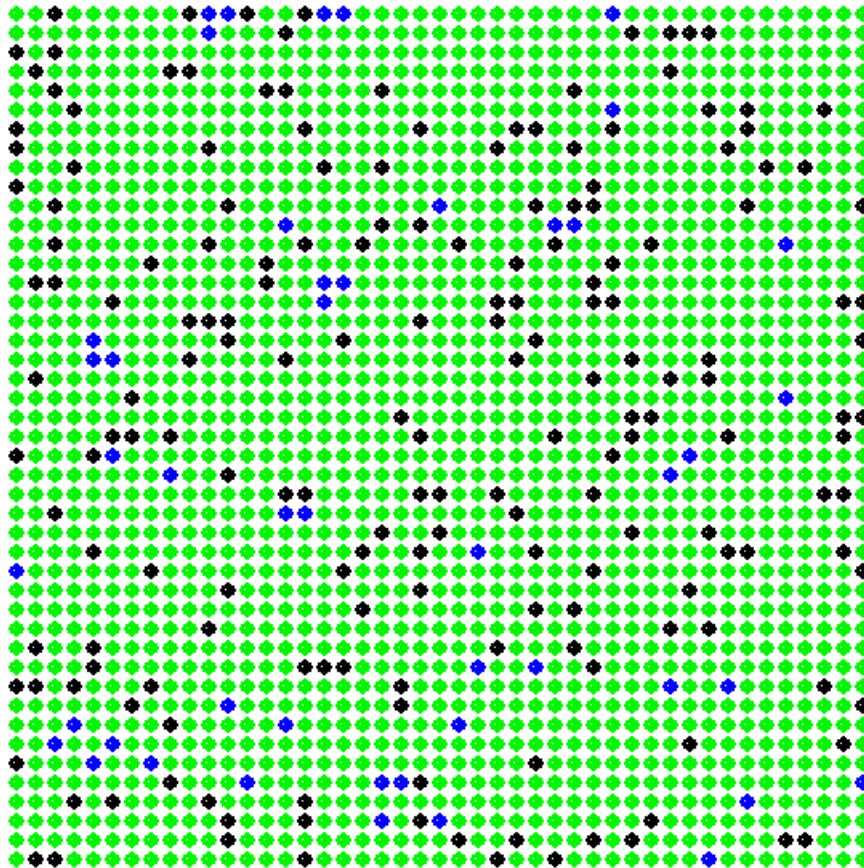
    pygame.draw.rect(display, WHITE, (250, 30, 260, 50)) # Grafico el fondo
    textsurface = font.render("Total muertes: " + str(total_muerte), False, (255,160,122)) #El numero de muertos
    display.blit(textsurface, (250, 30)) # Graficar el texto de muertes
    #Graficar el estado del paciente matriz
    for x in range(nb_cols):
        for y in range(nb_rows):
            if states[x][y] == 0:
                color = BLUE # No infectado
            if states[x][y] == 1:
                color = GREEN # Recupero
            if states[x][y] >= 10:
                color = (states[x][y] * 12, 50, 50) # Inyectado - Rojo
            if states[x][y] == -1:
                color = BLACK # Muerto
            pygame.draw.circle(display, color, (100 + x * 12 + 5, 100 + y * 12 + 5), 5)
            pygame.draw.rect(display, WHITE, (100 + x * 12 + 3, 100 + y * 12 + 4, 1, 1))
    #Escuchar los eventos del teclado
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE: #Presiona y Escape
            pygame.quit() #Termino simulacion
        if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE: #Presiona y espacio
            #Reiniciamos valores
            states = [[0] * nb_cols for i1 in range(nb_rows)]
            states_temp = states.copy()
            states[5][5] = 10
            it = 0
            total_muerte = 0
            vacunar() #Llamar a la funcion vacunar

    pygame.display.update()# Mandar actualizar la ventana

```

pygame 1.9.6

Hello from the pygame community. <https://www.pygame.org/contribute.html>



4.El valor 1.4 en el mejor de los casos

In []:

```
from random import randrange # Obtener un numero randomico
import pygame

#Parametros de inicio
Dias=7
PROBA_MUERTE = 8.4 # Probabilidad de que la gente muera COVID
CONTAGION_RATE =1.4# Factor R0 para la simulacion COVID probabilidad
PROBA_INFECT = CONTAGION_RATE * 10
PROBA_VACU = 0 # Probabilidad de que exista una vacuna, COVID = 0
SIMULACION_SPEED = Dias*25 # Tiempo de un día en milisegundos (Cada 25 es un dia)
nb_rows = 45 #Numero de filas
nb_cols = 45 #Numero de columnas

global display, myfont, states, states_temp #Declaracion de variables globales

#Declaro colores en formato RGB
WHITE = (255, 255, 255)
BLUE = (0, 0, 255)
GREEN = (0, 247, 0)
BLACK = (0, 0, 0)

#Obtiene los vecinos dado un punto x,y
def get_vecinos(x, y):
    incx = randrange(3)
    incy = randrange(3)
    incx = (incx * 1) - 1
    incy = (incy * 1) - 1
    x2 = x + incx
    y2 = y + incy
    #Validar limites
    if x2 < 0:
        x2 = 0
    if x2 >= nb_cols:
        x2 = nb_cols - 1
    if y2 < 0:
        y2 = 0
    if y2 >= nb_rows:
        y2 = nb_rows - 1
    return [x2, y2] # Nuevos contagiados

#Genero las personas que cuentan con inmunidad o vacuna
def vacunar():
    for x in range(nb_cols):
```

```

for x in range(nb_cols):
    for y in range(nb_rows):
        if randrange(99) < PROBA_VACU:
            states[x][y] = 1

#Funcion que permite contar el numero de muertos de la matriz states == -1
def contar_muertes():
    contador = 0
    for x in range(nb_cols):
        for y in range(nb_rows):
            if states[x][y] == -1:
                contador += 1
    return contador

#Definimos datos de inicio
states = [[0] * nb_cols for i1 in range(nb_rows)]
states_temp = states.copy()
states[randrange(50)][randrange(50)] = 10 # Estado inicial de la simulacion Posicion del Infectado
it = 0 # Variable para contar las iteraciones
total_muerte = 0 # Contabiliza el numero de muertos
vacunar() #Llamar a la funcion vacunar

pygame.init() #Incializo el motor de juegos pygame
pygame.font.init() #Incializo el tipo de letra
display=pygame.display.set_mode((800,750),0,32) #Tamaño de la ventana
pygame.display.set_caption("Simulacion de Epidemia Covid-19 Ecuador")# Titulo
font=pygame.font.SysFont('Calibri', 40) # Tipo de letra
display.fill(WHITE) # Color de fondo

while True:
    pygame.time.delay(SIMULACION_SPEED) # Sleep o pausa
    it = it + 1
    if it <= 10000 and it >= 2:
        states_temp = states.copy() #Copia de la matriz
        #Recorrera la matriz
        for x in range(nb_cols):
            for y in range(nb_rows):
                state = states[x][y]
                if state == -1:
                    pass
                if state >= 10: # Numero de dias de contagio
                    states_temp[x][y] = state + 1
                if state >= 20:
                    if randrange(99) < PROBA_MUERTE: # Genero un randomico para verificar si fallece o se recupera
                        states_temp[x][y] = -1 # Muere
                    else:
                        states_temp[x][y] = 1 # Cura o recupera
                if state >= 10 and state <= 20: # Rango de infectado
                    if randrange(99) < PROBA_INFECT: # Infecto a las personas cercanas entre 10 y 20
                        neighbour = get_vecinos(x, y) #Obtenemos los vecinos a contagiar
                        x2 = neighbour[0]
                        y2 = neighbour[1]
                        neigh_state = states[x2][y2]
                        if neigh_state == 0: #Verifico que este sano
                            states_temp[x2][y2] = 10 # Contagia
        states = states_temp.copy()
        total_muerte = contar_muertes() # contar el numero de muertos

    pygame.draw.rect(display, WHITE, (250, 30, 260, 50)) # Grafico el fondo
    textsurface = font.render("Total muertes: " + str(total_muerte), False, (255,160,122)) #El numero de muertos
    display.blit(textsurface, (250, 30)) # Graficar el texto de muertes
    #Graficar el estado del paciente matriz
    for x in range(nb_cols):
        for y in range(nb_rows):
            if states[x][y] == 0:
                color = BLUE # No infectado
            if states[x][y] == 1:
                color = GREEN # Recupero
            if states[x][y] >= 10:
                color = (states[x][y] * 12, 50, 50) # Inyectado - Rojo
            if states[x][y] == -1:
                color = BLACK # Muerto
            pygame.draw.circle(display, color, (100 + x * 12 + 5, 100 + y * 12 + 5), 5)
            pygame.draw.rect(display, WHITE, (100 + x * 12 + 3, 100 + y * 12 + 4, 1, 1))

    #Escuchar los eventos del teclado
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE: #Presiona y Escape
            pygame.quit() #Termino simulacion
        if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE: #Presiona y espacio
            #Reiniciamos valores
            states = [[0] * nb_cols for i1 in range(nb_rows)]
            states_temp = states.copy()
            states[5][5] = 10


```

```
it = 0
total_muerte = 0
vacunar() #Llamar a la funcion vacunar
```

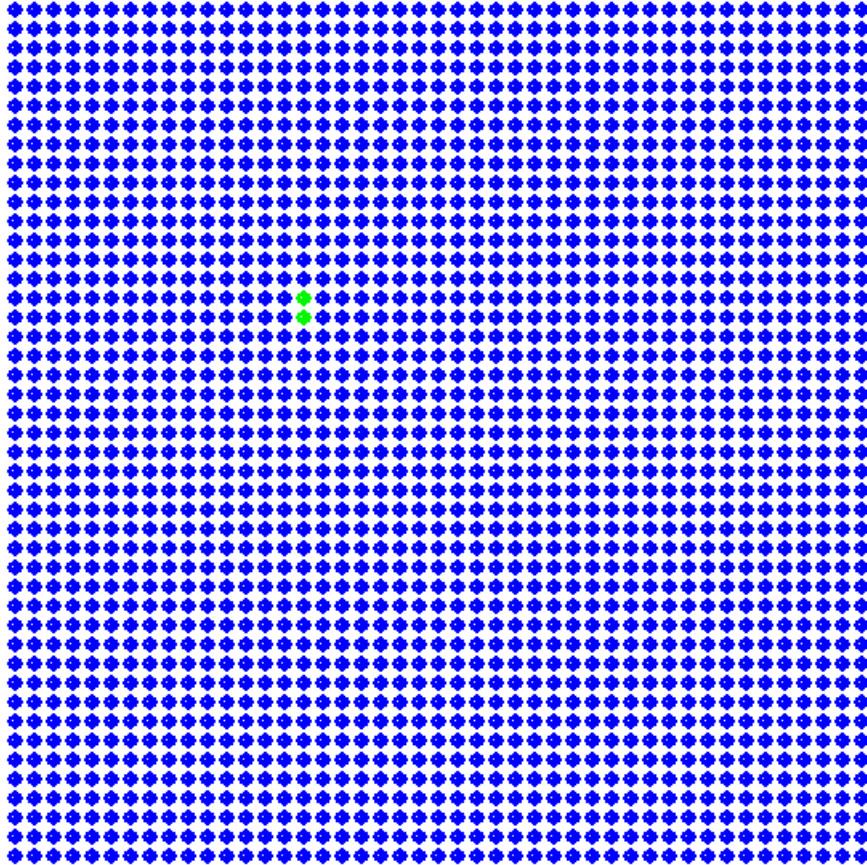
```
pygame.display.update()# Mandar actualizar la ventana
```

pygame 1.9.6

Hello from the pygame community. <https://www.pygame.org/contribute.html>

 Simulacion de Epidemia Covid-19 Ecuador

Total muertes: 0



5.R0 con las medidas realizadas por el Ecuador, obtenemos el R0 solo de los días sin cuarentena y lo evaluamos con las acciones de la cuarentena.

In [2]:

```
import numpy as np
import pandas as pd
from csv import reader
from csv import writer
from scipy.integrate import solve_ivp
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from datetime import timedelta, datetime
import argparse
import sys
import json
import ssl
import urllib.request
```

```
class Learner(object):
    def __init__(self, country, loss, start_date, predict_range, s_0, i_0, r_0):
        self.country = country
        self.loss = loss
        self.start_date = start_date
        self.predict_range = predict_range
        self.s_0 = s_0
        self.i_0 = i_0
        self.r_0 = r_0
        self.data = []
```

```

self.Beta = 0
self.Gamma = 0

def load_confirmed(self, country):
    df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv')
    country_df = df[df['Country/Region'] == country]
    return country_df.iloc[0].loc[self.start_date:]

def extend_index(self, index, new_size):
    values = index.values
    current = datetime.strptime(index[-1], '%m/%d/%y')
    while len(values) < new_size:
        current = current + timedelta(days=1)
        values = np.append(values, datetime.strptime(current, '%m/%d/%y'))
    return values

def predict(self, beta, gamma, data, country, s_0, i_0, r_0):
    new_index = self.extend_index(data.index, self.predict_range)
    size = len(new_index)
    def SIR(t, y):
        S = y[0]
        I = y[1]
        R = y[2]
        return [-beta*S*I, beta*S*I-gamma*I, gamma*I]
    extended_actual = np.concatenate((data.values, [None] * (size - len(data.values))))
    return new_index, extended_actual, solve_ivp(SIR, [0, size], [s_0, i_0, r_0], t_eval=np.arange(0, size, 1))

def train(self):
    data = (self.load_confirmed(self.country))
    self.i0=data[-1]
    optimal = minimize(loss, [0.001, 0.001], args=(data, self.s_0, self.i_0, self.r_0), method='L-BFGS-B', bounds=[(0.00000001, 1.75), (0.00000001, 0.899)])
    beta, gamma = optimal.x
    self.Beta, self.Gamma = optimal.x
    new_index, extended_actual, prediction = self.predict(beta, gamma, data, self.country, self.s_0, self.i_0, self.r_0)
    df = pd.DataFrame({'Infectados': extended_actual, 'Susceptibles': prediction.y[0], 'Recuperados': prediction.y[2]}, index=new_index)
    fig, ax = plt.subplots(figsize=(12, 7))
    ax.set_title(self.country)
    df.plot(ax=ax)
    self.r_0=(beta/gamma)
    #print(f"\n beta={beta:.8f}, gamma={gamma:.8f}, R0:{(beta/gamma):.8f}")

contadiadosA = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv')
contadiadosA = contadiadosA[contadiadosA['Country/Region'] == 'Ecuador'].iloc[0].loc[: ]

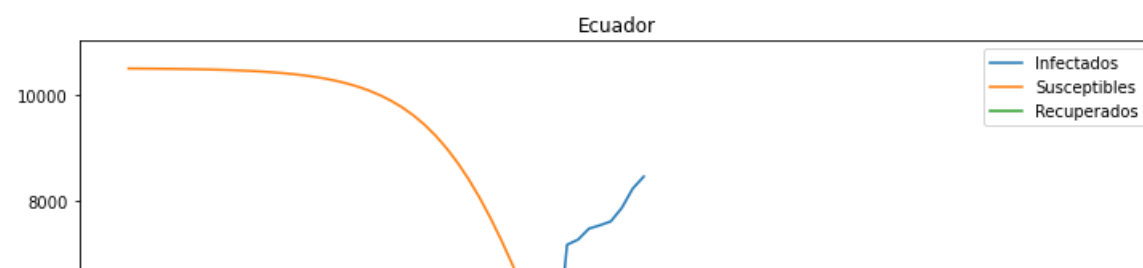
def loss(point, data, s_0, i_0, r_0):
    size = len(data)
    beta, gamma = point
    def SIR(t, y):
        S = y[0]
        I = y[1]
        R = y[2]
        return [-beta*S*I, beta*S*I-gamma*I, gamma*I]
    solution = solve_ivp(SIR, [0, size], [s_0, i_0, r_0], t_eval=np.arange(0, size, 1), vectorized=True)
    return np.sqrt(np.mean((solution.y[1] - data)**2))

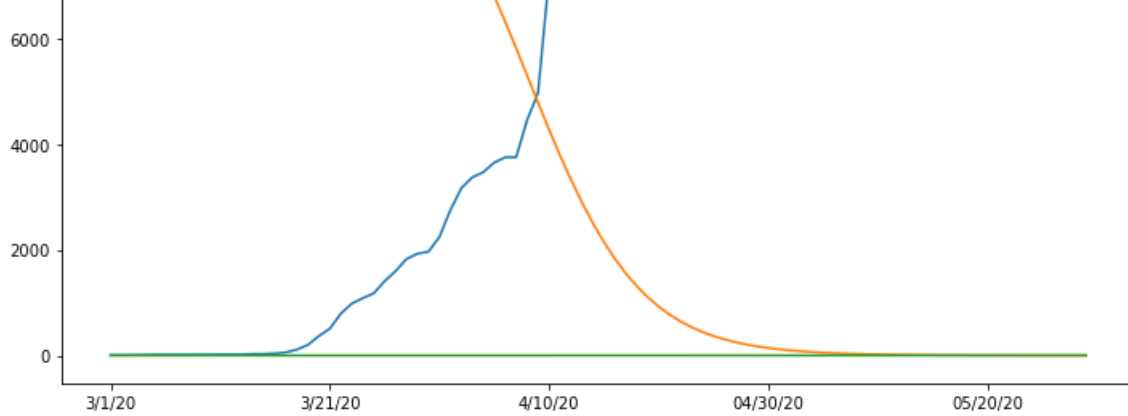
N=10500
i0=6
r0=0
s0=N - i0 - r0

print('Sin cuarentena')
Sin_cuarentena = Learner('Ecuador', loss, '3/1/20', 90, s0, i0, r0) #R0 cuando el pais no estaba en cuarentena
Sin_cuarentena.train()
Sin_cuarentena=(Sin_cuarentena.r_0)
print('Sin cuarentena:', Sin_cuarentena)

```

Sin cuarentena
Sin_cuarentena: 18.232518658414335





In []:

```
from random import randrange # Obtener un numero randomico
import pygame

#Parametros de inicio
Dias=7
PROBA_MUERTE = 8.4 # Probabilidad de que la gente muera COVID
CONTAGION_RATE = Sin_cuarentena # Factor R0 para la simulacion COVID probabilidad
PROBA_INFECT = CONTAGION_RATE * 10
PROBA_VACU = 0 # Probabilidad de que exista una vacuna, COVID = 0
SIMULACION_SPEED = Dias*25 # Tiempo de un dia en milisegundos (Cada 25 es un dia)
nb_rows = 45 #Numero de filas
nb_cols = 45 #Numero de columnas

global display, myfont, states, states_temp #Declaracion de variables globales

#Declaro colores en formato RGB
WHITE = (255, 255, 255)
BLUE = (0, 0, 255)
GREEN = (0, 247, 0)
BLACK = (0, 0, 0)

#Obtiene los vecinos dado un punto x,y
def get_vecinos(x, y):
    incx = randrange(3)
    incy = randrange(3)
    incx = (incx * 1) - 1
    incy = (incy * 1) - 1
    x2 = x + incx
    y2 = y + incy
    #Validar limites
    if x2 < 0:
        x2 = 0
    if x2 >= nb_cols:
        x2 = nb_cols - 1
    if y2 < 0:
        y2 = 0
    if y2 >= nb_rows:
        y2 = nb_rows - 1
    return [x2, y2] # Nuevos contagiados

#Genero las personas que cuentan con inmunidad o vacuna
def vacunar():
    for x in range(nb_cols):
        for y in range(nb_rows):
            if randrange(99) < PROBA_VACU:
                states[x][y] = 1

#Funcion que permite contar el numero de muertos de la matriz states == -1
def contar_muertes():
    contador = 0
    for x in range(nb_cols):
        for y in range(nb_rows):
            if states[x][y] == -1:
                contador += 1
    return contador

#Definimos datos de inicio
states = [[0] * nb_cols for i1 in range(nb_rows)]
states_temp = states.copy()
states[randrange(50)][randrange(50)] = 10 # Estado inicial de la simulacion Posicion del Infectado
it = 0 # Variable para contar las iteraciones
total_muerte = 0 # Contabiliza el numero de muertos
vacunar() #Llamar a la funcion vacunar
```

```

pygame.init() #Inicializo el motor de juegos pygame
pygame.font.init() #Inicializo el tipo de letra
display=pygame.display.set_mode((800,750),0,32) #Tamaño de la ventana
pygame.display.set_caption("Simulacion de Epidemia Covid-19 Ecuador")# Titulo
font=pygame.font.SysFont('Calibri', 40) # Tipo de letra
display.fill(WHITE) # Color de fondo

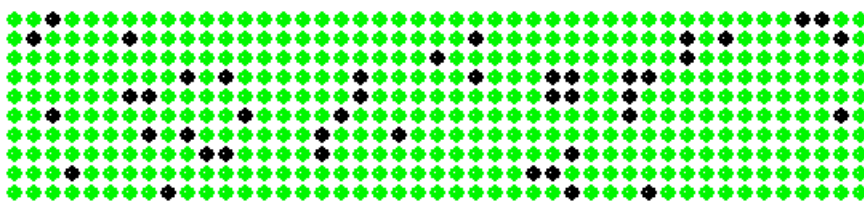
while True:
    pygame.time.delay(SIMULACION_SPEED) # Sleep o pausa
    it = it + 1
    if it <= 10000 and it >= 2:
        states_temp = states.copy() #Copia de la matriz
        #Recorrera la matriz
        for x in range(nb_cols):
            for y in range(nb_rows):
                state = states[x][y]
                if state == -1:
                    pass
                if state >= 10: # Numero de días de contagio
                    states_temp[x][y] = state + 1
                if state >= 20:
                    if randrange(99) < PROBA_MUERTE: # Genero un randomico para verificar si fallece o se recupera
                        states_temp[x][y] = -1 # Muere
                    else:
                        states_temp[x][y] = 1 # Cura o recupera
                if state >= 10 and state <= 20: # Rango de infectado
                    if randrange(99) < PROBA_INFECT: # Infecto a las personas cercanas entre 10 y 20
                        neighbour = get_vecinos(x, y) #Obtenemos los vecinos a contagiar
                        x2 = neighbour[0]
                        y2 = neighbour[1]
                        neigh_state = states[x2][y2]
                        if neigh_state == 0: #Verifico que este sano
                            states_temp[x2][y2] = 10 # Contagia
            states = states_temp.copy()
            total_muerte = contar_muertes() # contar el numero de muertos

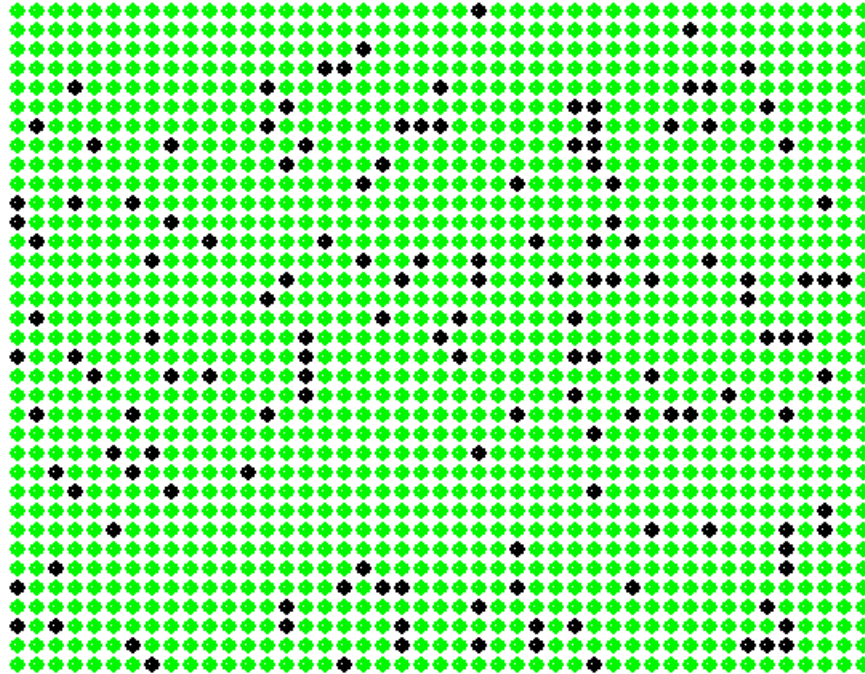
    pygame.draw.rect(display, WHITE, (250, 30, 260, 50)) # Grafico el fondo
    textsurface = font.render("Total muertes: "+ str(total_muerte), False, (255,160,122)) #El numero de muertos
    display.blit(textsurface, (250, 30)) # Graficar el texto de muertes
    #Graficar el estado del paciente matriz
    for x in range(nb_cols):
        for y in range(nb_rows):
            if states[x][y] == 0:
                color = BLUE # No infectado
            if states[x][y] == 1:
                color = GREEN # Recupero
            if states[x][y] >= 10:
                color = (states[x][y] * 12, 50, 50) # Inyectado - Rojo
            if states[x][y] == -1:
                color = BLACK # Muerto
            pygame.draw.circle(display, color, (100 + x * 12 + 5, 100 + y * 12 + 5), 5)
            pygame.draw.rect(display, WHITE, (100 + x * 12 + 3, 100 + y * 12 + 4, 1, 1))
    #Escuchar los eventos del teclado
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE: #Presiona y Escape
            pygame.quit() #Termino simulacion
        if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE: #Presiona y espacio
            #Reiniciamos valores
            states = [[0] * nb_cols for i1 in range(nb_rows)]
            states_temp = states.copy()
            states[5][5] = 10
            it = 0
            total_muerte = 0
            vacunar() #Llamar a la funcion vacunar

    pygame.display.update()# Mandar actualizar la ventana

```

Total muertes: 100





In [3]:

```
import numpy as np
import pandas as pd
from csv import reader
from csv import writer
from scipy.integrate import solve_ivp
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from datetime import timedelta, datetime
import argparse
import sys
import json
import ssl
import urllib.request
class Learner(object):
    def __init__(self, country, loss, start_date, predict_range,s_0, i_0, r_0):
        self.country = country
        self.loss = loss
        self.start_date = start_date
        self.predict_range = predict_range
        self.s_0 = s_0
        self.i_0 = i_0
        self.r_0 = r_0
        self.Beta = 0
        self.Gamma = 0

    def load_confirmed(self, country):
        df = pd.read_csv("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv")
        country_df = df[df['Country/Region'] == country]
        return country_df.iloc[0].loc[self.start_date:]

    def extend_index(self, index, new_size):
        values = index.values
        current = datetime.strptime(index[-1], '%m/%d/%y')
        while len(values) < new_size:
            current = current + timedelta(days=1)
            values = np.append(values, datetime.strptime(current, '%m/%d/%y'))
        return values

    def predict(self, beta, gamma, data, country, s_0, i_0, r_0):
        new_index = self.extend_index(data.index, self.predict_range)
        size = len(new_index)
        def SIR(t, y):
            S = y[0]
            I = y[1]
            R = y[2]
            return [-beta*S*I, beta*S*I-gamma*I, gamma*I]
        extended_actual = np.concatenate((data.values, [None] * (size - len(data.values))))
        return new_index, extended_actual, solve_ivp(SIR, [0, size], [s_0,i_0,r_0], t_eval=np.arange(0, size, 1))

    def train(self):
        data = (self.load_confirmed(self.country))
        self.i0=data[-1]
```

```

    optimal = minimize(loss, [0.001, 0.001], args=(data, self.s_0, self.i_0, self.r_0), method='L-BFGS-B', bounds=[(0.00000001, 1.75), (0.00000001, 0.899)])
    beta, gamma = optimal.x
    self.Beta, self.Gamma = optimal.x
    new_index, extended_actual, prediction = self.predict(beta, gamma, data, self.country, self.s_0, self.i_0, self.r_0)
    df = pd.DataFrame({'Infectados': extended_actual, 'Susceptibles': prediction.y[0], 'Recuperados': prediction.y[2]}, index=new_index)
    fig, ax = plt.subplots(figsize=(12, 7))
    ax.set_title(self.country)
    df.plot(ax=ax)
    self.r_0=(beta/gamma)*1000
    #print("\n beta={beta:.8f}, gamma={gamma:.8f}, R0: {(beta/gamma):.8f}")

contadiadosA = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv')
contadiadosA = contadiadosA[contadiadosA['Country/Region'] == 'Ecuador'].iloc[0].loc[: ]

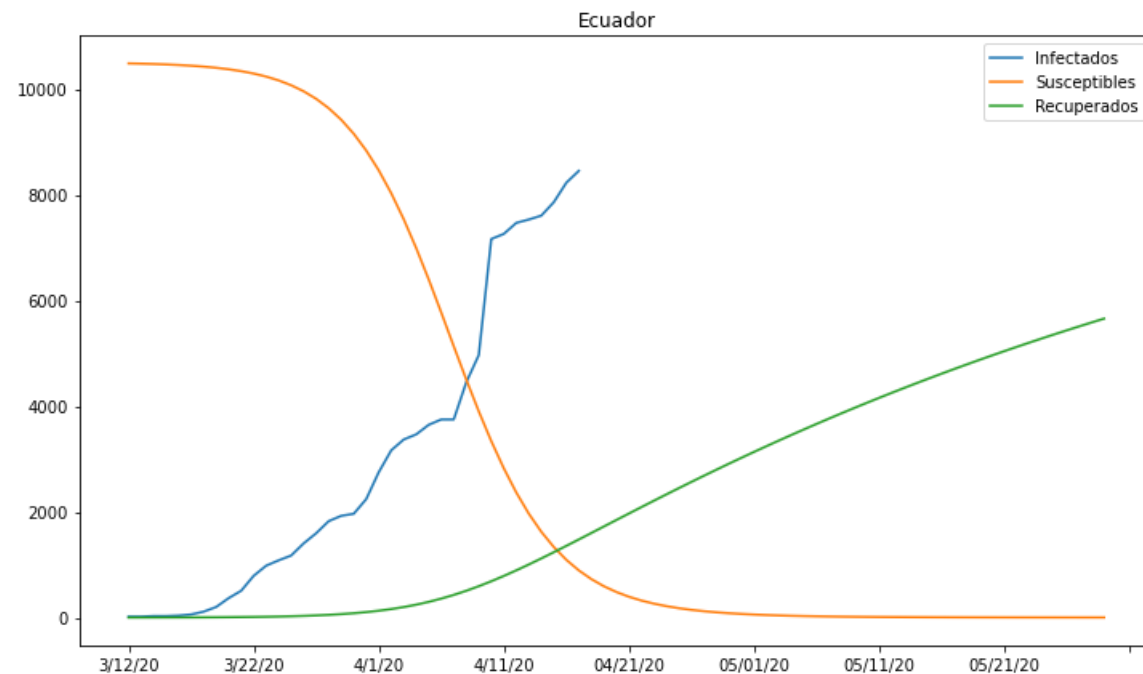
def loss(point, data, s_0, i_0, r_0):
    size = len(data)
    beta, gamma = point
    def SIR(t, y):
        S = y[0]
        I = y[1]
        R = y[2]
        return [-beta*S*I, beta*S*I-gamma*I, gamma*I]
    solution = solve_ivp(SIR, [0, size], [s_0, i_0, r_0], t_eval=np.arange(0, size, 1), vectorized=True)
    return np.sqrt(np.mean((solution.y[1] - data)**2))

N=10500
i0=contadiadosA[-1]
r0=0
s0=N - i0 - r0

print('cuarentena')
cuarentena = Learner('Ecuador', loss, '3/12/20', 79, 10483, 17, 0) #pais empieza la cuarentena
cuarentena.train()
cuarentena=cuarentena.r_0
print('cuarentena:', cuarentena)

```

cuarentena
cuarentena: 1.6667131969275573



In []:

```

from random import randrange # Obtener un numero randomico
import pygame

#Parametros de inicio
Dias=7
PROBA_MUERTE = 8.4 # Probabilidad de que la gente muera COVID
CONTAGION_RATE =cuarentena# Factor R0 para la simulacion COVID probabilidad
PROBA_INFECT = CONTAGION_RATE * 10
PROBA_VACU = 0 # Probabilidad de que exista una vacuna, COVID = 0
SIMULACION_SPEED = Dias*25 # Tiempo de un dia en milisegundos (Cada 25 es un dia)
nb_rows = 45 #Numero de filas
nb_cols = 45 #Numero de columnas

```

```
nb_cols = 45 #Numero de columnas
global display, myfont, states, states_temp #Declaracion de variables globales
```

```
#Declaro colores en formato RGB
```

```
WHITE = (255, 255, 255)
```

```
BLUE = (0, 0, 255)
```

```
GREEN = (0, 247, 0)
```

```
BLACK = (0, 0, 0)
```

```
#Obtiene los vecinos dado un punto x,y
```

```
def get_vecinos(x, y):
```

```
    incx = randrange(3)
```

```
    incy = randrange(3)
```

```
    incx = (incx * 1) - 1
```

```
    incy = (incy * 1) - 1
```

```
    x2 = x + incx
```

```
    y2 = y + incy
```

```
    #Validar limites
```

```
    if x2 < 0:
```

```
        x2 = 0
```

```
    if x2 >= nb_cols:
```

```
        x2 = nb_cols - 1
```

```
    if y2 < 0:
```

```
        y2 = 0
```

```
    if y2 >= nb_rows:
```

```
        y2 = nb_rows - 1
```

```
    return [x2, y2] # Nuevos contagiados
```

```
#Genero las personas que cuentan con inmunidad o vacuna
```

```
def vacunar():
```

```
    for x in range(nb_cols):
```

```
        for y in range(nb_rows):
```

```
            if randrange(99) < PROBA_VACU:
```

```
                states[x][y] = 1
```

```
#Funcion que permite contar el numero de muertos de la matriz states == -1
```

```
def contar_muertes():
```

```
    contador = 0
```

```
    for x in range(nb_cols):
```

```
        for y in range(nb_rows):
```

```
            if states[x][y] == -1:
```

```
                contador += 1
```

```
    return contador
```

```
#Definimos datos de inicio
```

```
states = [[0] * nb_cols for i1 in range(nb_rows)]
```

```
states_temp = states.copy()
```

```
states[randrange(50)][randrange(50)] = 10 # Estado inicial de la simulacion Posicion del Infectado
```

```
it = 0 # Variable para contar las iteraciones
```

```
total_muerte = 0 # Contabiliza el numero de muertos
```

```
vacunar() #Llamar a la funcion vacunar
```

```
pygame.init() #Incializo el motor de juegos pygame
```

```
pygame.font.init() #Inicializo el tipo de letra
```

```
display=pygame.display.set_mode((800,750),0,32) #Tamaño de la ventana
```

```
pygame.display.set_caption("Simulacion de Epidemia Covid-19 Ecuador")# Titulo
```

```
font=pygame.font.SysFont('Calibri', 40) # Tipo de letra
```

```
display.fill(WHITE) # Color de fondo
```

```
while True:
```

```
    pygame.time.delay(SIMULACION_SPEED) # Sleep o pausa
```

```
    it = it + 1
```

```
    if it <= 10000 and it >= 2:
```

```
        states_temp = states.copy() #Copia de la matriz
```

```
        #Recorrera la matriz
```

```
        for x in range(nb_cols):
```

```
            for y in range(nb_rows):
```

```
                state = states[x][y]
```

```
                if state == -1:
```

```
                    pass
```

```
                if state >= 10: # Numero de días de contagio
```

```
                    states_temp[x][y] = state + 1
```

```
                if state >= 20:
```

```
                    if randrange(99) < PROBA_MUERTE: # Genero un randomico para verificar si fallece o se recupera
```

```
                        states_temp[x][y] = -1 # Muere
```

```
                    else:
```

```
                        states_temp[x][y] = 1 # Cura o recupera
```

```
                if state >= 10 and state <= 20: # Rango de infectado
```

```
                    if randrange(99) < PROBA_INFECT: # Infecto a las personas cercanas entre 10 y 20
```

```
                        neighbour = get_vecinos(x, y) #Obtenemos los vecinos a contagiar
```

```
                        x2 = neighbour[0]
```

```
                        y2 = neighbour[1]
```



```

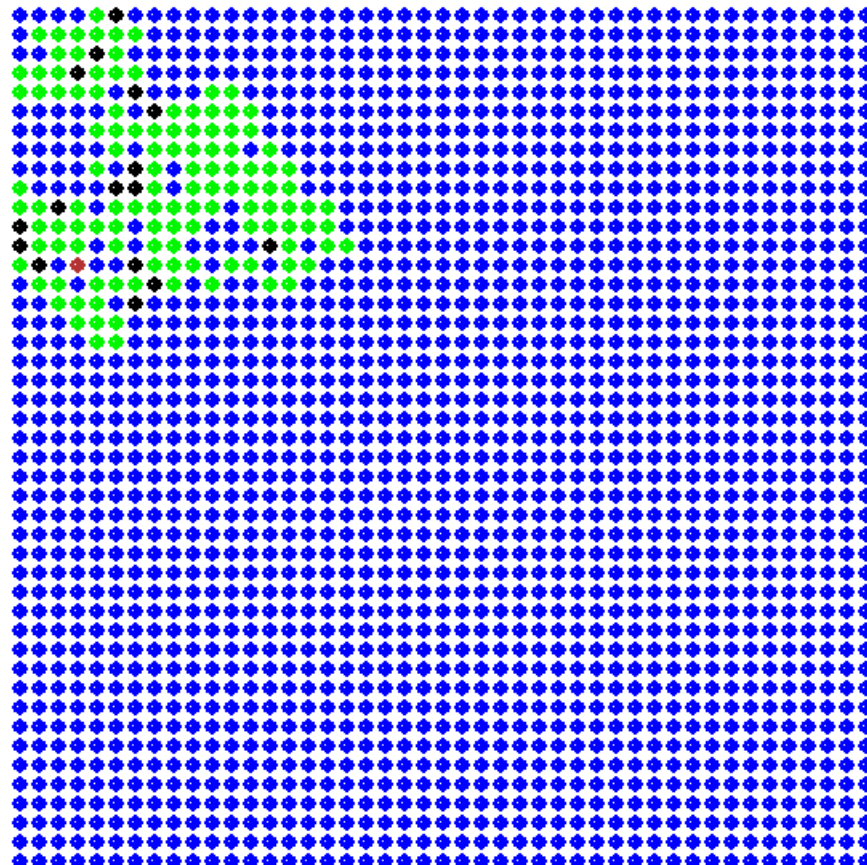
neigh_state = states[x2][y2]
if neigh_state == 0: #Verifico que este sano
    states_temp[x2][y2] = 10 # Contagia
states = states_temp.copy()
total_muerte = contar_muertes() # contar el numero de muertos

pygame.draw.rect(display, WHITE, (250, 30, 260, 50)) # Grafico el fondo
textsurface = font.render("Total muertes: "+ str(total_muerte), False, (255,160,122)) #El numero de muertes
display.blit(textsurface, (250, 30)) # Graficar el texto de muertes
#Graficar el estado del paciente matriz
for x in range(nb_cols):
    for y in range(nb_rows):
        if states[x][y] == 0:
            color = BLUE # No infectado
        if states[x][y] == 1:
            color = GREEN # Recupero
        if states[x][y] >= 10:
            color = (states[x][y] * 12, 50, 50) # Inyectado - Rojo
        if states[x][y] == -1:
            color = BLACK # Muerto
        pygame.draw.circle(display, color, (100 + x * 12 + 5, 100 + y * 12 + 5), 5)
        pygame.draw.rect(display, WHITE, (100 + x * 12 + 3, 100 + y * 12 + 4, 1, 1))
#Escuchar los eventos del teclado
for event in pygame.event.get():
    if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE: #Presiona y Escape
        pygame.quit() #Termino simulacion
    if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE: #Presiona y espacio
        #Reiniciamos valores
        states = [[0] * nb_cols for i1 in range(nb_rows)]
        states_temp = states.copy()
        states[5][5] = 10
        it = 0
        total_muerte = 0
        vacunar() #Llamar a la funcion vacunar

pygame.display.update()# Mandar actualizar la ventana

```

Total muertes: 10



Analysis

Es notable que en cada simulación, y gráfica SIR, cambia y varían los datos, debido a las valor de R_0 , tomando en cuenta lo que representa R_0 , el

valor que tiene en la la semana actual ya es un medio-alto, el valor para despues de una semana varia no es no mucho, en cuarentena y sin cuarenta cambia mucho, ya que sin cuarentena es donde más creció a tasa de contagio debido al contacto. Y con los valores 1.4 es mínimo ya que representa el valor más pequeño de R_0 y con 4, representa el valor máximo R_0 .

Conclusiones

Debido a la poca información que se cuenta para hacer este tipo de simulaciones, los datos no son verídicos, pero se debe considerar, ya que a pesar de este inconveniente, la tasa de contagio es alta sobretodo si analizamos cuando el país no estaba en cuarentena ya que supera el máximo de R_0 , y esto se debe a que muchas de las personas no sabían que ya estaban contagiados por el contacto. Se supone que ahora que ya estamos en cuarentena el valor debería bajar, pero no es así, por los círculos de contagios ya creados. También en las simulaciones se puede observar como la tasa de contagios va creciendo o no según el valor de R_0 y mostrando a su vez de manera interactiva la cantidad de muertos que puede haber.

Opinion

Lo que está pasando en el país se debe por no actuar oportunamente y de manera rápida, ya que se esperó demasiado tiempo para poner en cuarentena al país, entonces el círculo de contagio por el contacto con las personas creció demasiado rápido, hasta llegar a la situación que está ahora, donde ya no se puede controlar las zonas contagiadas ya que no se sabe quién está contagiado y quién no. Pero, se espera que ahora ya estando todos aislados, se vaya recuperando este control.