In [9]:
```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import pandas as pd
from scipy.integrate import solve_ivp
from scipy.optimize import minimize
from datetime import timedelta, datetime
from scipy.optimize import curve_fit
import plotly.graph_objects as go
from statsmodels.tsa.api import SimpleExpSmoothing
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from scipy.integrate import solve_ivp
from scipy.optimize import minimize
from scipy.integrate import odeint
from random import randrange # Obtener un numero randomico
import pygame
from IPython.display import Image
```

# Universidad Politecnica Salesiana

## Simulacion SIR

**Marcela Zhagüi**

# Practica

En concecuencia, generar 5 simulaciones:

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 evaluan con los las acciones de la cuarentena.

Finalmente, agregar el numero de dias transcurridos, personas recuperadas y generar la curva SIR de las simulaciones.

Puntos extras: Plantee y realize mejoras al modelo de simulacion.

# Implementar

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

# Modelo Exponencia;

In [2]:
```
url = 'C:\\Users\\Usuario\\Documents\\Simulacion\\NUEVO\\SimulacionPython-master\

df = pd.read_csv(url)
df = df[df['Country/Region'].isin(['Ecuador'])] #Filtro la Informacion solo para
df = df.loc[:,['Date','Confirmed','Recovered']]
FMT = '%Y-%m-%d'
date = df['Date']
df['Date'] = date.map(lambda x : (datetime.strptime(x, FMT) - datetime.strptime('
df
```

Out[2]:

| | Date | Confirmed | Recovered |
|---|---|---|---|
| **34344** | 248 | 171433 | 149048.0 |
| **34345** | 249 | 171783 | 149048.0 |
| **34346** | 250 | 172508 | 149048.0 |
| **34347** | 251 | 173486 | 149048.0 |
| **34348** | 252 | 174907 | 154956.0 |
| **34349** | 253 | 175269 | 154956.0 |
| **34350** | 254 | 175711 | 154956.0 |
| **34351** | 255 | 176630 | 154956.0 |
| **34352** | 256 | 177513 | 154956.0 |
| **34353** | 257 | 178674 | 154956.0 |
| **34354** | 258 | 179627 | 154956.0 |
| **34355** | 259 | 180295 | 160639.0 |
| **34356** | 260 | 180676 | 160639.0 |
| **34357** | 261 | 181104 | 160639.0 |
| **34358** | 262 | 182250 | 160639.0 |
| **34359** | 263 | 183246 | 160639.0 |
| **34360** | 264 | 183840 | 160639.0 |
| **34361** | 265 | 184876 | 160639.0 |
| **34362** | 266 | 185643 | 164009.0 |
| **34363** | 267 | 185944 | 164009.0 |
| **34364** | 268 | 186436 | 164009.0 |
| **34365** | 269 | 187230 | 164009.0 |
| **34366** | 270 | 188138 | 164009.0 |
| **34367** | 271 | 189534 | 164009.0 |
| **34368** | 272 | 190909 | 164009.0 |
| **34369** | 273 | 192117 | 169804.0 |
| **34370** | 274 | 192685 | 169804.0 |
| **34371** | 275 | 193673 | 169804.0 |
| **34372** | 276 | 194876 | 169804.0 |

| | Date | Confirmed | Recovered |
|---|---|---|---|
| **34373** | 277 | 195884 | 169804.0 |
| **34374** | 278 | 196482 | 169804.0 |

# Predecir que va a ocurrir la proxima semana.

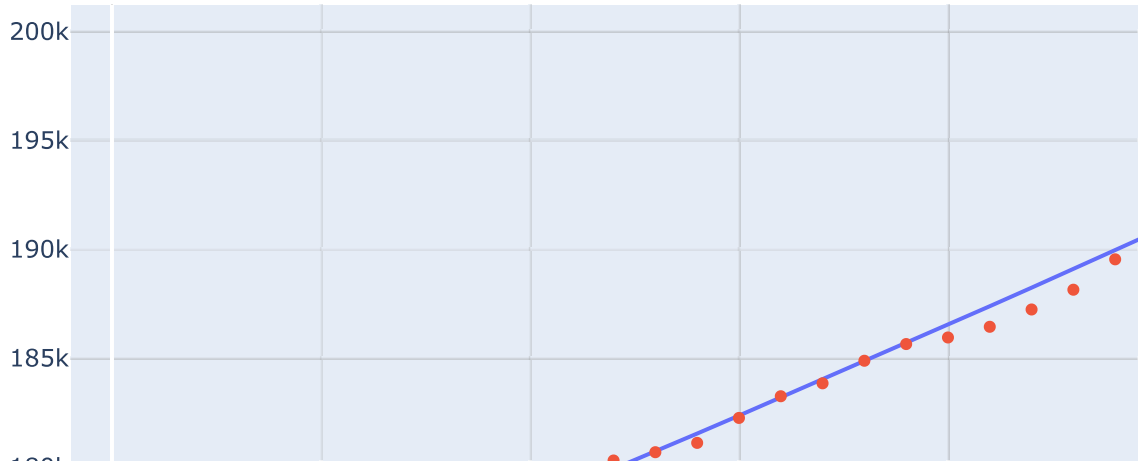
# Modelo exponencial

Type *Markdown* and LaTeX: $\alpha^2$

In [3]:
```
x=np.arange(1,len(df['Date'])+1,1)
y = df.iloc [:, 1]

def func(x, c, k):
    return (c*np.exp(k*x))
popt, pcov = curve_fit(func, x, y)
```

# Predecir para 10 dias

In [4]:
```
pred_x = np.array(list(range(min(x),max(x)+5)))
fig = go.Figure()
fig.add_trace(go.Scatter(y=func(pred_x ,*popt),x=pred_x,mode='lines',name='Predic
fig.add_trace(go.Scatter(y=y,x=x,mode='markers',name='Datos Reales'))
fig.show()
I00=func(pred_x ,*popt)[-1:]
I0=int( float(I00[0]) )
```

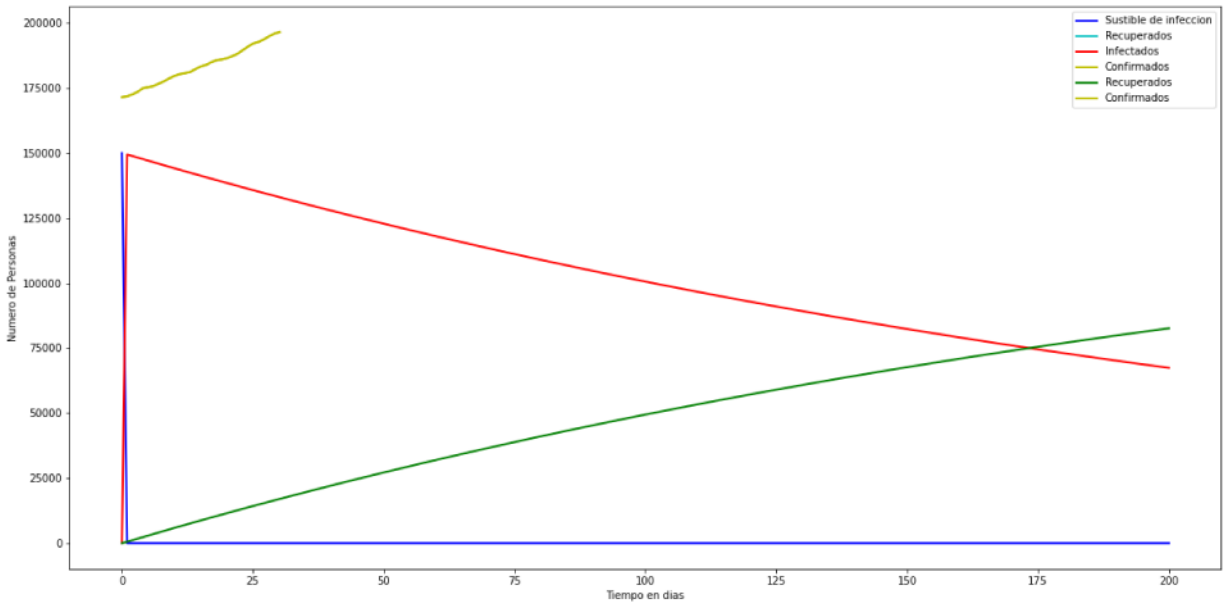



# Modelo SIR

# Implementar teniendo en cuenta los casos confirmados y recuperados.

In [5]:
```
def loss(point, datos, recovered,s0,i0, r0):
    size = len(datos)
    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], [s0,i0,r0], t_eval=np.arange(0, size, 1)
    l1 = np.sqrt(np.mean((solution.y[1] - datos)**2))
    l2 = np.sqrt(np.mean((solution.y[2] - recovered)**2))
    alpha =0.1
    return alpha * l1 + (1 - alpha) * l2

N=150000 #Numero habitantes del Ecuador
i0=15 #casos confirmados
r0=2 #Recuperados
s0= N - i0 - r0 #Resto de la poblacion
recovered=list(df.iloc [:, 2])
y=list(df.iloc [:, 1])
datos=list(df.iloc [:, 1])
d=[x1 - x2 for (x1, x2) in zip(datos,recovered)]
optimal = minimize(loss, [0.001, 0.001], args=(d, recovered, s0, i0, i0), method=
beta, gamma = optimal.x

R00=(beta/gamma)
new_index =y
size = len(new_index)
numero=len(func(pred_x ,*popt))-len(pred_x)
ea = np.concatenate((y, [None] * (size - len(y))))
er = np.concatenate((func(pred_x ,*popt)[:-numero], [None] * (size - len(func(pre


t = np.linspace(0, 200,200)
xa=t[0:len(df.iloc [:, 1])]
# Las ecuaciones diferenciales del modelo SIR..
def deriv(y, t, beta, gamma):
    S, I, R = y
    dSdt = -beta * S * I
    dIdt = beta * S * I- gamma * I
    dRdt = gamma * I
    return dSdt, dIdt, dRdt
y0 = s0,i0,r0 # Vector de condiciones iniciales

# Integre las ecuaciones SIR en la cuadrícula de tiempo, t. A traves de la funcio
ret = odeint(deriv, y0, t, args=(beta, gamma))

S, I, R = ret.T # Obtenicion de resultados

#Trazos en tres curvas separadas para S (t), I (t) y R (t)print(R0)
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111,  axisbelow=True)
ax.plot(t, S, 'b', alpha=1, lw=2, label='Sustible de infeccion')
ax.plot(xa, er, 'c', alpha=1, lw=2, label='Recuperados')
ax.plot(t, I, 'r', alpha=1, lw=2, label='Infectados')
ax.plot(xa, y, 'y', alpha=1, lw=2, label='Confirmados')
```

```
ax.plot(t, R, 'g', alpha=1, lw=2, label='Recuperados')
ax.plot(xa, y, 'y', alpha=1, lw=2, label='Confirmados')

ax.set_xlabel('Tiempo en dias')
ax.set_ylabel('Numero de Personas')
legend = ax.legend()
```



In [9]:

```
#Parametros de inicio
PROBA_MUERTE = 8.4  # Probabilidad de que la gente muera COVID
CONTAGION_RATE = R00  # 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 = 50 # Tiempo de un dia en milisegundos (Cada 25 es un dia)
nb_rows = 50 #Numero de filas
nb_cols = 50 #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()
```

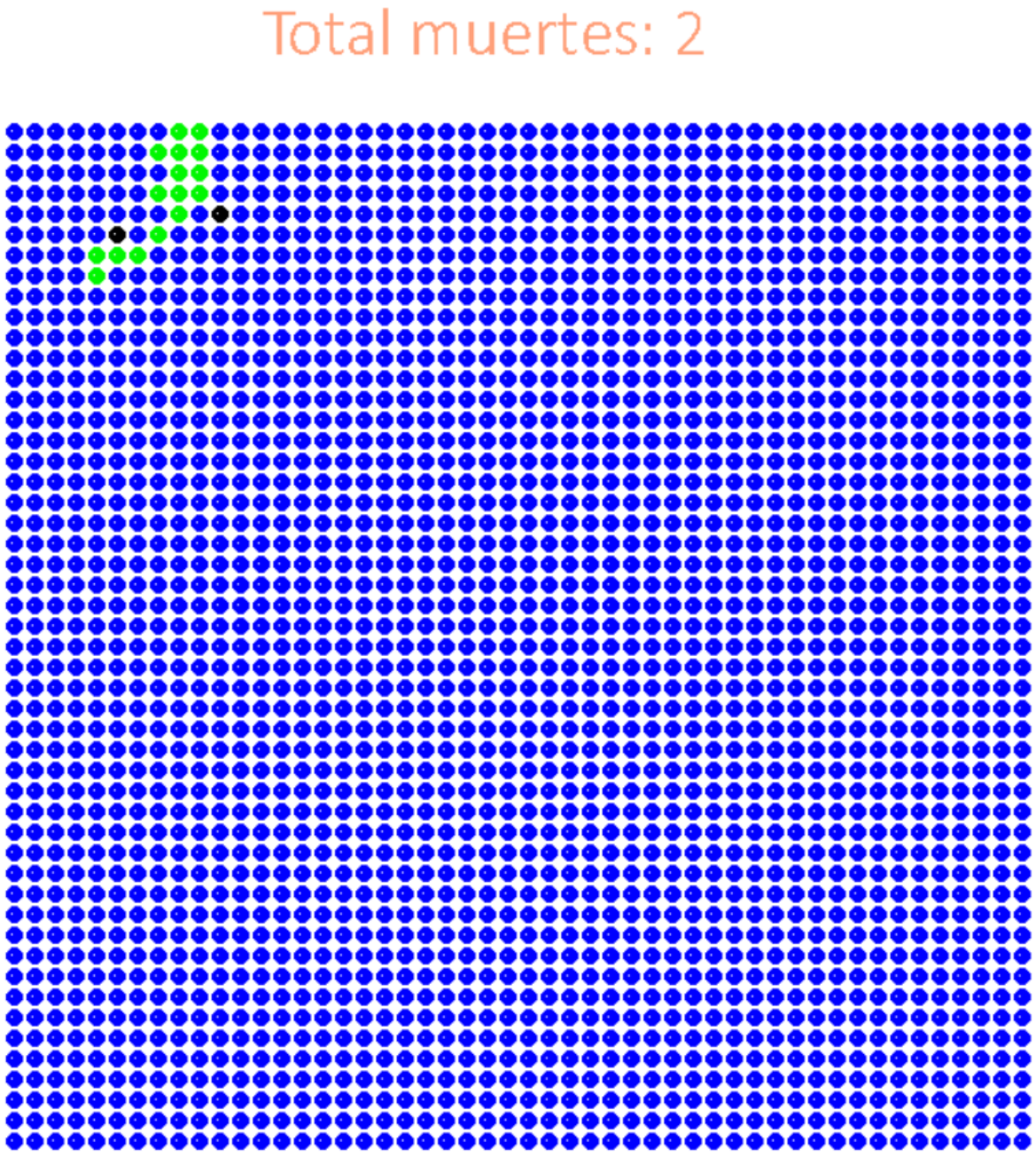
```
states[randrange(50)][randrange(50)] = 10 # Estado inicial de la simulacion Posic
it = 0 # Variable para contar las Iteraciones
total_muerte = 0 # Contabiliza el numero de muertos
vacunar() #Llamar a la funcion vacunar

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) #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 v
                        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 cer
                        neighbour = get_vecinos(x, y) #Obtenemos los vecinos a co
                        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, (255,1
    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) # Injectado - Rojo
            if states[x][y] == -1:
                color = BLACK # Muerto
            pygame.draw.circle(display, color, (100 + x * 12 + 5, 100 + y * 12 +
            pygame.draw.rect(display, WHITE, (100 + x * 12 + 3, 100 + y * 12 + 4,
```

```
    #Escuachar los eventos del teclado
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE: #Presio
            pygame.quit() #Termino simulacion
        if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE: #Presior
            #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
```

```
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
<ipython-input-9-56bc27d54845> in <module>
    124             vacunar() #Llamar a la funcion vacunar
    125
--> 126     pygame.display.update()# Mandar actualizar la ventana

error: video system not initialized
```

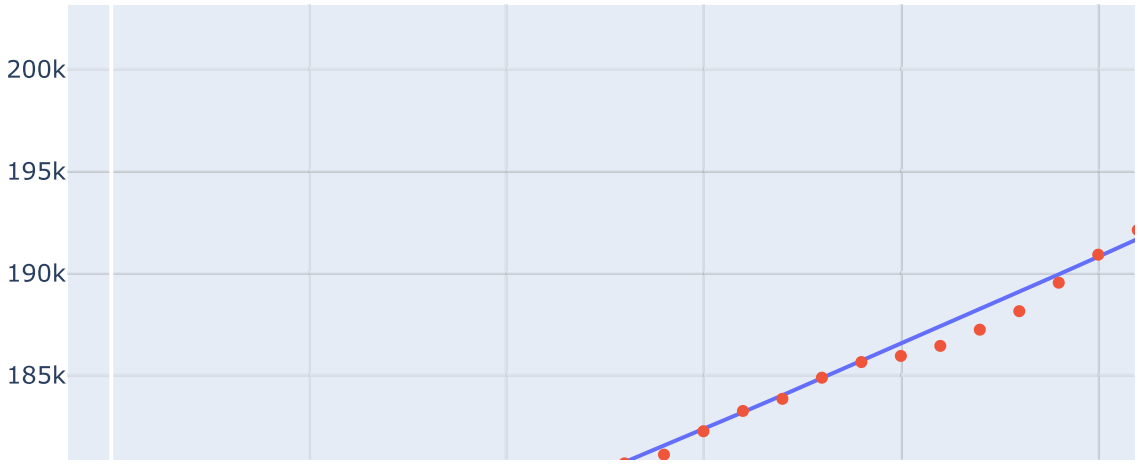
In [10]:
```
Image("punto1.png")
```

Out[10]:



# 2. Predecir que va a ocurrir la proxima semana.

In [6]:
```
#Modelo exponencial
x=np.arange(1,len(df['Date'])+1,1)
y = df.iloc [:, 1]

def func(x, c, k):
    return (c*np.exp(k*x))
popt, pcov = curve_fit(func, x, y)

pred_x = np.array(list(range(min(x),max(x)+7)))#Predecir para una semana
fig = go.Figure()
fig.add_trace(go.Scatter(y=func(pred_x ,*popt),x=pred_x,mode='lines',name='Predic
fig.add_trace(go.Scatter(y=y,x=x,mode='markers',name='Datos Reales'))
fig.show()
I00=func(pred_x ,*popt)[-1:]
I0=int( float(I00[0]) )
```




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

In [7]:
```
#Parametros de inicio
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 = 50 # Tiempo de un dia en milisegundos (Cada 25 es un dia)
nb_rows = 50 #Numero de filas
nb_cols = 50 #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 Posic
```

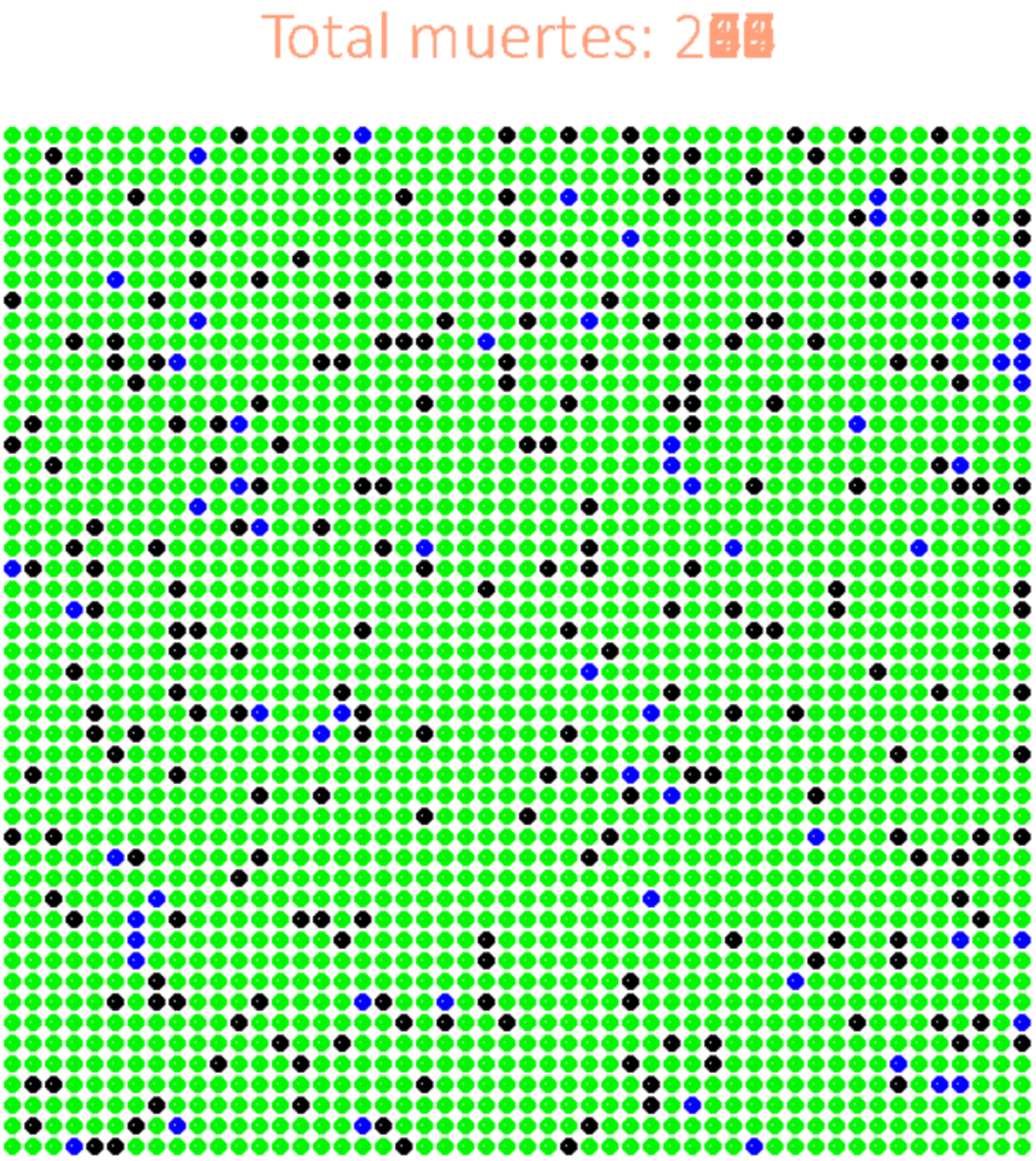
```
it = 0 # Variable para contar las Iteraciones
total_muerte = 0 # Contabiliza el numero de muertos
vacunar() #Llamar a la funcion vacunar

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) #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 v
                        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 cer
                        neighbour = get_vecinos(x, y) #Obtenemos los vecinos a co
                        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, (255,1
    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) # Injectado - Rojo
            if states[x][y] == -1:
                color = BLACK # Muerto
            pygame.draw.circle(display, color, (100 + x * 12 + 5, 100 + y * 12 +
            pygame.draw.rect(display, WHITE, (100 + x * 12 + 3, 100 + y * 12 + 4,
    #Escuachar los eventos del teclado
```

```
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE: #Presio
            pygame.quit() #Termino simulacion
        if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE: #Presior
            #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
```

```
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
<ipython-input-7-d31193e68071> in <module>
    124             vacunar() #Llamar a la funcion vacunar
    125
--> 126     pygame.display.update()# Mandar actualizar la ventana

error: video system not initialized
```

In [11]:
```
Image("punto3.png")
```

Out[11]:



# 4. El valor 1.4 en el mejor de los casos

In [7]:
```
#Parametros de inicio
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 = 50 # Tiempo de un dia en milisegundos (Cada 25 es un dia)
nb_rows = 50 #Numero de filas
nb_cols = 50 #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 Posic
```

```
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) #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 v
                        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 cer
                        neighbour = get_vecinos(x, y) #Obtenemos los vecinos a co
                        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, (255,1
    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:
                 # Recupero
                color = GREEN
            if states[x][y] >= 10:
                # Injectado - Rojo
                color = (states[x][y] * 12, 50, 50)
            if states[x][y] == -1:
                color = BLACK # Muerto
            pygame.draw.circle(display, color, (100 + x * 12 + 5, 100 + y * 12 +
```

```
            pygame.draw.rect(display, WHITE, (100 + x * 12 + 3, 100 + y * 12 + 4,
    #Escuachar los eventos del teclado
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
            pygame.quit() #Termino simulacion
        if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
            #Reiniciamos valores
            states = [[0] * nb_cols for i1 in range(nb_rows)]
            states_temp = states.copy()
            states[5][5] = 20
            it = 0
            total_muerte = 0
            vacunar() #Llamar a la funcion vacunar

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

```
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
<ipython-input-7-cac2352801c8> in <module>
    126             vacunar() #Llamar a la funcion vacunar
    127
--> 128     pygame.display.update()# Mandar actualizar la ventana

error: video system not initialized
```

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

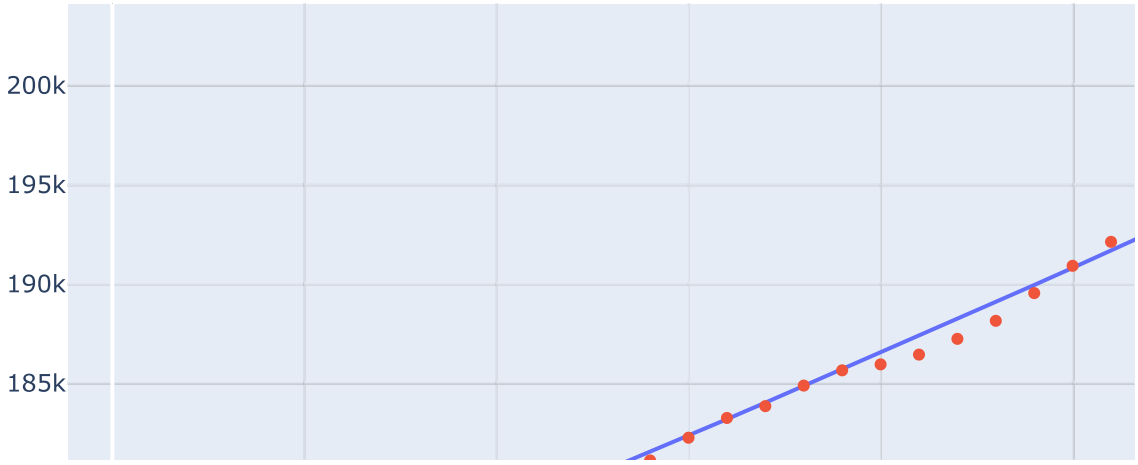
In [3]:
```
#Modelo exponencial
#sin cuarentena
x=np.arange(1,len(df['Date'])+1,1)
y = df.iloc [:, 1]



def func(x, c, k):
    return (c*np.exp(k*x))
popt, pcov = curve_fit(func, x, y)

#Predecir para una semana
pred_x = np.array(list(range(min(x),max(x)+8)))
fig = go.Figure()
fig.add_trace(go.Scatter(y=func(pred_x ,*popt),x=pred_x,mode='lines',name='Predic
fig.add_trace(go.Scatter(y=y,x=x,mode='markers',name='Datos Reales'))
fig.show()
I00=func(pred_x ,*popt)[-1:]
I0=int( float(I00[0]) )

#Modelo SIR

# 3. Implementar teniendo en cuenta los casos confirmados y recuperados.
def loss(point, datos, recovered,s0,i0, r0):
    size = len(datos)
    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], [s0,i0,r0], t_eval=np.arange(0, size, 1)
    l1 = np.sqrt(np.mean((solution.y[1] - datos)**2))
    l2 = np.sqrt(np.mean((solution.y[2] - recovered)**2))
    alpha =0.1
    return alpha * l1 + (1 - alpha) * l2
#Numero habitantes del Ecuador
N=150000
#casos confirmados
i0=20
#Recuperados
r0=5
#Resto de la poblacion
s0= N - i0 - r0
recovered=list(df.iloc [:, 2])

y = df.iloc [:, 1]


datos=list(df.iloc [:, 1])

d=[x1 - x2 for (x1, x2) in zip(datos,recovered)]
optimal = minimize(loss, [0.001, 0.001], args=(d, recovered, s0, i0, i0), method=
beta, gamma = optimal.x

R00=(beta/gamma)*1000
```

```
new_index =y
size = len(new_index)
ea = np.concatenate((y, [None] * (size - len(y))))
numero=len(func(pred_x ,*popt))-len(pred_x)
er = er = np.concatenate((func(pred_x ,*popt)[:-numero], [None] * (size - len(fur



t = np.linspace(0, 200,200)
xa=t[0:len(df.iloc [:, 1])]
# Las ecuaciones diferenciales del modelo SIR..
def deriv(y, t, beta, gamma):
    S, I, R = y
    dSdt = -beta * S * I
    dIdt = beta * S * I- gamma * I
    dRdt = gamma * I
    return dSdt, dIdt, dRdt
# Vector de condiciones iniciales
y0 = s0,i0,r0

# Integre las ecuaciones SIR en la cuadrícula de tiempo, t. A traves de la funcio
ret = odeint(deriv, y0, t, args=(beta, gamma))

S, I, R = ret.T # Obtenicion de resultados

#  Trace los datos en tres curvas separadas para S (t), I (t) y R (t)print(R0)

fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111,  axisbelow=True)
ax.plot(t, S, 'b', alpha=1, lw=2, label='Sustible de infeccion')
ax.plot(t, I, 'r', alpha=1, lw=2, label='Infectados')
ax.plot(t, R, 'g', alpha=1, lw=2, label='Recuperados')
ax.plot(xa, y, 'y', alpha=1, lw=2, label='Confirmados')
ax.plot(xa, er, 'c', alpha=1, lw=2, label='Recuperados')
ax.set_xlabel('Tiempo en dias')
ax.set_ylabel('Numero de Personas')
legend = ax.legend()

#Parametros de inicio
listadias = []
listainfectados = []
listarecuperados=[]
PROBA_MUERTE = 8.4  # Probabilidad de que la gente muera COVID
CONTAGION_RATE = R00  # 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 = 50 # Tiempo de un dia en milisegundos (Cada 25 es un dia)
nb_rows = 50 #Numero de filas
nb_cols = 50 #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)
```

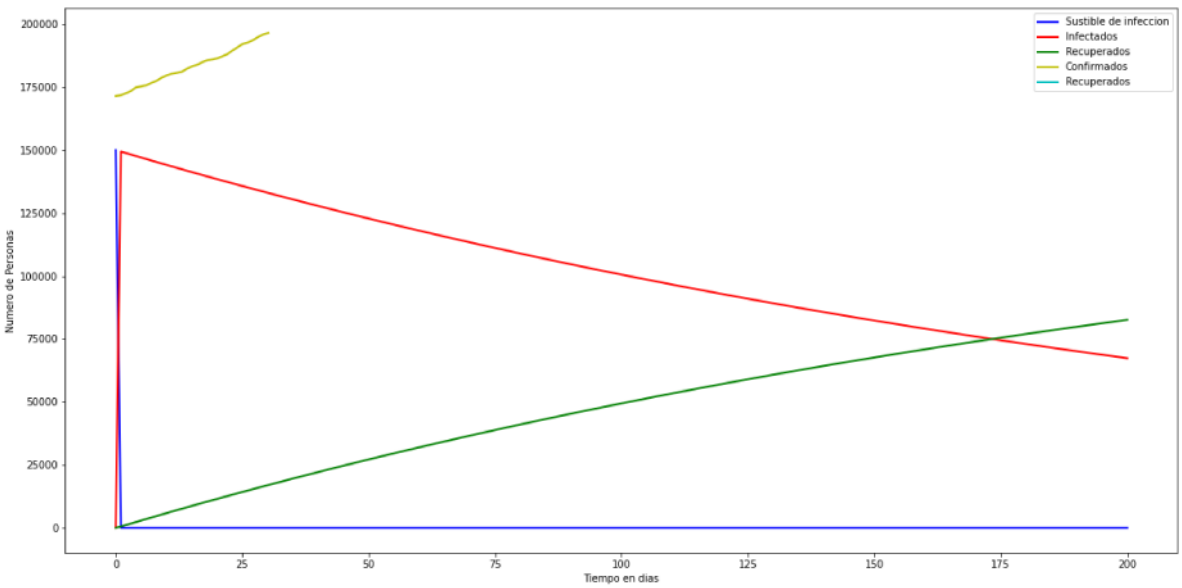
```
BLACK = (0, 0, 0)

#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 Posic
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) #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
cont=0
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 v
                        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 cer
                        neighbour = get_vecinos(x, y) #Obtenemos los vecinos a co
                        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
        cont +=  1
        listadias.append(cont)
    pygame.draw.rect(display, WHITE, (250, 30, 260, 50)) # Grafico el fondo
    textsurface = font.render("Total muertes: "+ str(total_muerte), False, (255,1
    display.blit(textsurface, (250, 30)) # Graficar el texto de muertes
    #Graficar el estado del paciente matriz
    con=0
    con2=0
    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
                con +=  1
                listarecuperados.append(con)
            if states[x][y] >= 10:
                color = (states[x][y] * 12, 50, 50) # Infectado - Rojo
                con2 +=  1
                listainfectados.append(con2)
            if states[x][y] == -1:
                color = BLACK # Muerto
            pygame.draw.circle(display, color, (100 + x * 12 + 5, 100 + y * 12 +
            pygame.draw.rect(display, WHITE, (100 + x * 12 + 3, 100 + y * 12 + 4,
    #Escuachar los eventos del teclado
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE: #Presio
            pygame.quit() #Termino simulacion
        if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE: #Presion
            #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
```




```
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
<ipython-input-3-d4537dd2d092> in <module>
    232             vacunar() #Llamar a la funcion vacunar
    233
--> 234     pygame.display.update()# Mandar actualizar la ventana
    235

error: video system not initialized
```




In [12]:
```
Image("SIMULACION5.png")
```

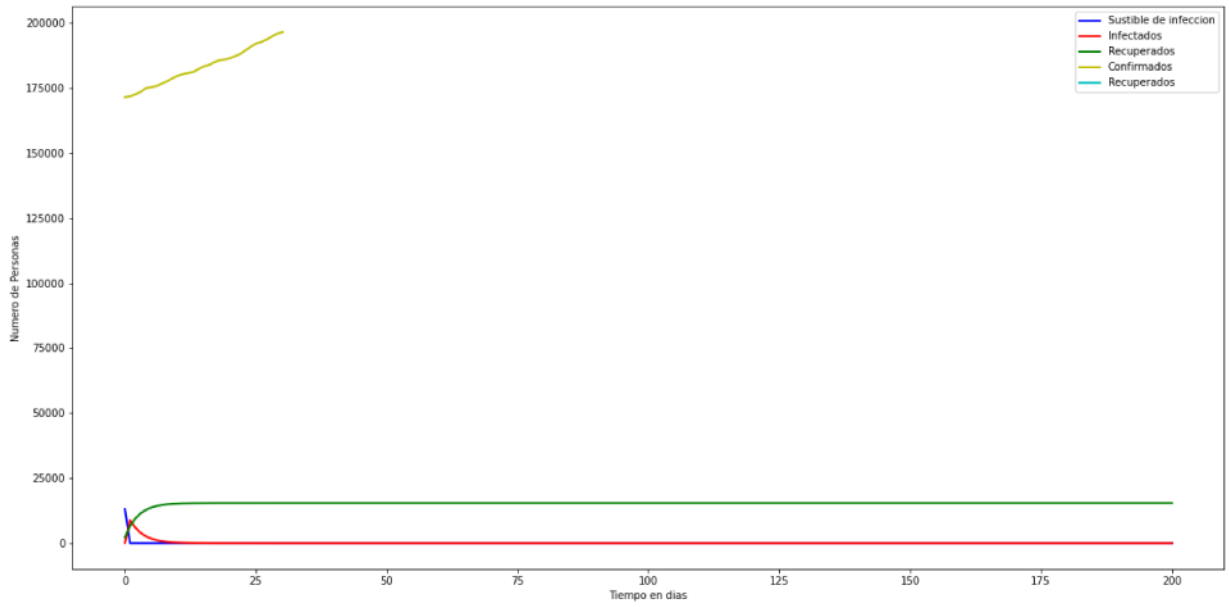
Out[12]:

# Agregar el numero de dias transcurridos, personas recuperadas y generar la curva SIR de las simulaciones.

In [5]:
```
print('Contagiados ',I0)
print('Recuperados ',con)
print('Numero de dias ',cont)

# 3. Implementar teniendo en cuenta los casos confirmados y recuperados.
def loss(point, datos, recovered,s0,i0, r0):
    size = len(datos)
    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], [s0,i0,r0], t_eval=np.arange(0, size, 1)
    l1 = np.sqrt(np.mean((solution.y[1] - datos)**2))
    l2 = np.sqrt(np.mean((solution.y[2] - recovered)**2))
    alpha =0.1
    return alpha * l1 + (1 - alpha) * l2
#Numero habitantes del Ecuador
N=15486
#casos confirmados
i0=100
#Recuperados
r0=con
#Resto de la poblacion
s0= N - i0 - r0
recovered=list(df.iloc [:, 2])
y=list(df.iloc [:, 1])
datos=list(df.iloc [:, 1])
d=[x1 - x2 for (x1, x2) in zip(datos,recovered)]
optimal = minimize(loss, [0.001, 0.001], args=(d, recovered, s0, i0, i0), method=
beta, gamma = optimal.x

R00=beta/gamma

new_index =y
size = len(new_index)
ea = np.concatenate((y, [None] * (size - len(y))))
numero=len(func(pred_x ,*popt))-len(pred_x)
er = er = np.concatenate((func(pred_x ,*popt)[:-numero], [None] * (size - len(fur



t = np.linspace(0, 200,200)
xa=t[0:len(df.iloc [:, 1])]
# Las ecuaciones diferenciales del modelo SIR..
def deriv(y, t, beta, gamma):
    S, I, R = y
    dSdt = -beta * S * I
    dIdt = beta * S * I- gamma * I
    dRdt = gamma * I
    return dSdt, dIdt, dRdt
# Vector de condiciones iniciales
y0 = s0,i0,r0
```

```
# Integre las ecuaciones SIR en la cuadrícula de tiempo, t. A traves de la funcic
ret = odeint(deriv, y0, t, args=(beta, gamma))

S, I, R = ret.T # Obtenicion de resultados

#  Trace los datos en tres curvas separadas para S (t), I (t) y R (t)print(R0)

fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111,  axisbelow=True)
ax.plot(t, S, 'b', alpha=1, lw=2, label='Sustible de infeccion')
ax.plot(t, I, 'r', alpha=1, lw=2, label='Infectados')
ax.plot(t, R, 'g', alpha=1, lw=2, label='Recuperados')
ax.plot(xa, y, 'y', alpha=1, lw=2, label='Confirmados')
ax.plot(xa, er, 'c', alpha=1, lw=2, label='Recuperados')
ax.set_xlabel('Tiempo en dias')
ax.set_ylabel('Numero de Personas')
legend = ax.legend()
```

```
Contagiados  202376
Recuperados  2272
Numero de dias  617
```




# Analisis

R0 < 1: no se da la epidemia por lo tanto la enfermedad se termina R0 > 1: la enfermedad se propaga R0=1.4: se mantiene en equilibrio,probabilidad de muerte baja R0=4: el nivel de contagio es EXCESIVO

# Conclusiones

Se determina la estimación del número productivo básico R0 en una epidemia, tiene gran peso pues dado esto se determina si la epidemia crece considerablemente al ser mayor que 1 decrece o se mantiene en equilibrio si R0<1, esto depende de cada país y el número de habitantes que tenga y la cantidad continua se dio el contagi.

## Opinion

El modelo SIR ayuda a determinar y observar la curva de crecimiento tanto de infectados como de recuperados muertes, por otra parte, la librería pygame es de gran ayuda para visualizar como ocurre el contagio.

Dado a que estos modelos no se apegan a la realidad, podemos simular o predecir ciertos criterios de evalucion o aproximacio, como esta el estado del pais o tener una idea proxima a la realidad, es decir podemos evaluar los datos en base a estos experimentos o bien modelos.

# Referencias


- http://code.intef.es/simulamos-una-epidemia-virica/ (http://code.intef.es/simulamos-una-epidemia-virica/)
-


In [ ]: