In [20]:

```
# PHY104b Project Prompt 5:
# Write a code which simulates the 'Game of Life'.
# Matt Lund (Working Independently)
%pylab inline
import matplotlib.pyplot as plt
import matplotlib.animation as animation
set_printoptions(threshold=sys.maxsize)
```

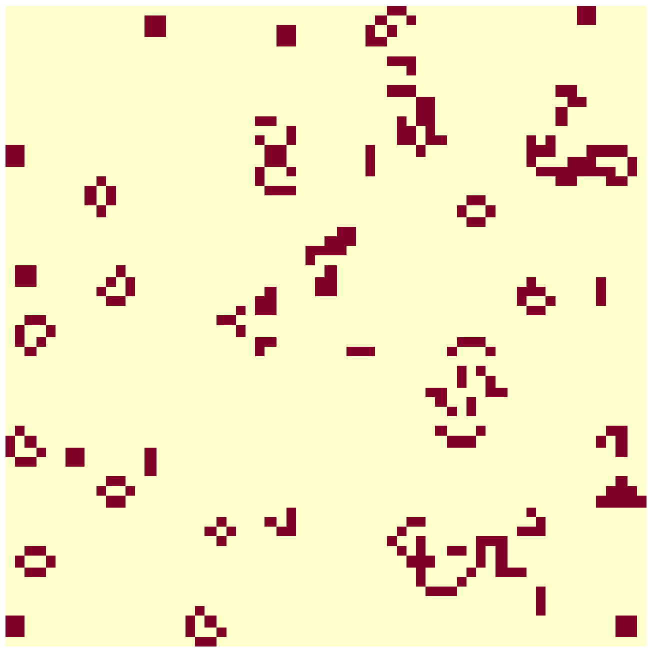
Populating the interactive namespace from numpy and matplotlib

In [42]:

```
N = 64 # size of game board

np.random.seed(123456789) # establish seed for consistent results

def random(N): # creates random matrix of size N with equal probability of cells being 1(alive) or 0(dead)
    board = np.random.choice([0,1],size=(N,N),p=[.5,.5])
    return board

def update(frame): # function that updates game of life board when called
    global board,img_plot # defines board and image frame as globabl variable for coding simplicity
    updateboard = np.array([]) # creates new board used to update existing board
    if(frame < 5): # pauses board for roughly half a second before animation starts (helpful visually)
        updateboard = board
    else:
        for i in range(N):
            updaterow = np.array([]) # update board will be appended row by row
            for j in range(N): # have to calculate number of neighbors to enforce game of life rules
                # there are many boundary conditions that must be accounted for to ensure accurate data
                # I have marked each boundary condition (each side and corner) with comments below:
                if((i+1)>=N):
                    if((j+1)>=N):
                        neighbors = ((board[i-1,j-1]) + (board[i-1,j]) + (board[i,j-1])) # for bottom right corner element
                    elif((j-1)<0):
                        neighbors = ((board[i-1,j]) + (board[i-1,j+1]) + (board[i,j+1])) # for bottom left corner element
                    else:
                        neighbors = ((board[i-1,j-1]) + (board[i-1,j]) + (board[i-1,j+1]) + (board[i,j-1]) +
                                     (board[i,j+1])) # for bottom row
                elif((j+1)>=N):
                    if((i-1)<0):
```

```python
                neighbors = ((board[i,j-1]) + (board[i+1,j-1]) + (board[i+1,j])) # for top right corner element
            else:
                neighbors = ((board[i-1,j-1]) + (board[i-1,j]) + (board[i,j-1]) +
                             (board[i+1,j-1]) + (board[i+1,j])) # for right-most column
        elif((j-1)<0):
            if((i-1)<0):
                neighbors = ((board[i,j+1]) + (board[i+1,j]) + (board[i+1,j+1])) # for top left corner element
            else:
                neighbors = ((board[i-1,j]) + (board[i-1,j+1]) + (board[i,j+1]) +
                             (board[i+1,j]) + (board[i+1,j+1])) # for left-most column
        elif((i-1)<0):
            neighbors = ((board[i,j-1]) + (board[i,j+1]) + (board[i+1,j-1]) +
                         (board[i+1,j]) + (board[i+1,j+1])) # for top row
        else:
            neighbors = ((board[i-1,j-1]) + (board[i-1,j]) + (board[i-1,j+1]) + (board[i,j-1]) +
                        (board[i,j+1]) + (board[i+1,j-1]) + (board[i+1,j]) + (board[i+1,j+1])) # every other box

        if((board[i,j])==1):
            if(neighbors < 2):
                # If cell is ON and has fewer than 2 neighbors that are ON, it turns OFF
                updaterow = np.append(updaterow,0)
            elif((neighbors == 2) or (neighbors == 3)):
                # If cell is ON and has 2 or 3 neighbors that are ON, it stays ON
                updaterow = np.append(updaterow,1)
            elif(neighbors > 3):
                # If cell is ON and has more than 3 neighbors that are ON, it turns OFF
                updaterow = np.append(updaterow,0)
        elif((board[i,j])==0):
            if(neighbors == 3):
                # If cell is OFF and has exactly 3 neighbors that are ON, it turns ON
                updaterow = np.append(updaterow,1)
            else:
                # If cell is OFF and has more/less than 3 neighbors that are ON, it stays OFF
                updaterow = np.append(updaterow,0)

    if(i==0):
        updateboard = np.append(updateboard,updaterow) # appends first row directly to update board
    elif(i>0):
        updateboard = np.vstack([updateboard,updaterow]) # appends each preceeding line row by row
board = updateboard # overwrites old board with new board
img_plot.set_data(updateboard) # creates an image frame of this board
```

```python
    return img_plot, # returns cell map as an image frame to be called in animate function

def animate(N): # function that animates game of life by creating a gif
    global board,img_plot # defines existing global variables
    board = random(N) # creates random board using existing function
    fig,ax=plt.subplots()
    img_plot=ax.imshow(board,interpolation='nearest',cmap='YlOrRd') # creates image
    plt.axis('off') # removes numbers on axes for cleaner look
    anim = animation.FuncAnimation(fig,frames=100,func=update,interval=100)
    # above line animates the board over 100 preceeding frames at 100ms intervals
    # using my update function. FuncAnimation found from animate.py file in canvas notes.
    plt.tight_layout() # allows  image to scale properly
    anim.save('conwaytest.gif') # saves animation frames to gif
    plt.show() # displays image
    return anim # returns gif
    # gif will be stored wherever your code saves, or if using jupyter notebook it will store there

animate(N) # Creates the animation in one line
```



Out[42]: <matplotlib.animation.FuncAnimation at 0x7d80336b3eb0>

In [ ]: