1. No.
If \( x_1 \), then by the second conjunct \( \neg x_3 \) and by the fourth conjunct \( x_2 \). Which contradicts the third conjunct.
And if \( \neg x_1 \), then by the first conjunct \( \neg x_2 \) and by the fifth conjunct \( x_3 \). Which contradicts the sixth conjunct.

2. The following formula expresses that, for given \( n \) and \( m \), each pigeon occupies a hole (to be more precise, occupies at least one hole):

\[
\Phi_{n,m} = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} x_{ij}
\]

The following formula expresses that each hole contains at most one pigeon:

\[
\Psi_{n,m} = \bigwedge_{1 \leq i < i' \leq n} \bigwedge_{j=1}^{m} (\neg x_{ij} \lor \neg x_{i'j})
\]

The overall (unsatisfiable) pigeonhole formula is \( \Phi_{n,m} \land \Psi_{n,m} \).

3. We can simply instantiate the construction presented on the slides.
Of the five tile types to build the first row, let \( t_1 \) be the one with top color \( q_0 \), \( a \), \( t_2 \) the one with top color \( q_0 \), \( b \), \( t_3 \) the one with top color \( a \), \( t_4 \) the one with top color \( b \), and \( t_5 \) the one with top color \( \Box \). The first row must consist of \( t_5 t_5 t_5 t_1 t_4 t_4 t_5 \): 

\[
x_{11t_5} \land x_{21t_5} \land x_{31t_5} \land x_{41t_1} \land x_{51t_4} \land x_{61t_4} \land x_{71t_5}
\]

Each position must hold at most one tile type:

\[
\bigwedge_{k=1}^{7} \bigwedge_{\ell=1}^{7} \bigwedge_{t \neq t'} \neg (x_{k\ell t} \land x_{k\ell t'})
\]

Horizontally touching sides must have the same color:

\[
\bigwedge_{k=1}^{6} \bigwedge_{\ell=1}^{7} \bigvee_{t t' \text{ legal}} (x_{k\ell t} \land x_{(k+1)\ell t'})
\]
There are 12 pairs of tiles with matching horizontal colors with a “non-empty” color
\((q_0,R, q_1,R, q_f,L)\), and 272 pairs with the “empty” color (because there are 16 tile types
with the empty color at the left side and 17 tile types with the empty color at the right
side). So 284 pairs in total.

Vertically touching sides must have the same color:

\[
\bigwedge_{k=1}^{7} \bigwedge_{\ell=1}^{6} \bigvee_{t' \text{ legal}} (x_{k\ell t} \land x_{k(\ell+1)t'})
\]

4. We argue that the question whether a number is not prime is in NP (which is equivalent
to showing that the question whether a number is prime is in co-NP).

Build a nondeterministic TM which, on a given input \(n\), has \((n - 2)^2\) different (nondeter-
nomistic) executions. Each execution selects two numbers \(k\) and \(\ell\) in \(\{2, \ldots, n - 1\}\); there are \((n - 2)^2\) different ways of doing this. Next it is checked whether \(k \cdot \ell = n\). If
this is the case, the TM goes to the final state (end else it halts in a non-final state).

Clearly, the input \(n\) is not a prime number if and only if there is an accepting execution.

Note that \(n\) can be represented with \(O(\log n)\) decimals, and \(k \cdot \ell\) can be computed in
\(O(\log^2 n)\) time. So computation time is polynomial with regard to the input size.

5. Suppose that languages \(L_1\) and \(L_2\) are in NPSpace, so they are accepted by polynomial-
space bounded (nondeterministic) TMs \(M_1\) and \(M_2\), respectively.

For both union and intersection, the argumentation from exercise 4 and 5 can be copied,
but now for polynomial-space instead of polynomial-time bounded TMs.

Moreover, for the case of complements, we can employ Savitch’s theorem: PSpace = NPSpace.
This implies that \(L_1\) is accepted by a polynomial-space bounded deterministic TM. So
the argumentation from exercise 4 can be copied to argue that \(\overline{L_1}\) is in PSpace = NPSpace.

6. Suppose that the NP-complete language \(L\) is in co-NP. Consider any language \(L'\) in NP.
We prove that \(L'\) is in co-NP.

Since \(L\) is NP-complete and \(L'\) is in NP, there exists a polynomial-time reduction \(f\)
with \(x \in L' \Leftrightarrow f(x) \in L\). And so \(x \in \overline{L'} \Leftrightarrow f(x) \in \overline{L}\).

Since \(L\) is in co-NP, \(\overline{L}\) is accepted by a polynomial-time bounded TM. This implies
that \(f(x) \in \overline{L}\) can be computed in polynomial time. So \(x \in \overline{L'}\) can be computed in
polynomial time. In other words, there exists a polynomial-time bounded TM that
accepts \(\overline{L'}\). Hence \(L' \in \text{co-NP}\).

Vice versa, consider any language \(L''\) in co-NP. We prove that \(L''\) is in NP.

Since \(L''\) is in co-NP, by definition, \(\overline{L''}\) is in NP. We have shown above that this implies
that \(\overline{L''}\) is in co-NP. So, by definition, \(\overline{L''} = L''\) is in NP.
7. Input of the binary bounded tiling problem is a single tile, of size $O(1)$, together with the binary representation of the number $n$, of size $O(\log n)$. So the overall size of the input is $O(\log n)$.

Given a candidate solution of $n^2 - 1$ randomly placed tiles, it can be checked in time at most $O(n^2)$ whether this solution is correct. And \[ n^2 = 2^{\log_2 n} \] is exponential compared to the size $O(\log n)$ of the input. So the binary bounded tiling problem is in NEXP.

The proof that the binary bounded tiling problem is NEXP-complete, is very similar to the proof that the original bounded tiling problem (with as input the entire first row of tiles) is NP-complete. The main difference is that the input now has a logarithmic (instead of a linear) size in terms of $n$, so that the $n \times n$-tiling has an exponential (instead of a quadratic) size in terms of the size of the input. And a special series of tile types is needed, with special side colors, that can only be used in constructing the first row of tiles.