1. (a) False, hierarchy theorem.  (b) True, Savitch’s theorem.
(c) True, PSPACE is closed under complement.
(d) Open, stated in lecture.  (e) True, follows from definition.
(f) Open, implies NP = coNP (considering language 0SAT ∪ 1SAT);
negation implies PSPACE ≠ NP.
(g) Open, equivalent to NP = coNP.  (h) Open, stated in lecture.
(i) False, implies PSPACE = EXPSPACE.  (j) True, recompute bits of first reduction.
(k) Open, equivalent to NP = coNP.  (l) True, SAT is decidable.
(m) False, implies PSPACE = NL.  (n) Open, equivalent to P = NP.
(o) True, PATH ∈ P.  (p) True, NL = coNL.

2. First, show that C is in EXPTIME. Here’s the algorithm:

“On input ⟨M, w, i, j, α⟩:
1. Run M on w for j steps. If it halts in fewer steps, reject.
2. Accept if the ith symbol of the configuration of the jth step is α. Otherwise, reject.”

To analyze the running time of this algorithm, observe that to simulate one step of M we only need to update M’s configuration and the counter which records how long M has been running. Both can be done within O(j) steps (actually much less is possible, but unnecessary here). We run M for at most j steps, so the total running time of this algorithm is O(j^2), and that is exponential in the size of the input, because j represented in binary, so |j| = log_2 j and thus j^2 = (2^|j|)^2 = 2^{2 |j|} ≤ 2^{2n}, where n is the length of the entire input.

Second, we show that C is EXPTIME-hard, that is, that every language in EXPTIME is polynomial time reducible to C. Let A ∈ EXPTIME where M decides A in time 2^{n^k}. Modify M so that when it accepts it first moves its head to the left-hand end of the tape and then enters the accept state q_{accept}. Then the reduction of A to C is the polynomial time computable function f, where f(w) = ⟨M, w, 1, j, q_{accept}⟩ and j = 2^{n^k}.

3. First, SOLITAIRE ∈ NP because we can check in polynomial time that a solution works.

Second, show that 3SAT ≤_P SOLITAIRE.
Given φ with k variables x_1, ..., x_k and l clauses c_1, ..., c_l, first remove any clauses that contain both x_i and ¬x_i. These clauses are useless anyway and would mess up the coming construction. Construct the following l × k game G.

If x_i is in clause c_j put a blue stone in row c_j, column x_i.
If ¬x_i is in clause c_j put a red stone in row c_j, column x_i.
(We can make it a square m × m by repeating a row or adding a blank column as necessary without affecting solvability).
Claim: φ is satisfiable iff G has a solution.
(→): Take a satisfying assignment. If x_i is true (false), remove the red (blue) stones from the corresponding column. So, stones corresponding to true literals remain. Since every clause has a true literal, every row has a stone.
(←): Take a game solution. If the red (blue) stones were removed from a column, set the corresponding variable true (false). Every row has a stone remaining, so every clause has a true literal. Therefore φ is satisfied.
4. Show that $A_{TM} \leq_m INP$.
   Assume (to get a contradiction) that TM $R$ decides $INP$. Construct the following TM $S$ deciding $A_{TM}$.

   “On input $\langle M, w \rangle$:
   1. Construct the following TM $M_1$:
      “On input $x$:
      1. If $x \in EQ_{REX}$, accept.
      2. Run $M$ on $w$.
      3. If $M$ accepts $w$, accept.”
   4. Run $R$ on $M_1$.
   5. If $R$ accepts, accept; otherwise, reject.”

   Observe that if $M$ accepts $w$, then $L(M_1) = \Sigma^*$, and if $M$ doesn’t accept $w$, then $L(M_1) = EQ_{REX}$. So, $L(M_1) \in P$ exactly when $M$ accepts $w$.

5. (a) Obviously $ODD-PARITY \in L$ and we know $L \subseteq NL$. We proved that $PATH$ is NP-complete and so every language in NL is log-space reducible to $PATH$.
   Note: Giving a direct reduction from $ODD-PARITY$ to $PATH$ is possible too.

   (b) If $PATH \leq_L ODD-PARITY$ then $PATH \in L$ and thus $NL = L$, solving a big open problem.

6. We can assume without loss of generality that our BPP machine makes exactly $n^r$ coin tosses on each branch. Thus the problem of determining the probability of accepting a string reduces to counting how many branches are accepting and comparing this number with $\frac{1}{3}2^{n^r}$.

   So given $w$, we generate all binary strings $x$ of length $n^r$ (we can do this in PSPACE) and simulate $M$ on $w$ using $x$ as the source of randomness. If $M$ accepts, then we increment a count. At the end, we see how many branches have accepted. If that number is more than $\frac{1}{3}2^{n^r}$ we accept else we reject. This works because of the definition of what it means for a BPP machine to accept. If $w \in L$ then more than $\frac{2}{3}$ of $M$’s branches must accept. If $w \notin L$ then at most $\frac{1}{3}$ of its branches can accept.

7. (a) No, the prover for $\#SAT$ is not a weak Prover, as far as we know. Calculating the coefficients of the polynomials seems to require more than polynomial time.

   (b) The class weak-IP = BPP. Clearly, BPP $\subseteq$ weak-IP because the Verifier can simply ignore the Prover. Conversely, weak-IP $\subseteq$ BPP because we can make a BPP machine which simulates both the Verifier and the weak Prover $P$. If $w \in A$ then $P$ causes the Verifier to accept with high probability and so will the BPP machine. If $w \notin A$ then $P$ causes the Verifier to accept with low probability and so will the BPP machine.