5.8 b) If MAX has a choice between a ? node or a 1 node, it will choose the 1 node, and vice versa for MIN with -1 nodes. This is because the payoff is bounded by the constraints of the game to $1 \geq \text{payoff} \geq -1$. Players need not consider nodes of unknown value when know they can travel to a sibling that has the best value possible.

c) The minimax algorithm would fail in this game because of the cycles that are created by repeated states. This would create loops of recursive calls in the algorithm that never reach a base case and therefore would continue indefinitely.

MAX-VALUE and MIN-VALUE could pass all ancestor states to their children with each recursive call. Then a base case could be added to both: if the current state is not a terminal state but it is in the ancestor list, return 0. In this particular example this change is effective, because the only cases in which a node has to choose between an unknown path and a known one is when the known path leads to a victory. In other games the known path(s) may lead to a defeat or a negative outcome, which would case the player to choose the unknown path, possibly creating a cycle.

d) The optimum strategy for both players is to travel toward their destination at every move. If a player does and the other player does not, the first player will reach a state where it has less moves to its destination than the second. When both players move toward each other, they will meet in the middle at some point. If n = 3, then state 2,3 will be reached with B to move and B will win by moving to 1,3. If n = 4, then state 2,3 will be reached with A to move and A will win by moving to state 1,3. All games with odd values of n will be winning for B because the game can be visualized as an n = 3 game with an equal number of squares added to either side of the point where the two players meet, and B because B was winning where n = 3, it will reach its destination faster than A because it takes each player the same amount of time to traverse an equal number of squares. The same argument can be made to justify a win for A when n is even.

5.10 a) In each node in the tree, each square could be either marked with an X, marked with an O, or not yet marked. So there could be at most 3^N nodes.

b) The tree has at least as many nodes as it has leaves. Each leaf would be a cat's game with all squares marked as either X or O. So there must be at least 2^N nodes.

c) The evaluation function used in 5.9 is plausible because X_2 and O_2 are quite a bit more important than X_1 and O_1. In particular, it seems plausible that X_{n+1} should be weighted with 3 times as much importance as X_n, and similarly for O. For generalized tic-tac-toe, redefine $X_n(s)$ as the number of winning positions containing exactly n X's and no O's within state s, and vice versa for O_n. The evaluation function then becomes:

$$Eval(s) = \sum_{i=1}^{n} (3^{i-1}X_i(s) - 3^{i-1}O_i(s))$$

d)

$$1s = 3^N \frac{100Ninstr}{2 \times 10^9 instr/s} \Longrightarrow N \approx 13$$

$$60s = 3^N \frac{100Ninstr}{2 \times 10^9 instr/s} \Longrightarrow N \approx 16$$

$$3600s = 3^N \frac{100Ninstr}{2 \times 10^9 instr/s} \Longrightarrow N \approx 20$$

6.2 a) Let $X = \{N_1, N_2, \ldots, N_k\}$, the set of all knights (N is the abbreviation for knight in chess notation).

b) Each knight can be assigned one of the squares on the chessboard. Formally, D is the cartesian product of the set of natural numbers $\leq$ the width of the chessboard, for each knight. I.e.

$$D_i = \{[1,1],[1,2],\ldots,[1,n],[2,1],[2,2],\ldots,[2,n],\ldots,[n,n-1],[n,n]\}$$

c) Each knight is constrained to only take squares that cannot be attacked by any other knight already assigned a square. More formally, this is a large set of binary constraints such as

$$\langle (N_1, N_2), (N_{1,row} \neq N_{2,row} + 1) or (N_{1,column} \neq N_{2,column} + 2) \rangle$$

d) This could be formulated as a local search problem by making all knights unassigned to squares in the initial state. An action would be to assign a knight to a square, which would lead to a new state. The objective function would simply count the number of assigned knights. The more knights assigned, the better. States that are not arc-consistent would be failures and not expanded further.