Problem 1

d is the correct answer.
(i) is not learning because the developers pin it down mathematically (they derive a statistical model). (ii) is supervised learning because we get an input and an output (the output is the label). (iii) is reinforcement learning because the computer just chooses outputs (moves) and sees what happens eventually, and then grades the outputs (moves) that lead to losing.

Problem 2

a is the correct answer.
(i) doesn't need learning to be done (can be done with a simple algorithm) and (iii) can be pinned downed mathematically with physics. On the other hand, (ii) is well suited for the learning approach because a pattern exists, we cannot pin it down mathematically, and there would probably be data on it. More specifically, we could look at how much is spent, where the card is being charged, how often the card is being charged, etc, and look for patterns in those attributes. (iv) is also well suited, again because a pattern exists, we cannot pin it down mathematically, and we have data on it. More specifically, the lights could keep learning until their functioning/timing approximately matches the desired functioning/timing.

Problem 3

d is the correct answer.
When picking two balls from the same bag, there are four possibilities:

1. Black Black
2. Black Black
3. Black White
4. White Black

Because we picked a black ball, we are in the first three options. So the probability of the next ball being black is 2/3.

Problem 4

b is the correct answer.
The probability that any marble we draw is not red is $\mu = .45$. The probability that we draw 10 not red marbles in a row is $.45^{10} = 3.405 \times 10^{-4}$, which is the same as the probability that $\nu = 0$.

Problem 5

c is the correct answer.
We have that the probability of one sample having $\nu = 0$ is 3.405×10^{-4}. So the probability that one sample does not have $\nu = 0$ is $1 - 3.405 \times 10^{-4}$. Then we have that the probability that all 1,000 of the samples do not have $\nu = 0$ is $(1 - 3.405 \times 10^{-4})^{1000}$. So the probability that at least one of the samples has $\nu = 0$ is $1 - (1 - 3.405 \times 10^{-4})^{1000} = .289$.

Problem 6

e is the correct answer.
The 8 possible target functions are as follows

1. $f(101) = 0$, $f(110) = 0$, $f(111) = 0$
2. $f(101) = 1$, $f(110) = 0$, $f(111) = 0$
3. $f(101) = 1$, $f(110) = 1$, $f(111) = 0$

4. $f(101) = 1$, $f(110) = 0$, $f(111) = 1$
5. $f(101) = 1$, $f(110) = 1$, $f(111) = 1$
6. $f(101) = 0$, $f(110) = 1$, $f(111) = 0$
7. $f(101) = 0$, $f(110) = 1$, $f(111) = 1$
8. $f(101) = 0$, $f(110) = 0$, $f(111) = 1$

Let us now consider each hypothesis.

(a) Score $= 1 * 3 + 3 * 2 + 3 * 1 + 1 * 0 = 3 + 6 + 3 = 12$

(b) Score $= 1 * 3 + 3 * 2 + 3 * 1 + 1 * 0 = 3 + 6 + 3 = 12$

(c) 1. has 2 agreeing points, 2. has 1 agreeing point, 3. has 0 agreeing points, 4. has 2 agreeing points, 5. has 1 agreeing point, 6. has 1 agreeing point, 7. has 2 agreeing points, 8. has 3 agreeing points. Score $= 1 * 3 + 3 * 2 + 3 * 1 + 1 * 0 = 3 + 6 + 3 = 12$

(d) 1. has 1 agreeing points, 2. has 2 agreeing points, 3. has 3 agreeing points, 4. has 1 agreeing point, 5. has 2 agreeing points, 6. has 2 agreeing points, 7. has 1 agreeing point, 8. has 0 agreeing points. Score $= 1 * 3 + 3 * 2 + 3 * 1 + 1 * 0 = 3 + 6 + 3 = 12$

Problem 7

b is the correct answer.
I wrote a program to repeat the experiment for 1000 runs, with 10 points, and the value closest to my results was 15. The program did exactly what the problem told us to do (used the PLA). See code for more details.

Problem 8

c is the correct answer.
After finding g, I compared it to f for 10,000 random points, and found out what ratio of points the functions disagreed on. This was done for each of the 1000 runs, and I averaged the ratios to approximate the probability. This was using 10 points. See code for more details.

Problem 9

b is the correct answer.
I wrote a program to repeat the experiment for 1000 runs, with 100 points, and the value closest to my results was 100. The program did exactly what the problem told us to do (used the PLA). See code for more details.

Problem 10

b is the correct answer.
After finding g, I compared it to f for 10,000 random points, and found out what ratio of points the functions disagreed on. This was done for each of the 1000 runs, and I averaged the ratios to approximate the probability. This was using 100 points. See code for more details.

Code

```
import java.awt.*;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;

/**
 * Created by mattlim on 10/4/14.
 */
public class Assignment1 {
    static Point.Double firstPoint;
    static Point.Double secondPoint;
    static Double slope;
    static double[] weightVector;
    static ArrayList<MyPoint> myPoints = new ArrayList<MyPoint>();
    static ArrayList<MyPoint> largePointList = new ArrayList<MyPoint>();
    static int globalCounter = 0;
    static int globalCounterTotal = 0;
    static double globalProbabilityCounterTotal = 0;
    static Random random = new Random();
    static final int NUM_POINTS = 100;

    public static void main(String[] args) {
        System.out.println("Hello");
        Assignment1.initializeVariables();

        while (isMisclassified()) {
            runPLAIteration();
        }


        for (int i = 0; i < 1000; i++) {
            globalCounter = 0;
            System.out.println("Outside iteration #" + i);
            while (isMisclassified()) {
                runPLAIteration();
            }
            globalCounterTotal += globalCounter;
            double counter = 0;
            for (MyPoint point : largePointList) {
                if (signOfInnerProduct(point) != point.score) {
                    counter++;
                }
            }
            globalProbabilityCounterTotal += counter / (double) largePointList.size();
            Assignment1.initializeVariables();
        }
        System.out.println("Average iterations = " + globalCounterTotal / 1000);
        System.out.println("Average probability = " + globalProbabilityCounterTotal / (double)
            1000);
    }

    public static void setScoreForPoint(MyPoint point) {
        double lineX = (point.y - firstPoint.y) / slope + firstPoint.x;
        if (lineX > point.x) {
            point.score = -1;
        } else {
            point.score = 1;
        }
    }
```

```
    public static void runPLAIteration() {
        Collections.shuffle(myPoints);
        for (MyPoint point : myPoints) {
            if (signOfInnerProduct(point) != point.score) {
                adjustWeightVector(point);
                break;
            }
        }
    }

    public static int signOfInnerProduct(MyPoint point) {
        double innerProduct = weightVector[0] * point.x + weightVector[1] * point.y +
             weightVector[2];
        if (innerProduct > 0)
            return 1;
        else
            return -1;
    }

    public static void adjustWeightVector(MyPoint point) {
        globalCounter++;
        weightVector[0] += point.score * point.x;
        weightVector[1] += point.score * point.y;
        weightVector[2] += point.score;
    }

    public static boolean isMisclassified() {
        for (MyPoint point : myPoints) {
            if (signOfInnerProduct(point) != point.score) {
                return true;
            }
        }
        return false;
    }

    public static void initializeVariables() {
        firstPoint = new Point.Double(2 * random.nextDouble() - 1, 2 * random.nextDouble() - 1);
        secondPoint = new Point.Double(2 * random.nextDouble() - 1, 2 * random.nextDouble() -
             1);
        slope = (secondPoint.y - firstPoint.y) / (secondPoint.x - firstPoint.x);
        weightVector = new double[] {0, 0, 0};
        myPoints.clear();
        for (int i = 0; i < NUM_POINTS; i++) {
            MyPoint randomPoint = new MyPoint(2 * random.nextDouble() - 1, 2 *
                 random.nextDouble() - 1);
            Assignment1.setScoreForPoint(randomPoint);
            myPoints.add(randomPoint);
        }
        largePointList.clear();
        for (int i = 0; i < 500000; i++) {
            MyPoint randomPoint = new MyPoint(2 * random.nextDouble() - 1, 2 *
                 random.nextDouble() - 1);
            Assignment1.setScoreForPoint(randomPoint);
            largePointList.add(randomPoint);
        }
    }
}
```