2a: The program was created using Greenfoot in the Java programming language. It sets up 9 "cards" that can be flipped to reveal either a check or an x behind them. There is one check and 8 "x"s, assigned randomly every run of the program. By pressing the number keys on the keyboard, the cards can be flipped. Once a card with an "x" is flipped, the game is over and the user can reset the game by pressing "r."

2b: I completed the program independently. I started out by setting up the background and arranging the cards before developing the Card class. After looking at the pixel size of the cards, I determined the coordinates I needed to use. I tested the program multiple times to make sure I got the coordinates and placement of the cards right with each version of the positioning algorithm. I was able to visually confirm the accuracy of the program by running it to check if it worked. Then, I worked on getting them to flip. Lastly, I developed the algorithm that tests if the card flipped had an x behind it and made the Gameover class to end the game. At first, I wanted to make the program work so that whenever I clicked the card, the card would flip. However, I had trouble implementing Greenfoot's mouseClicked() method. In the act() method of the Card class, I called the turnCard() method I wrote, which would flip the card. Although I used the mouseClicked() method properly, it did not flip the card; the image of the card did not change as intended. To solve this issue, I replaced the mouseClicked() call with a call to Greenfoot's isKeyDown() method in the same location to flip the card based on pressing its order value (1-9). Each card is instantiated with its order value, so after assigning that parameter to a private instance variable, I could use that variable as a parameter in the isKeyDown() call. I also needed an easier way to restart the game, so I developed a way to press r in order to reset the background when the Gameover screen pops up. With an if statement checking if the user presses "r" in the act() method of the Gameover class, pressing "r" would reset the cards.

2c.

```
private void turnCard()
{

    if (pressVal == rVal)
    {
        setX();
    }
    else
    {
        setCheck();
    }
}
```

This algorithm is the central decision making part of my program. Located in the Card class and called through each card's act method, the turnCard method will set the image to either a check or x. Each card is instantiated with their order value (1-9), which is shown on the picture of the card. In the Background class, I set a variable to a random integer value, which every card is instantiated with as a parameter. In the constructor, the order value and the random value are assigned to the class's private instance variables. The order value of the card is then compared with the random value that decides the "x" card as a parameter. This method tests if the card's value is the same as the "x" value, then decides if it should call the setX() method or the

setCheck() method.

```
public void setX()
{
    cardVal = false; //false = x
    setImage(new GreenfootImage("x.png"));
    GameOver g = new GameOver();
    getWorld().addObject(g, 315,345);
}
```

If the card's value is the same as the "x" value, it calls the setX method, which changes the image of the card to an "x" and creates a new "GameOver" object and adds it to the middle of the background.

```
public void setCheck()
{
    cardVal = true; //true = check
    setImage(new GreenfootImage("check.png"));
}
```

If the card's value doesn't match the "x" value, the algorithm calls the setCheck method to just set to the "check" image (the player can still flip more cards.) If wished, these two smaller methods could be called independently of the algorithm by a Card object.

2d:

```
int numCard = 1;
for(Card c: cardList)
 {
     addObject(c,x,y);
     numCard++;
     if(numCard ==4 || numCard ==7)
     {
         x=105;
         y+= 230;
     }
     else
     {
         x+=210;
     }
 }

}
}
```

I created a list consisting of 9 cards to apply abstraction. The iterative process of going through each item in the list increases code efficiency because I no longer had to add every card to the background separately (I had around 9 separate addObject statements and 9 separate calls to the card's act() method.) I made a counter variable which I use to decide what coordinates to update. For example, if the program has already added 3 cards, it must know that it needs to start the second row. I also made variables for the x and y coordinate. If the counter variable equals 4 or 7, I would need to add 230 to the y coordinate variable and reset the x coordinate variable to 105,

which sets the image to the far left. Using those updated variables, the program then adds the card object.