First, I started out with the basic things at the start, including the error handling for invalid test cases because of incorrect area inputs and other things like that. I then tried creating a dictionary of distinct letters in the alphabet that mapped to a tuple of a rectangle, but then I scrapped that idea because it was clunky and not an easily implementable solution to the problem that I had which was actually modeling the board. I later switched to a 2D matrix which seemed to be probably the best solution in the scenario, and whenever I added a new rectangle, I decided I would just look for a distinct letter to represent the rectangle that wasn't already in the matrix.

So after figuring out how I wanted to actually model it, I decided to create a method that added rectangles to the board. I thought of just handling error checking in the method itself instead of doing it elsewhere so that I could just call the method whenever I wanted without checking if it fit the constraints beforehand. In my method, I took in the arguments of the rectangle dimensions and its top-left position and checked that it was possible to add the rectangle in the intended area before even adding it just to make sure I wouldn't be hit with any confusing errors that I couldn't understand; I checked whether it wasn't already taking up a space with another rectangle and making sure that it would still fit on the board. This was practically going to be my "get sorted values" method since it would tell me what rectangles I would be able to put in the next open position. This method is pictured below which my entire code is pretty much built around.

```
50
51  # Pass in a tuple (length, width), 2D array, top-left x-coordinate, and top-left y-coordinate
52  def add_rect(rectangle, board, x, y):
53      alphabet = [letter for letter in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ']
54      for row in board:
55          for elem in row:
56              if elem in alphabet:
57                  alphabet.remove(elem)
58      height = rectangle[0]
59      width = rectangle[1]
60      designated_letter = alphabet[0]
61      for y_coord in range(y, y + height):
62          for x_coord in range(x, x + width):
63              # First check if the index will be out of bounds
64              try:
65                  val = board[y_coord][x_coord]
66              except:
67                  return False
68              # Check if space is empty space
69              if board[y_coord][x_coord] == "":
70                  board[y_coord][x_coord] = designated_letter
71              else:
72                  # Rectangle cannot be placed there so return False
73                  return False
74      return board
```

I knew this was going to be a backtracking lab since I would have to keep going back and forth checking which rectangles would fit in which positions so I had to choose how I wanted to do my "get next unassigned var" (which returned a tuple of x and y coordinates) and my "get sorted values" method; I used the add rectangle method as a way to do my get sorted values because it told me which ones fit and which ones didn't and I decided just to use the next empty spot for my unassigned var. The difficult part of this is to make it so I wouldn't use duplicates; I had to know if a rectangle was already on the board that I could not use it again. The first time I tried without checking duplicates, I was given this board output which just used the same rectangle 3 times in a row without even acknowledging the other

rectangle.

```
tanmai@tanmais-macbook Calibron % python3 Blue2_Calibron12.py "2 3 1x2 2x2"
A A B
C C B
```

It ended up using the 1x2 rectangle 3 times instead of ever touching the 2x2 rectangle and putting it in the left 2x2 square part. I decided to go back to the dictionary strategy of mapping certain rectangles to different letters in the alphabet. Unfortunately, no matter how many times I tried this, it still didn't work; it either ended up putting the same letter in every spot or putting the same rectangle in multiple places. Sometimes it would end up placing rectangles diagonally or in the same spot which I tried debugging on the smaller size test cases which I still couldn't figure out. I am declaring failure because after working on it for a few hours, I could not figure out why it was duplicating the same rectangle or placing them in the board weirdly. I tested my add_rect method with a bunch of different cases and it was right and I couldn't seem to point out the spot where my code was bugging out. I know that for sure I didn't have to implement forward looking or constraint propagation because it was already running pretty quickly on the smaller test cases so I decided to leave that part out but it still ended up not working.