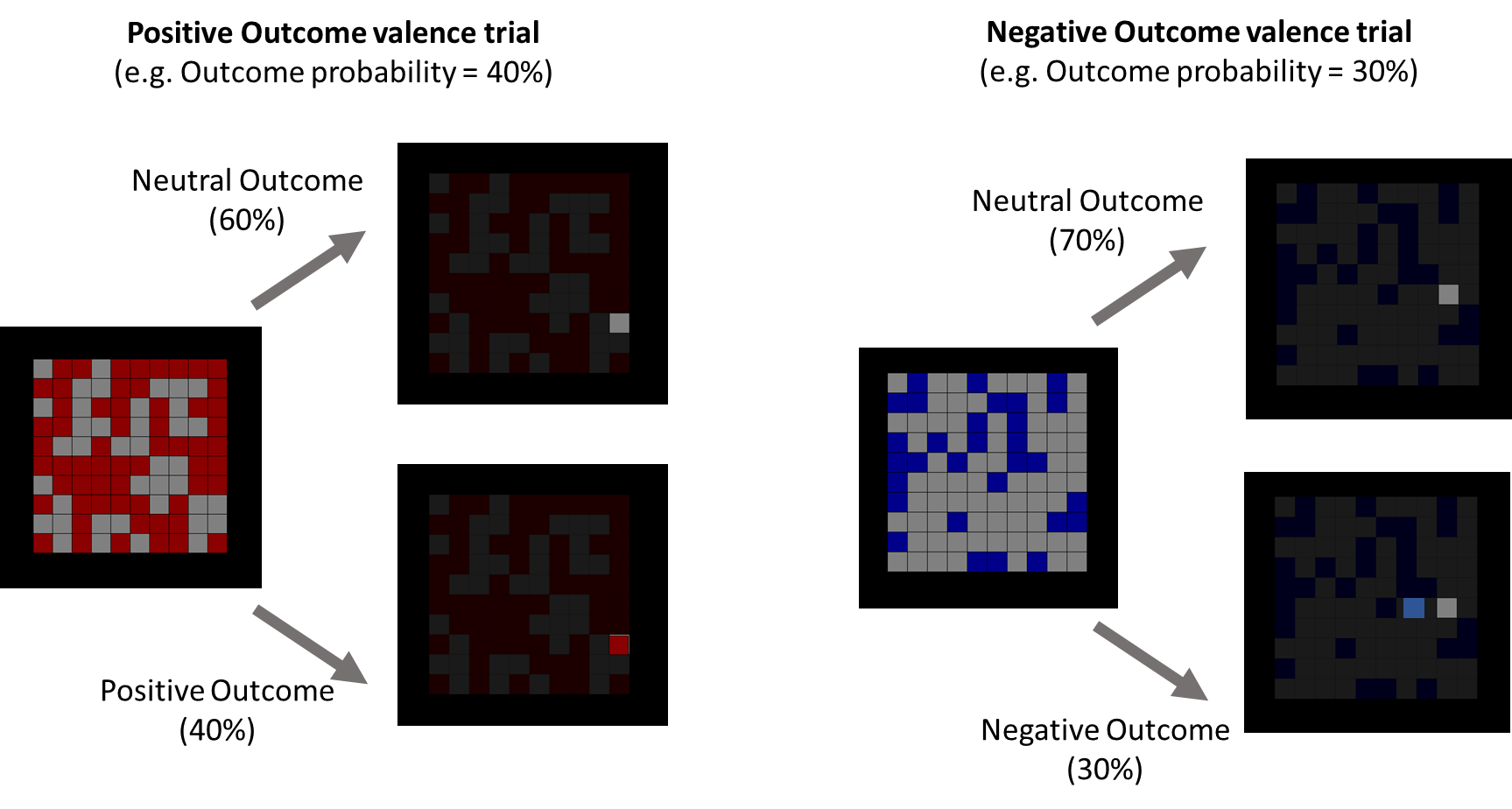
**User case for the task**

Participants will be given instructions about the computerized task they will be engaging in. In the instruction, participants will be told 1) that they will be predicting the color of a square the computer would randomly pick from a grid-like stimuli, 2) that they could win and lose points on each trial. Specifically, the participants will be told that whether they win or lose points depends on the value of the color of square the computer picked at random. The main goal of the task for the participants is to forecast as accurately as possible which color square will be chosen throughout the trials.

In a typical trial, a grid stimuli consisting of 20 x 20 squares will be presented to the participants, and one square will be randomly drawn by the computer. The color of the selected square will be associated with either winning or losing points. If the computer picked a square with that color, which we call the “critical” square, the participant would win 10 points or lose 10 points, depending on the trial. The other color will indicate zero points won or lost (i.e. “neutral” square). Participants will be asked to make predictions as to which of the two colors the computer would randomly pick from the grid. Once participants provide a response, a given square will remain while the rest of the grid disappears, providing feedback to the participant on which square was selected, whether they predicted correctly, and how much they won or lost in terms of points.

Overall, outcome valence (i.e. positive vs. negative outcome) and outcome probability will be manipulated in a within-subject fashion. For positive outcome conditions, the grid would consist of critical squares that result in 10 points won and the neutral squares (0 points). For negative outcome conditions, the grid would involve combination of critical squares that yields 10 points lost and neutral squares (0 points). Importantly, outcome probability will be manipulated by varying the proportion of critical squares to neutral squares (i.e. 30% - 70%).



**User Case for function ‘SetGridPattern’**

This function will generate an array that sets the color pattern for a grid stimuli. The array will contain *n* number of strings indicating color names (i.e. color strings) for each square in the grid, *n* being the number of squares (i.e. sqr\_N) preset from previous code. Once the array of color strings are created, the order of the strings will be randomized. This randomization will allow the later grid to have randomly colored square patterns, rather than a grid with two blocked colors. The output of this function will be used later in the DrawGrid function.

Inputs:

- crit\_col = a string. color of the critical square

- neut\_col = a string. color of the neutral square

- prob = a float. proportion of critical square out of all squares

Output:

- colorarray\_temp = an array that contains gridN x gridN number of color strings

**Code**

def SetGridPattern(crit\_col = 'darkred', neut\_col = 'gray', prob = .5):

######Create random array for coloring squares

crit\_N = int(sqr\_N\*prob) #number of critical squares

crit\_list = [crit\_col]\*crit\_N #create list that contains crit\_N number of color strings

neut\_N = int(sqr\_N\*(1-prob)) #number of neutral squares, catches errors for miscalculations in the probs above 50% (for some reason that happens)

if neut\_N%2 == 0:

neut\_list = [neut\_col]\*neut\_N

elif neut\_N%2 == 1:

neut\_list = [neut\_col]\*(neut\_N+1)

#######Scatter the colored square distribution if conditions are met

colorarray\_temp = crit\_list+ neut\_list

random.shuffle(colorarray\_temp)

return colorarray\_temp

**Example (with sqr\_N = 100)**

>>> x = SetGridPattern(crit\_col = 'darkred', neut\_col = 'gray', prob = .5)

>>> print(x)

['darkred', 'gray', 'darkred', 'darkred', 'gray','darkred', 'gray', 'gray', 'gray','darkred', 'darkred', 'darkred', 'darkred'… 'gray', 'darkred', 'gray'] \*this is a truncated version of the original array

>>>Print(length(x))

100

>>>y = SetGridPattern(crit\_col = 'darkblue', neut\_col = 'gray', prob = .7)

>>>print(y)

['gray', 'darkblue', 'gray', 'gray', 'gray', 'gray', 'darkblue', 'darkblue', 'gray', 'darkblue', 'darkblue', 'darkblue', 'darkblue', 'gray', 'darkblue'…..'gray', 'darkblue', 'darkblue'] \*this is a truncated version of the original array

>>>Print(length(y))

100