Lab 5 - Red Light, Green Light, Approaching the Doll

Last week, we built the Red Light, Green Light game, which is essentially a "Squid Game" themed Go/No-Go task.

To follow along today, please download the Lab 5 module from BruinLearn. We will be starting with the **lab5_demo_redlight_greenlight.psyexp** file, and modifying it as we work through the worksheet. Let's run this program and take a look at what we have so far...

Today, we will learn to derive some of the values that you will need to calculate signal detection theory metrics. We will also add some embellishments to make this game more interesting, both in terms of the research question that we may want to ask and in terms of making it more gamified and engaging for the participant.

Red Light, Green Light, Approaching the Doll

In this version of the Red Light, Green Light game, participants take a "step" closer to the finish line every time they make a Hit (i.e., pressing space when the back of the doll is facing them). If participants take a "step" closer when the doll turns around (i.e., False Alarming or pressing space when the doll is facing them), they need to take a few "steps" back. To give the sense of progress, the doll will appear larger with each "step". Once the participant reaches the target number of Hits, the session finishes.

To-Do List:

- Calculate and display the Hit Rate and False Alarm Rate
- Set a number of "steps" to get to the finish line
- With each "step", make the doll appear larger
- Introduce incentives/costs for each Hit and False Alarm (e.g., each Hit is 1 step closer, but each FA is 2 steps back)
- Enable experimenter to specify parameters at the start of the experiment
- Fun embellishments (if time)!

Calculating Hit Rate and False Alarm Rate

Picking up from last week, let's look at how to calculate the Hit Rate and False Alarm Rate. We then want to display these values at the end of the session.

Remember:

- Hit = pressing the spacebar when the doll is facing backwards
- False Alarm = pressing the spacebar when the doll is facing forwards

To get the hit rate, we need to count the number of hits and divide that by the number of "signal" trials (i.e., "go" trials). To get the false alarm rate, we need to count the number of false alarms and divide that by the number of "noise" trials (i.e., "no-go" trials).

Steps

1. Create a "feedback" routine. Add a Code component, a Keyboard component called `fb_kb`, and a Text component called `fb_text`.

2. In the Begin Routine panel of the Code component, we will calculate the commission error rate of the subject. First, access the stored responses of the `task_kb` object with this line of code: `task_kb_keys = trials.data['task_kb.keys']`. Then loop through each of the recorded responses and derive the `nogo_count` (number of no-go trials) and `fa_count` (number of false alarm trials) using a nested if statement. Then calculate `fa_rate` from those two variables. Then build a `feedback_str` to be displayed by the text component.
3. In the text component, change duration to 20s. Add `$feedback_str` to the Text field.
4. Set up the `fb_kb` Keyboard component. Duration to 20s, allowable keys to 'space', make sure "Force end routine" is checked.

In [1]:
```
### (Step 2) The commented template for calculating hit and false alarm rates:
# retrieve keypresses from task_kb.keys


# create new counter variables for hit/fa trials and go/nogo trials


# count the number of signal/noise trials; count the number of hit/fa


# calculate the rates


# create the string to display in fb_text
```

Make the doll image larger or smaller with each trial

The goal with this is to give a sense of movement when the doll appears larger (approaching) and when the doll appears smaller (retreating). Moreover, this gives the participant a sense of progress towards their target "distance".

Steps:

1. **In Begin Experiment tab**, create a `distance_goal` variable where we specify the number of hits or "steps" needed to pass the game. Also create a `distance_counter` variable that keeps track of the number of hits.
2. Create the `min_w`, `min_h`, `max_w`, `max_h` variables so that we can specify the starting and ending dimensions of the image.
3. Calculate `inc_w` and `inc_h` using the previous variables. These are the increments (in image dimensions) to be added each time a hit is recorded (i.e., this is like how "large" of a step to take). Calculated as: `inc_w = (max_w - min_w)/distance_goal`.
4. **In Begin Routine tab**, Update the `im_w` and `im_h` variables, which are the *current* dimensions of the doll image. These variables are calculated as: `im_w = min_w + distance_counter*inc_w`.
5. **In End Routine tab**, Here, we need to update the distance counter by 1 if a hit occurred.
6. We also want to check if `distance_counter == distance_goal`. When this conditional is True, terminate the trial early using this: `trials.finished = True`.
7. **In doll_im Properties**, In the Layout tab, find the "Size" field and enter `[im_w, im_h]`. Make sure to change the field to set every repeat!

Now test the code and see if the doll gets bigger!

Reward and Penalty

Our research objective is to shift our participant's response criterion by manipulating the reward/penalty outcomes of Hits and False Alarms. Reward is operationalized by pairing Hits with a number of steps forward, and penalty is operationalized by pairing False Alarms with a number of steps backward.

We want to include several different reward/penalty conditions. For the first condition, a Hit moves the subject 1 step forward and a False Alarm moves the subject 1 step back. This is essentially our "control" or "baseline" condition.

Steps:

1. **In End Routine tab**, when a False Alarm occurs (i.e., `sequence[trials.thisRepN] == 1 and task_kb.keys == 'space'`), subtract one step from the distance counter.
2. **Edge Case!** What if `distance_counter` is 0 and a False Alarm occurs? Should it be set to -1? We should handle this case by checking that `distance_counter` is greater than 0 before subtracting 1 from it.

User Defines the Reward and Penalty

In addition to the first condition, we may want to allow the experimenter to specify rules at the beginning of the session. To enable this, we will make use of the **Experiment info** attributes that can be set in the Experiment Settings window.

Steps:

1. **In Experiment Settings**, add 5 additional fields (and their defaults) to 'Experiment info': `hit_outcome` (1), `fa_outcome` (0), `hits_goal` (20), `num_go_trials` (200), `num_nogo_trials` (100)
2. **In Begin Experiment**, replace the assigned values of `num_0s`, `num_1s`, and `distance_goal` with `int(expInfo['num_go_trials'])`, `int(expInfo['num_nogo_trials'])`, and `int(expInfo['hits_goal'])`, respectively.
3. **In End Routine**, replace the value to update distance counter with `int(expInfo['hit_outcome'])` and `int(expInfo['fa_outcome'])`.
4. **Edge Case!** What happens if the `distance_counter` is less than the `fa_outcome`? `distance_counter` would be negative. We need to handle these cases.

Question: Let's say that our study consists of 2 group conditions. For one group, a False Alarm moves you 5 steps back. For a second group, a False Alarm moves you 10 steps back. Which group would you expect to False Alarm more (i.e., which group would have their decision criterion more biased towards the noise distribution)?

Now let's test our experiment! At the prompt, set `hit_outcome` to 1 and `fa_outcome` to 10. This means that for every hit, the subject moves forward one step. For everay false alarm, the subject moves back 10 steps.

As you can see, this flexibility in allowing the user to specify parameters is one of the advantages of taking a more programmatic approach earlier (instead of hard-coding everything).

Special Reward/Penalty Condition

Let's create one special reward/penalty condition where subjects are *returned to the beginning* if they make a False Alarm. The experimenter should be able to specify this conditions by typing "beginning" in the `fa_outcome` experiment info field.

Steps:

1. **In End Routine**, in the code block of the if statement that checks for False Alarms, add an additional nested code block that tests for whether the `expInfo['fa_outcome'] == "beginning"` . If it does, set `distance_counter` to 0; else, do what was done earlier.
2. Since the last step was difficult to articulate in words, I'll just explain the full code:

In [ ]:
```
# update distance counter if trial
if sequence[trials.thisRepN] == 0 and task_kb.keys == 'space':      # test for HIT
    distance_counter = distance_counter + int(expInfo['hit_outcome'])
elif sequence[trials.thisRepN] == 1 and task_kb.keys == 'space':    # test for FA
    if expInfo['fa_outcome'] == "beginning":
        distance_counter = 0
    else:
        if distance_counter >= int(expInfo['fa_outcome']):
            distance_counter = distance_counter - int(expInfo['fa_outcome'])
        else:
            distance_counter = 0
```

Embellishments!

Now let's add some features to make this experiment feel more gamified*. Here are a couple of ideas:

- Add background music that continuously loops. Maybe a song from the Squid Game soundtrack?
- "Game Over" screen whenever you false alarm. For example, maybe have the doll's eye turn red and screen turn red?
- "Finished" screen when you complete the experiment. One idea is to have some ominous-sounding text like "Congratulations on passing the first game. Next game..." and then show the Squid Game dalgona symbols.
- Any other easy to execute ideas?

I've curated the stimuli to make these work, but I haven't coded them yet so I don't have instructions.

*Note that in a cognitive psychology lab experiment, there's probably no need to "gamify" the study. This is just for fun!

Next Week...

We will talk about data and recording the data to a custom csv file!