# Assignment 3 Supporting Document: VJVL7

## (1) Experiment Overview

This experiment is a replication of Bechlivanidis and Lagnado (2013). As those authors review, several experiments have previously shown influences of causal beliefs on perception. The current experiment is concerned with the effect of causal beliefs on the perceived temporal order of events.

## (2) Description of Procedure

Subjects are randomly assigned to two conditions, experimental and control. Subjects in the experimental condition play four 'training' rounds in an interactive physics world, featuring different shapes with different causal properties. Subjects can move certain shapes into different positions, and upon pressing play, the objects are moved by their own acceleration, or by the forces exerted by other objects (blue 'balls'). The aim is to get a yellow triangle ('target') into a purple box ('goal'). On later trials, the target begins as a red square, and must be turned into a yellow triangle by colliding a green square ('key') with a black platform ('lock'), before it will be accepted by the goal. A reset button allows subjects to reset a trial if they fail to get the target into the goal. In the fifth 'test' round, subjects cannot move any of the shapes, and are asked to press play and observe the events that occur. In violation of the causal rules established in the training rounds, the target square enters the goal and becomes a triangle shortly *before* the key collides with the lock. Subjects in the control condition are not given interactive training rounds, and instead only view the final clip.
All subjects are then asked to report the perceived order of the different events, explain their answers, and attribute the red square's transformation into a triangle to a cause.

## (3) Experimenter's Manual

To run the experiment with default settings, simply open and run 'main.py'. This will launch a window where subjects will be asked to enter demographics before playing the interactive game described above and, finally, reporting what they saw and being debriefed.

The first time the experiment is run, an output csv file will be created in the same folder as 'main.py' (by default, this is called 'causationData.csv'). Subsequent sessions will append a new line of data to this file. The top row of the file is a header listing the different fields of data being collected. Each subsequent row represents one participant. Fields include the following:

- a participant ID
- condition (experimental or control)
- gender and age
- variables representing the reported order (1 = first, 4 = last) of the key colliding with the platform ('unlockTime'), the target turning into a triangle ('triangleTime'), the target entering the goal ('goalTime') and the congratulations message appearing ('congratulationsTime')
- boolean variables representing whether or not subjects explained their answer according to what they remember ('expRemember' – subjects in the control condition did not have this option, and so it is set to FALSE by default), what they saw ('expSaw') or what made sense ('expSense'). The variable 'expOther' contains any explanations typed into an 'other' box, or 'NA' if this is left empty.
- A variable representing subjects' attribution of the square's transformation ('green' = attributed to the green key hitting the lock, 'red' = attributed to the red target hitting the goal, anything else = text typed into an 'other' box).

There are several parameters of the experiment that can be changed:

- Settings relating to demographics and data collection (listed in CAPITALS at the top of 'main.py'):
    - FILENAME sets the name of the output file
    - CONDITIONS sets the number of conditions and their names
    - FIELDS contains the different fields of data being collected, and their names
- Settings relating to the interactive physics world (listed in CAPITALS at the top of 'causationProgram.py'):
    - Parameters setting the size of shapes (e.g. BALL_SIZE) take integer values
    - Parameters setting the colour of shapes, backgrounds and text (e.g. TARGET_COLOUR) take a tuple of RGB values
    - A WIN_MESSAGE parameter contains the text to be displayed open completing a round: takes a string
    - DEFAULT_CURSOR and DRAG_CURSOR parameters set the cursor to be used under normal circumstances and when hovering over a draggable object: take pygame.cursors objects
    - REPEL_MAGNITUDE sets the strength of the ball's repulsion force: takes an integer
    - REPEL_RANGE and GOAL_RANGE set the range over which repulsion (balls) and attraction (goals) occurs: takes an integer
    - TICK_DURATION sets the duration of each frame of the animation: takes an integer
    - SHAPES is a dictionary containing the rounds to be used in the experiment, and for each round a list of entities of each type to be initialised that round. The entities are stored as tuples of parameters; see the class definitions later in the file for descriptions of these. You can add or remove rounds, or add, remove or alter the objects present in each round.
    - INSTRUCTIONS is a dictionary containing instruction messages to precede each round. If a round's message is left empty, no message will be displayed.

# (4) Program Highlights

## Customisability

- There are many parameters (detailed above) that can be changed, allowing the experimenter to tailor the program to fit their needs with relative ease (e.g. one could easily create a version with 20 rounds featuring large, yellow, immobile balls that slowly pull other items towards them!).

## Interactivity

- Not only is this program animated, but objects will exert forces on one another according to an internal physics, and subjects can move objects around and play, pause or reset each round via both the keyboard and clickable buttons.

## Complexity

- Several features have required complex solutions to implement in Pygame. For example, Pygame will not automatically print text into neat paragraphs, and instead draws one long line of text onto a separate surface and draws *this* surface onto the main one. My listOfLines function therefore cuts text into lines of an appropriate length and my createTextSurface function pastes these one by one into the appropriate positions on the main screen, centred. This allows the user to print text into neat paragraphs. For further examples, see the implementation of draggable objects in 'causationProgram.py', or equal assignment to conditions using setCondition in 'auxiliaryFunctions.py'.