Computer Programming 23/24

Candidate Number: HJZW6
Module: PSYC0157

January 2024

1 Abstract

The program is to measure the response time of participants solving mazes while listening to music in different speeds, in an effort to understand whether auditory fluency has an impact on participants' problem solving capacity.

There are three speed options: fast, normal, and slow. Participants are asked to solve two mazes while randomly assigned to two different music speeds. The program will make sure that the same maze will never assigned twice to the same participant.

The program will collect three types of personal data from participants: 1) demographics including age, gender, ethnicity, educational backgrounds, and profession; 2) personality using Myers and Briggs' four dimensions questionnaire; 3) auditory or disorientation disabilities, in which case the data should be discarded.

The experimenter can alter the number of mazes participants are going to solve, and the program is capable to generate an unlimited number of new mazes automatically, while ensuring that music speeds and mazes are equally and randomly distributed as much as possible. The results will be saved in csv files for further analysis.

2 Experiment overview

2.1 Background

In this experiment, we will test the influence of processing fluency on a planning cognitive task. Processing fluency is defined as the ease to process information, such as perceptive fluency (visual ease such as font size or colour contrast) and contextual fluency (coherence, similar to social consensus), and it has been shown that processing fluency has an impact on judgment and decisions such as liking or truth.

2.2 Problem

What is less well known, however, is the impact of the processing fluency on a cognitive task such as problem solving and planning. Moreover, contemporary fluency measure is often a monotonous one-dimensional easy/difficult scale. In the real world, more nuanced fluency measure may exhibit a normal distribution rather than monotonous scale. For example, the ease of listening to music may be greatest for medium speed, and becomes more challenging if the music is played too fast or too slow.

2.3 Method

In this experiment, we aim at measuring the capacity of solving mazes while listening to the same music played at different speeds as processing fluency factors. The music will play at three different tempos: fast, normal, and slow, whereas the capacity of solving mazes will be measured by the time needed to exit mazes.

The choices of mazes as well as the speeds of music playing will be randomised. Participants will be asked to enter their ages, genders, ethnicity, educational backgrounds, professions, as well as some personality measures based on Myers and Briggs' four personality dimensions, and potential auditory or disorientation disabilities.

2.4 Technical Notes

The experiment results will be saved in two different files: a) in an aggregated file, where each participant attempted maze test will be recorded in one row, including participant's ID, their demographic information, the maze ID, the music speed, and the time needed to complete the maze; b) in a separate individual file, where each participant attempted maze test will be recorded in a standalone file, and each time the participant makes a move in the maze, its location (x, y coordinates) and the time elapsed since the last move will be recorded in each row. The more detailed record will allow us to carry out the analysis of more refined granularity.

3 Description of procedure

3.1 Participants' Data Collection

The recruited participants will be provided with an informed consent form, which will specify the experiment procedures and their rights. Once they agree to participate, they then need to fill out a demographic form, which includes their age, gender, ethnicity, education and employment backgrounds, as well as Myers and Briggs' personality traits, and whether they have spacial disorientation and auditory disabilities.

3.2 Experiments

Participants are then expected to solve two mazes by controlling a small leopard figure to exit the maze while listening to a pre-recorded music. The exit is marked by a trophy.

Once participants have completed the first maze, an acknowledgment screen will be presented, and participants will have the option to continue to solve the second maze or to quit the experiment. If they choose to continue, the second maze will be shown, which will be different from the first one, while the music will be played again at a randomised speed.

Once participants have completed the second maze, an end-of-experiment screen will be presented to thank them for their efforts, and then after they click on the exit, the program will be back to the initial screen again, ready for the next test.

3.3 Experiment Analysis

Participants' ID, demographic information, along with maze ID, music speed, and the time needed to complete the maze will be recorded in an aggregated file, as well as more detailed information of participant's each move, location, and time elapsed will be recorded in a separate individual file.

Note that mazes and music are randomly assigned to each test, such that the independent variables will be equally distributed as much as possible, under a precondition that no participants should do the same maze twice.

Experimenters then will analyse the aggregated file for general analysis, as well as more fine-grained individual files for detailed analysis such as acceleration or slowing down of movements in solving mazes, and how many unnecessary detours participants make during the maze test.

4 Experimenter's manual

4.1 Experiment Procedures

Experimenter should simply execute the main python file: **HJZW6.py**, and the program will execute experiments, play music, save results, and generate new mazes if necessary.

4.2 Experiment Results

All aggregated experiment results will be saved under the **\results** sub-folder in a csv format, where each row records the information of each maze test; and more fine-grained individual results will be saved under the sub-folder **\results \detailed_time_data**, where each single file records all positions and time data of each maze test.

4.3 Experiment Parameters

The default settings are stored in the file **params.py** under the sub-folder **\src**. Experimenters can modify the following parameters:

- Experiment mode:
  Current format: $maze\_dev = False$, which means the program is ready to be tested on participants. If experimenter simply desire to test program procedures for QA purpose, set $maze\_dev = True$, which will result a maze with a straight passage to save time for QA.
- Music mode:
  Current format: $music\_on = True$, which means the program will play music while participants solving mazes. If experimenter desires to establish benchmark without music, set: $music\_on = False$.
- The number of mazes each participant will solve:
  Current format: $num\_tests = 2$, which means each participant can do up to 2 mazes. Experimenter can change it to any desirable number n by: $num\_tests = n$. For example, $num\_tests = 10$ will let each participant do up to 10 mazes.
- The ID of mazes to be tested:
  Current format: $maze\_ids = [1, 2]$, which means maze no.1 and maze no.2 will be tested, and they will be randomly randomly equally distributed to participants. Ideally the number of available mazes should

be greater than the number of mazes each participant will do. Experimenter can add more mazes by: $maze\_ids = [1, 2, 3, ..., n]$. For example, $maze\_ids = [1, 2, 3, 4, 5]$ will provide 5 different mazes to be randomly distributed to participants. Note that corresponding mazes are stored as files: `maze_1.txt`, `maze_2.txt`, `maze_3.txt`, etc, under the sub-folder **\mazes**. If no correspondent maze files are found, the system will automatically generate and save mazes to files for test. Experimenter do not need to worry about designing new mazes.

- The speed of music to be tested:
  Current format: $music\_speed = [1, 2, 3]$, which means the music speed of 1 (slowest), 2 (medium), and 3 (fastest) will be tested, and randomly equally distributed to participants. Note that the corresponding music pieces are stored as pre-recorded files: `myMusic_1.mp3`, `myMusic_2.mp3`, `myMusic_3.mp3` under the sub-folder **\music**. If experimenter intends to use a different kind of music, or want to test more a nuanced speed scale with more than 3 options, they must generate the corresponding music files under mp3 format in this sub-folder.

- The maze dimension:
  The maze is designed to be a square maze and its dimension is defined by two parameters: 1) block size, which is a square to represent a wall or an empty space, with the format; $block\_size = 44$, where 44 is each block size in pixel; 2) the number of blocks each side, with the format: $dim = 17$, which is the maze dimension in blocks; there are thus 17 blocks on each side of a maze. Experimenter can change the maze dimension (default 17) in terms of the number of blocks on each side of a maze to n, under the condition that $n = 2 \times m + 1$, where m is an integer. However, if experimenter desires to change the block size, they would need to change the corresponding images, saved under the sub-folder **\images**, and convert the images of the wall, the leopard, and the trophy from the height and width of 44 pixels to the chosen size.

- Legal age to participate:
  The default legal age to participate in the experiment is 18, with the format: $legal\_age = 18$. Experimenter can change the legal age if deemed appropriate.

5 Program highlights

The program has three modules: 1) MazeApp Class under **\src\app**; 2) Player Class under **\src\player**; 3) Menus Class under **\src\menus**. It also implemented design patterns like Abstract Factory and Factory, Mixins, functional wrapper, lambda functions, and multiple inheritance (with debatable merits).

A few highlights are listed below:

- MazeApp Class under **\src\app**:
  A class that coordinates Menus class, Maze class and Player class to control and update maze experiment procedures. There is an implementation of **Abstract Factory and Factory** design patterns.

- Maze Class under **\src\maze**:
  A class that randomly and equally distributes mazes to participants, and selects, loads and updates mazes. There is an implementation of **Abstract Class**, with the hierarchy: ABC → MazeABC → Maze / MazeTest.

- MazeGenerator Class under **\src\maze**:
  A class to create and save new mazes into files, if the required IDs of mazes cannot be found in pre-saved files. It is more a demonstration of **algorithms** and Python problem solving rather than pattern design. It is the logically most challenging class.

- Player Class under **\src\player**:
  A class that controls the player (shown as a leopard in a maze) movements, ensures that: 1) the movements are discrete and not continuous (i.e., the player moves in a maze block by block, and not pixel by pixel); 2) the movement speed can be followed by eyes; and 3) the player cannot go through the walls even if the detected key press events would cause the player to jump from one side to the other. There is an implementation of functional **wrapper** applied to the PlayerHelper Class, with **lambda functions**. PlayerHelper Class is inherited by the Player Class.

- ExperimentMemory Class under **\src\experiment_control**:
  A class that tracks realised tests, including maze IDs, music speeds, and participants' IDs, such that mazes and music speed can be equally and randomly distributed, under the condition that a participant will not do the same maze twice. No fancy designs in this class.

- Menus Class under **\src\menus**:
  A class that controls page orders presented to participants, and logic of the flow, and elicits participants' input information. There is an implementation of **Mixins** design patterns and multiple inheritance, with the hierarchical structure is: pygame_menu.Menu → CustomMenu → Mixins (multiple classes) → Menus.