Card Sorting System Design

Alexandra Coroiu (s1841742)
a.coroiu@student.utwente.nl
University of Twente
Pre-master: Psychology
Module: Human Factors and Engineering Psychology

Table of Contents

System

Background

Card sorting is a useful method for getting insight into users' mental model. It is often used in the design process for a variety of applications. A good example is the use of card sorting in the development of web applications as it can help with structuring information for menus and navigation (Spencer D., Warfel T., 2004). Card sorting involves making a potential user group different cards into categories. There are several card sorting variations that can be used. Closed card sorting already provides the categories while open card sorting lets the participant create the categories during the experiment. Single level card sorting only allows the participant to sort the cards into one hierarchical levels, while multi level card sorting provides the opportunity of defining subcategories, resulting in a deeper hierarchical structure (Wood, J. R., & Wood, L. E., 2008). The data resulted from card sorting can be presented in the form of tree structures. Different types of analysis can be used to make sense of this data and create a users' mental model (Fincher, S., & Tenenberg, J., 2005).

Card sorting is usually done physically using paper cards and a suitable space (e.g. a big working table or a magnetic board) that allows participants to group the cards into categories. Lately, several card sorting implementations have emerged to provide a digital alternative to the physical experiment. (Chaparro, B. S., Hinkle, V. D., & Riley, S. K. 2008).

Requirements

This design document presents the implementation of a new card sorting system. The main goal of this system is to provide a computer program for **multi-level hierarchical card sorting**. The general system requirements of such a program are defined bellow:

1. The system should provide a graphical user interface that allows te user to group cards into categories
2. The system should allow open and closed card sorting
3. The system should allow single and multi-level card sorting
4. The system should provide generated results that can be used for analysis

The main actors of this system are: the researcher and the participant. An extensive list of functional requirements for each actor can be found in Appendix 1. The functional requirements are prioritized on a scale from 0-3 (requirements with 0 having the lowest priority and 3 the highest). The functional requirements with the highest priority ensure the basic functionality of a card sorting system: inputting cards, sorting the cards into categories in an open and single-level card sorting experiment and saving the results in a simple format. The next functional requirements enhance the user experience by providing a welcome/goodbye screen and instructions, and allow more complex functionality and flexibility like multi-level card sorting, the saving of more complex processed results and removing a card from a category after it has been sorted. The last two categories of requirements with the lowest priority are focused on further improving user experience through more advanced user interactions, user control and customization. An initial mock up of the GUI can be found in Appendix 2.

Implementation

The Card Sorting System has been designed and implemented during a five weeks time period as part of a programming course at the University of Twente, Netherlands. The system was developed according to the curriculum of the course. It is implemented in python 3.8.3 and it mainly makes use of standard python libraries (e.g time) and the py.game library. In addition, pandas library is also used for easy reading and writing into external files. The README file provides all the required information about what is needed to run the program.

Packaging

The package contains several parts which are described bellow. Each file is also explained, but further detail and comments can be found in the source code itself.

1. *README.txt - contains the installation and run instructions*

2. **Main program files**

Card_sorting.py: This is the main file used to run the Card Sorting program. This module uses the painter and event_handler to interact with the user. It uses a loop to constantly print and listen to user input and updates the program state.

Data_manager.py: This module deals with the tree data structure that represents the cards and categories of the experiment. It uses a self defined tree class. It reads from the input file, creates and writes results data to the output files. It uses pandas to easily read and wite data.

Event_handler.py: This module deals with the input from the user for each state. It updates the its data structure and state variables according to the user actions. It defined checks actions according to positions of buttons from painter. It communicates the actions to the card_sorting to determine when to change program states.

Painter.py: This module prints output to the user. It draws the pages and all the elements on them for each program state. It uses the button classes from UI_objects to draw clickable elements. It always prints up-to-date information by constantly updating its variables according to event_handler values received as parameters to its functions.

UI_objects.py: This module defines the clickable elements of a page. It uses a self defined button superclass and other button subclasses.

Program_invariables.py: This module contains all the program invariables. It uses different Enums to define them. This modules is used by all other modules (except data_manager).

Instructions.txt: This file contains the instructions texted displayed on the instructions page.

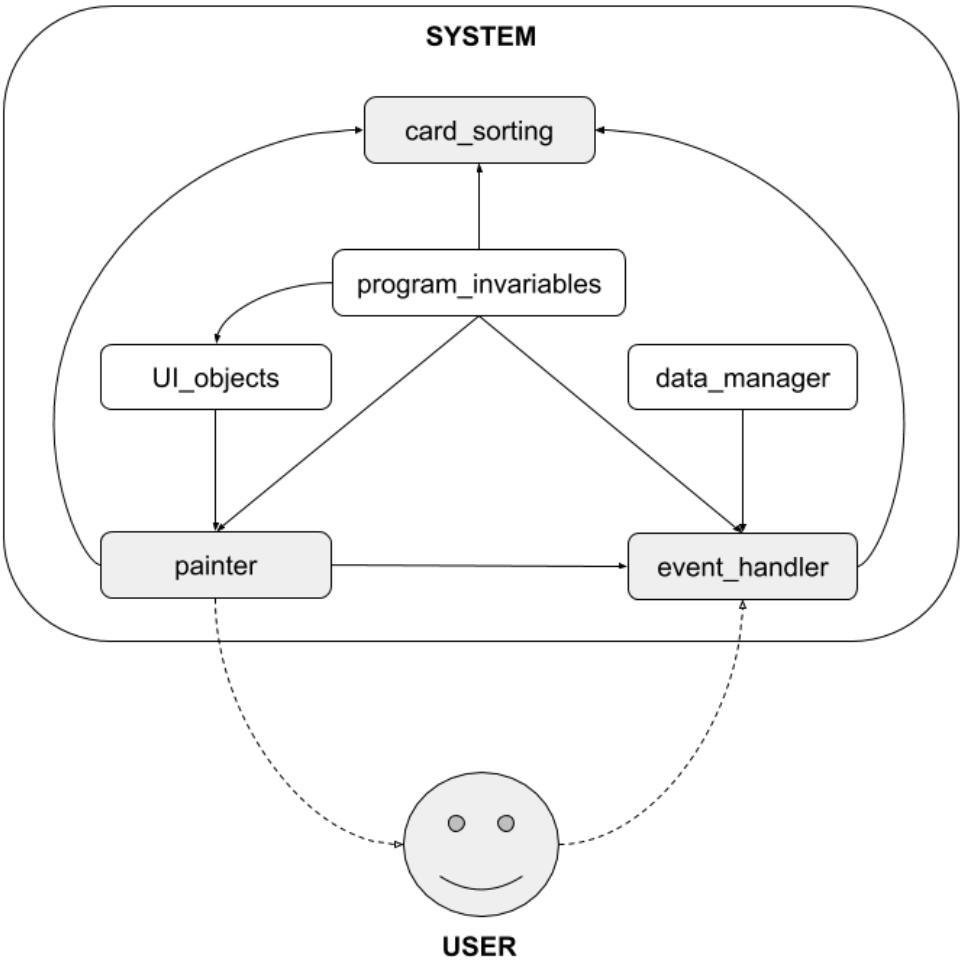

Figure 1: Diagram of the main python modules

The python modules follow a Model-View-Controller pattern with card_sorting.py and event_handler.py representing the controller, painter.py and UI_objects.py the view, and data_manager.py and program_invariables.py are the model.

3. Example input/output files

Cards_file.csv: This file contains a list of cards, each of them as a new row.

Results_file.txt: This file contains a textual, visual representation of the resulted data tree after sorting the cards into categories on multiple levels.

Matrix_file.csv: This file contains the results in the form of a similarity matrix calculated based on the data tree containing the cards into categories on multiple levels.

4. Test files

Test_cards_file.csv: This file contains a list of cards used for testing purposes.

Test_data_manager.py: This module tests the functionality of the data_manager. It creates a data structure that can be used to generate results.

Test_matrix_file.csv: This file stores the matrix results of the test run.

Test_results_file.txt: This file stores the text results of the test run.

Functionality

The functionality of the Card Sorting system expends over different program states that the user can navigate between. A full transition table containing states and substates of the program can be found in Appendix 3. A walkthrough with screenshots for all of the pages and subpages (associates with the states and substates of the program) can be found in Appendix 4.

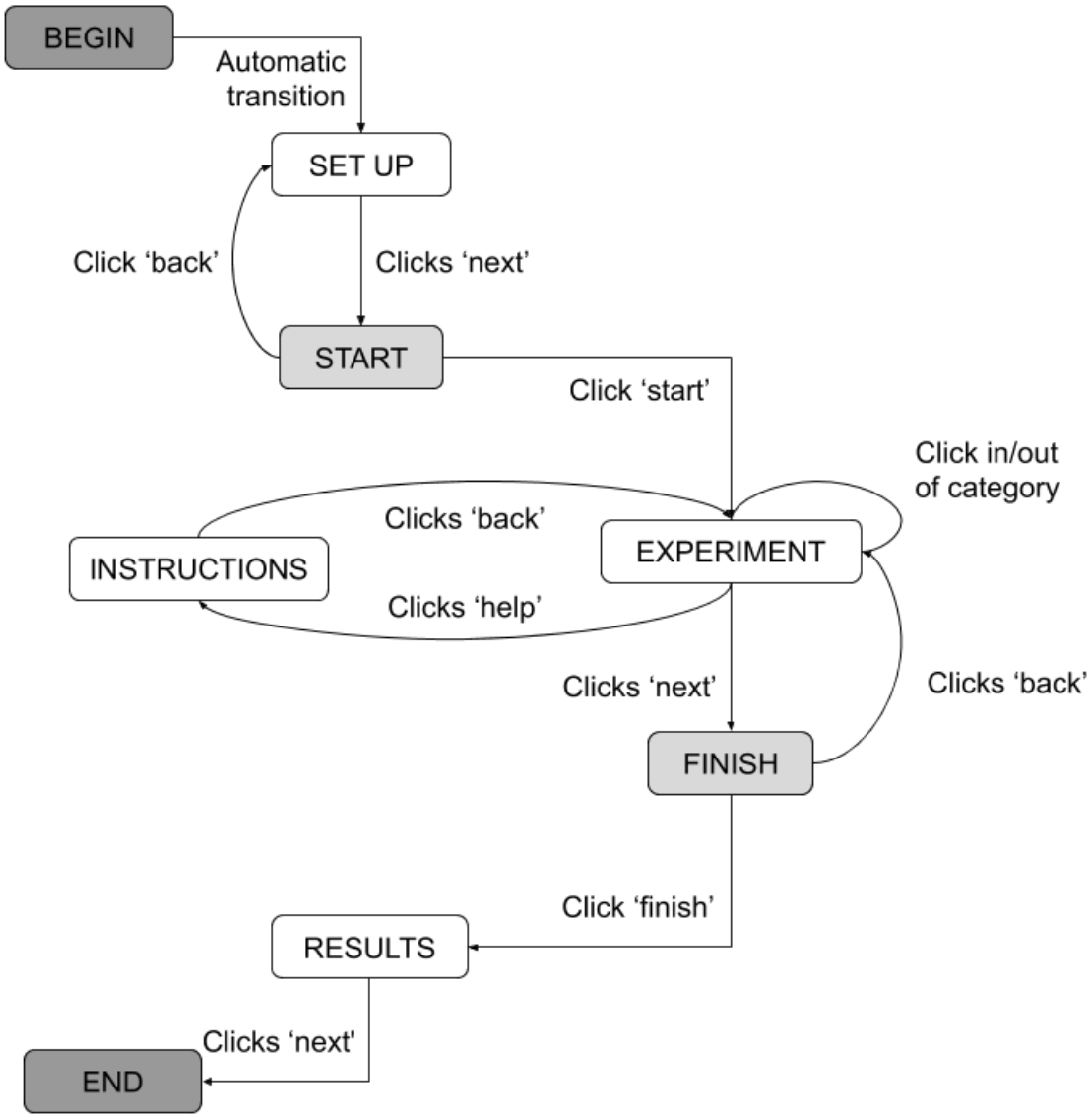

Figure: Program States diagram

Before/after experiment

The system features a Welcome screen for 5 seconds at the beginning, while input data is loaded from a .csv file using the panda library. A sanitizing function is also used when reading the input to remove potential duplicate cards. The loaded input cards are displayed in the Set Up page. The cards are represented as a tree data structure in the model. This tree gets updated during the experiment according to how the participant sorts the cards into categories and subcategories.

At the end of the experiment, the tree is converted into result data. A Good Bye screen is featured for 5 seconds and the results data is written into the output files. A simple textual representation of the data tree structure is saved in the .txt output file and a similarity matrix is saved in the .csv output file. Results cannot be calculated without the cards being sorted and blank categories are not taken into consideration for the results. An example of a similarity matrix and text results can be found in Appendix 5. The Results page points the user to the right results files. The similarity matrix is calculated using the Jaccard similarity coefficient for two cards according to the formula:

$$Jaccard\ index\ =\ \frac{|A \cap B|}{|A|+|B|-|A \cap B|}$$

A and B are sets. The set A is the set which contains all the categories in which the first card is present and set B, all the categories in which the second card is present. The intersection of A and B are the common categories for two cards. The cardinality of these sets is equal to the level at which we can find the the cards in the tree. For the first card, the cardinality of set A is the deepest level we can find this card at. The cardinality of the intersection is equal the deepest level we can find both cards at. The similarity matrix is calculated based on a level matrix containing these values. The tree base level is 0 and each new level increases by 1.

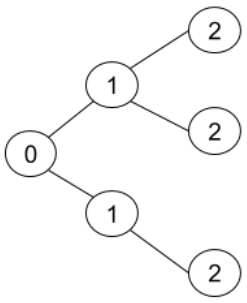

Figure 2: Data tree levels

The Welcome, Set up, Results and Good Bye states of the program are intended for the researcher, but a participant can run the whole program by themselves as well. The Start and Finish page are used for confirmation, to make a clear transition between the pages intended for the researcher and the ones for the participant.

Experiment

The experiment page is the most complex in terms of functionality. It allows the user to select unsorted cards and move them into categories. The experiment features a number of already created empty category tabs that span across multiple pages. All of these categories can be filled with cards until there are no more unsorted cards. Once all cards are sorted on one level, a new level can be opened, so the user can go into a category and further divide the cards in subcategories. This process can be repeated as many times as the user wants. The navigation between categories and subcategories, and whether a participant has finished sorting or not, creates a new set of substates for the experiment. These states define the working board which contains the sorted and unsorted cards at one level of the tree. Each category is a new child for the base tree. Subcategories are children to the category in the data tree. Each category in the tree holds its level, its cards, and its children. Each child is a tree in itself. The last levels of the tree does not have any more children. Any tree is a subcategory if its level is greater than 0. The base and any (sub)subcategory always start as new, but once one card has been sorted, all of the cards have to be sorted in order to reach the final state and continue beyond this current (sub)category.

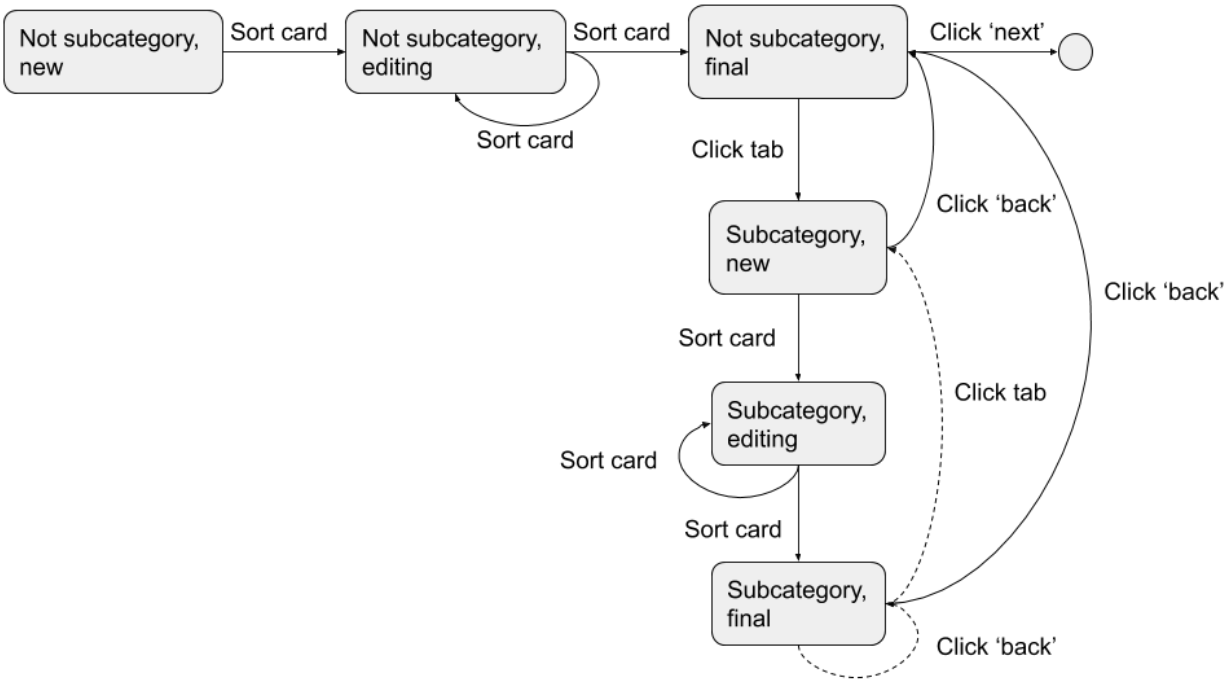

Figure: Working Board States

The instruction page can be accessed at any time from the experiment (independent of the state of the working board) and contains simple instructions for the user on how to sort cards.

Reflection

All of the basic functional requirements (priority score 3) have been fulfilled. Most of the other important requirements (priority score 2) have been achieved as well. Two requirements from this category remain:

1. The design for removing a card from a category after it has been sorted has been made, however the implementation would highly increase the complexity of the program. Editing a category would affect all the subcategories, so new working board states should be used for the experiment to assure valid results. In Appendix 6 a new displays the working board states containing a new “done” state (when all the cards have been sorted, but the user has not confirmed the final state yet). This state would allow the user to remove cards from a category and add it to another, before starting to create subcategories.
2. Creating a new category could be implemented, however at the moment the experiment can already start with a high number of empty categories that the user can freely use to sort. Being able to create a new category with a given name, would be a nice addition for the future.

Almost all functional requirements scored 1 on the priority scale have also been fulfilled. These mainly improve the system flow by adding additional states for the researcher. For the participant, the current implementation allows the visualization of the unsorted cards. Currently the program is set to only display one. This can be easily changed according to the needs of the experiment. Three requirements remain from this category:

1. Moving cards by dragging can be implemented using pygame, it can be easily built on top of the current program, but it is not a necessary improvement. The select/deselect functionality is a good replacement at the moment.
2. The results page at the moment only points to the results files. Displaying the full results in the results page might not be desirable because a high number of cards would make it impossible to display the full tree or matrix on the current screen size. A summary of the results could be displayed, or the screen could be bigger in an improved version.
3. The input file name is set, so the user always has to name the file accordingly. An alternative to this would be adding the file as a run argument if the program is ran from the command line.

Finally, none of the functional requirements with priority 0 have been implemented. These requirements would add a lot more flexibility for the user by allowing extensive customization. The current state of the system successfully fulfills the system requirements. The GUI allows the user to easily sort cards into a single or multi-level hierarchy during an open card sorting experiment. The generated results can be used for further analysis (e.g. matrixes from multiple users can be added into a heatmap). The only main functionality that the program is lacking right now is the ability to conduct a closed card sorting experiment. The feature of already imputing categories easily though the interface is not available yet. With the current state of the program, a tech-savvy researcher could hardcode predefined categories into the code before each experiment.

Overall, the current state of this Card Sorting system bring forward a useable program with extended features and documentation. It has a minimalist and consistent design that makes it easy to use. The user is always up to date with the information about the data structure and the system states. The limitations of each state are clear to the user and no invalid actions can be performed. There are clear indications of what can be done on a page for each specific state or substrate: buttons only become visible when they can be clicked, categories can only be entered after sorting is finished, the experiment cannot be finished before sorting all cards. These measures help with error prevention. However, one error that is not accounted for is if the user accidently closes the program, or the computer crashes. In these cases the partial state of the board would not be saved. This would be an important feature to improve on. FInally, the display for all pages is limited by the amount of lines that fit on the screen. The current size of the screen has been selected so that it can work on small laptops as well. All variables related to the display, positions and sizes of elements in pages can be changed from program_invariables.py. Generally, there shouldn't be any adjustments needed for running the program.

References

Chaparro, B. S., Hinkle, V. D., & Riley, S. K. (2008). The usability of computerized card sorting: A comparison of three applications by researchers and end users. Journal of usability Studies, 4(1), 31-48.

Fincher, S., & Tenenberg, J. (2005). Making sense of card sorting data. Expert Systems, 22(3), 89-93.

Pandas - Python Data Analysis Library. Retrieved January 21, 2021, from https://pandas.pydata.org/

Pygame. Retrieved January 21, 2021, from https://www.pygame.org

Python programming language. Python.Org. Retrieved January 21, 2021, from https://www.python.org/

Spencer, D., & Warfel, T. (2004). Card sorting: a definitive guide. *Boxes and arrows*, *2*, 1-23

Wood, J. R., & Wood, L. E. (2008). Card sorting: current practices and beyond. Journal of Usability Studies, 4(1), 1-6.

Appendix 1

actor	functional requirements	priority	progress
participant	visualize current card	3	2
participant	visualize categories	3	2
participant	put card in category	3	2
participant	visualize sorted cards in categories	3	2
researcher	save raw results	3	2
researcher	input cards	3	2
participant	read instructions	2	2
participant	use subcategories	2	2
researcher	save processed results	2	2
researcher	see good bye screen	2	2
researcher	see welcome screen	2	2
participant	create new category	2	
participant	remove card from category	2	1
participant	visualize unsorted cards	1	2
researcher	finish experiment	1	2
researcher	start experiment	1	2
researcher	visualize input cards	1	2
participant	move card by dragging	1	
researcher	visualize results	1	1
researcher	add input as run arguments	1	
participant	use adaptive GUI elements	0	
participant	skip card from deck	0	
researcher	customize results file name	0	
researcher	input categories	0	
researcher	add input through GUI	0	
researcher	choose single/multi-level experiment	0	
researcher	choose open/closed experiment	0	
researcher	save experiment set-up	0	
researcher	add participant name	0	

Table: Functional requirements

Appendix 2

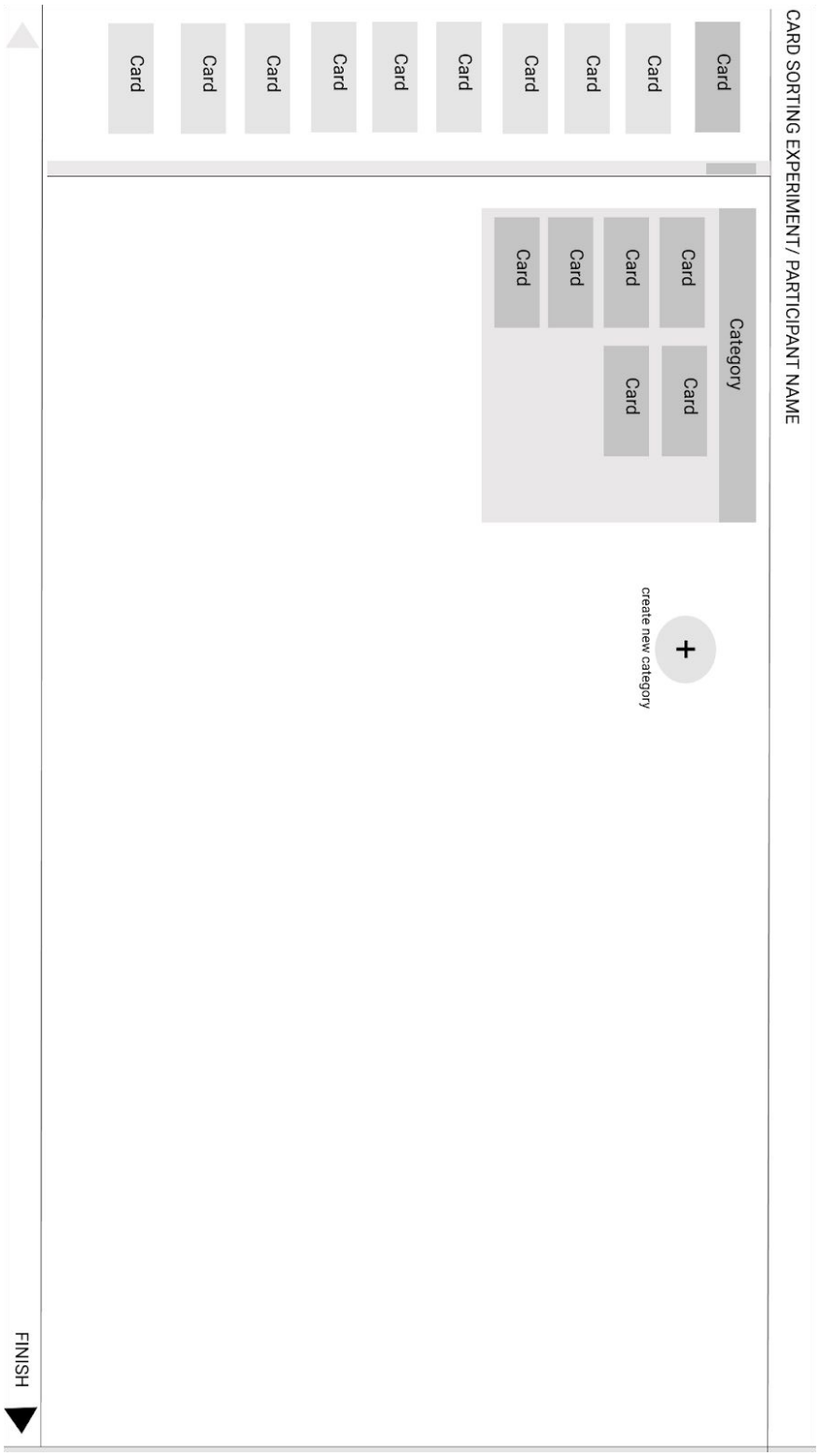

Figure: Initial Mock-up of the Card Sorting Experiment page

Appendix 3

		future state												
		BEGIN	SET UP	START	EXPERIMENT-B-N	EXPERIMENT-B-E	EXPERIMENT-B-F	EXPERIMENT-S-N	EXPERIMENT-S-E	EXPERIMENT-S-F	INSTRUCTIONS	FINISH	RESULTS	END
current state	BEGIN	-	auto	-	-	-	-	-	-	-	-	-	-	-
	SET UP	-	-	next	-	-	-	-	-	-	-	-	-	-
	START	-	-	back	start	-	-	-	-	-	-	-	-	-
	EXPERIMENT-B-N	-	-	-	-	sort	-	-	-	-	help	-	-	-
	EXPERIMENT-B-E	-	-	-	-	sort	sort	-	-	-	help	-	-	-
	EXPERIMENT-B-F	-	-	-	-	-	-	tab	-	-	help	next	-	-
	EXPERIMENT-S-N	-	-	-	-	-	back	-	sort	-	help	-	-	-
	EXPERIMENT-S-E	-	-	-	-	-	-	-	sort	sort	help	-	-	-
	EXPERIMENT-S-F	-	-	-	-	-	back	tab	-	back	help	-	-	-
	INSTRUCTIONS	-	-	-	back	back	back	back	back	back	-	-	-	-
	FINISH	-	-	-	-	-	back	-	-	-	-	-	finish	-
	RESULTS	-	-	-	-	-	-	-	-	-	-	-	-	next
	END	-	-	-	-	-	-	-	-	-	-	-	-	-

Table: State transitions. There are 6 experiment states representing the substates of the working board as well. B stands for base (not subcategory), S stands for subcategory, N stands for new, E stands for editing and F stands for final.

Appendix 4

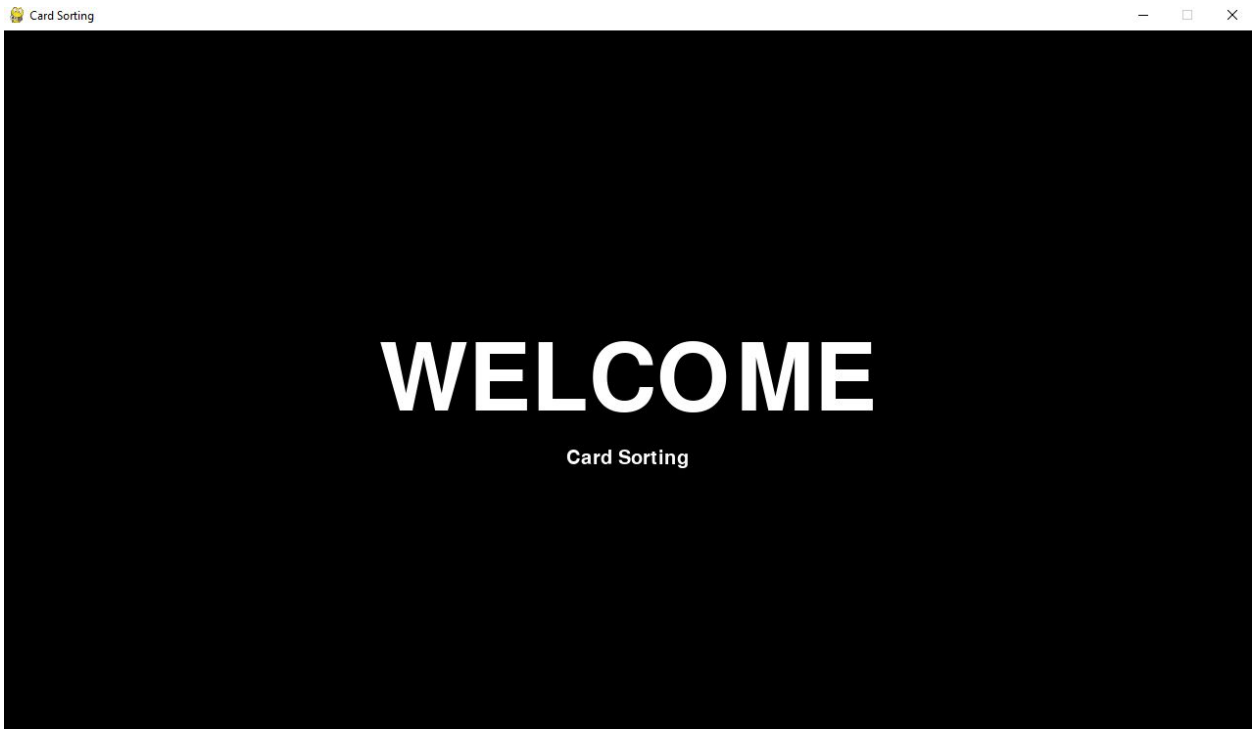

Figure: Welcome page

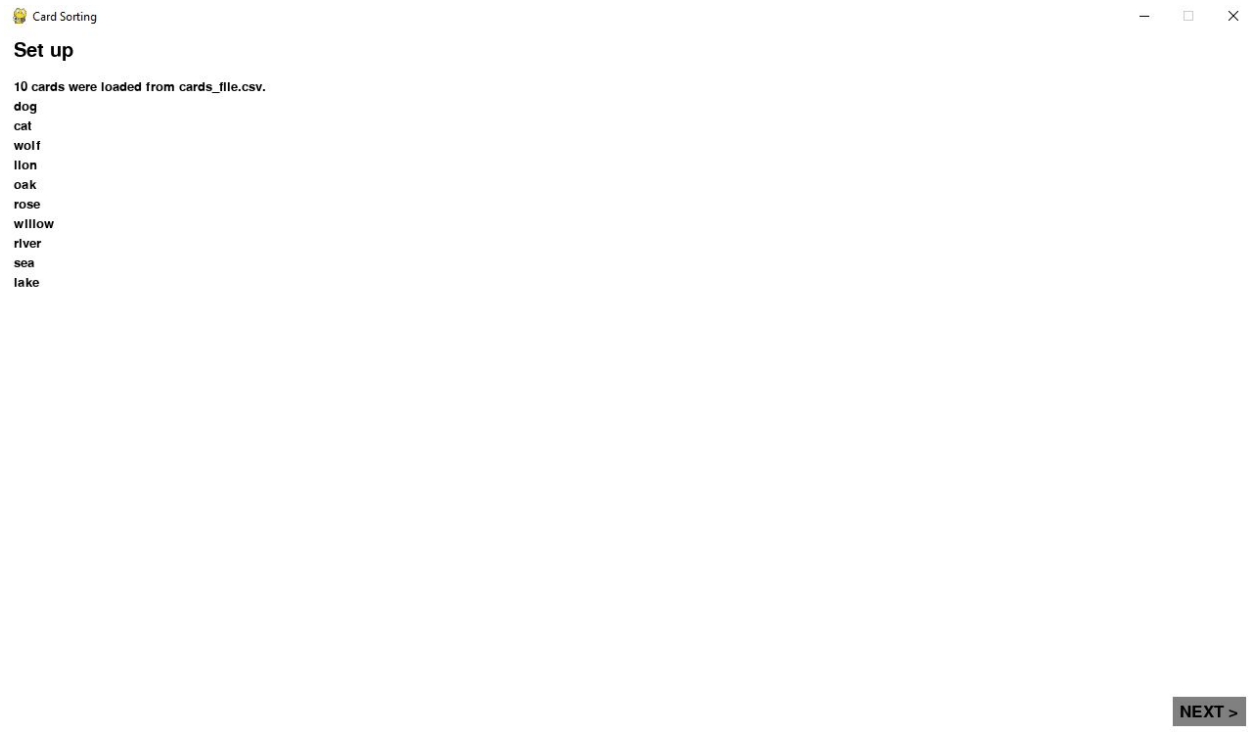

Figure: Set up page

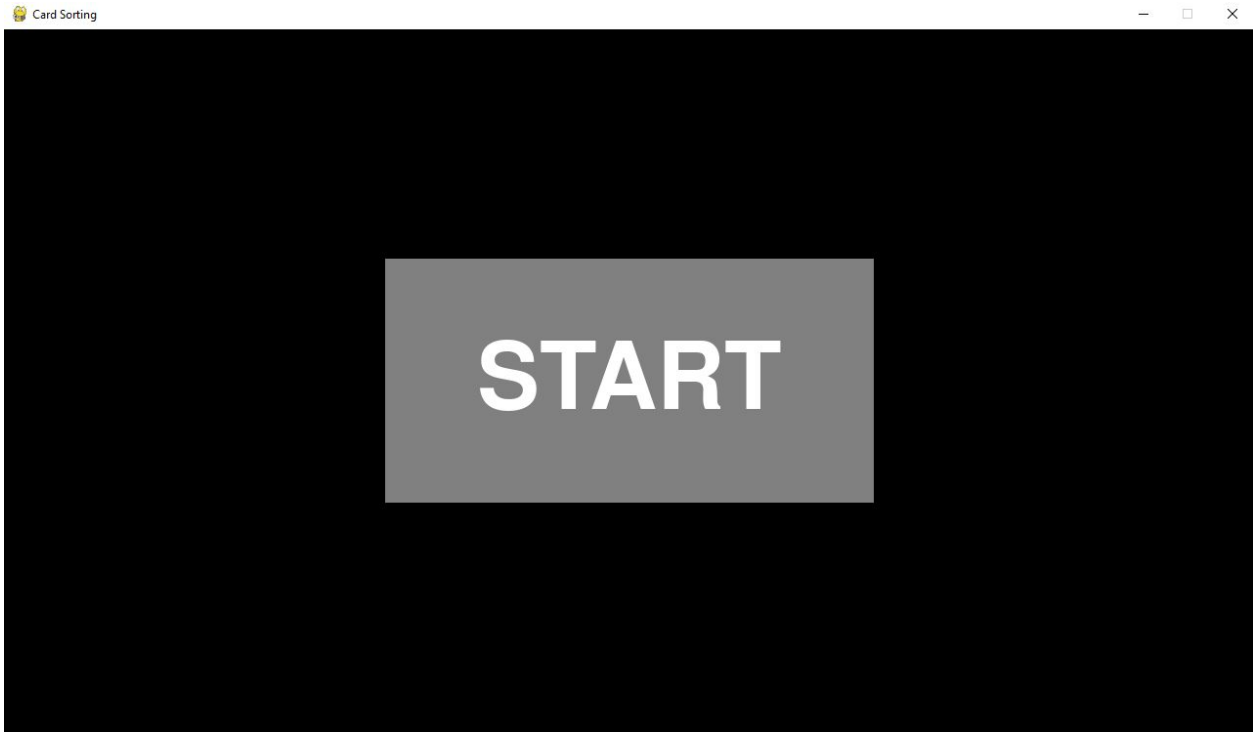

Figure: Start page

Figure: Experiment - subage: not subcategory (base), new (no card selected, first category page)

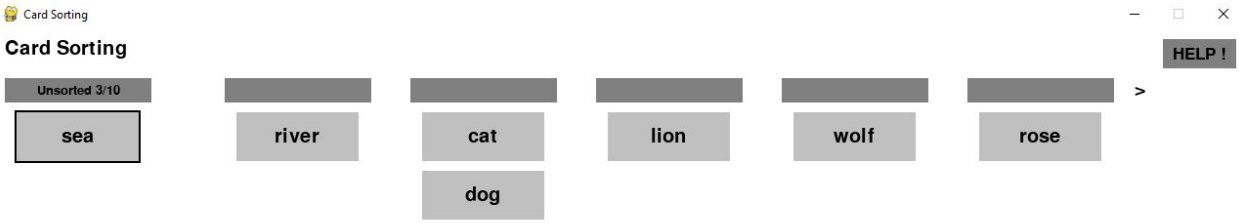

Figure: Experiment - subage: not subcategory (base), editing (sea card selected, first category page)

Figure: Experiment - subage: not subcategory (base), final (no card selected, second category page)

Figure: Experiment - subage: subcategory, new (river card selected, first category page)

Figure: Experiment - subage: subcategory, editing (sea card selected, first category page)

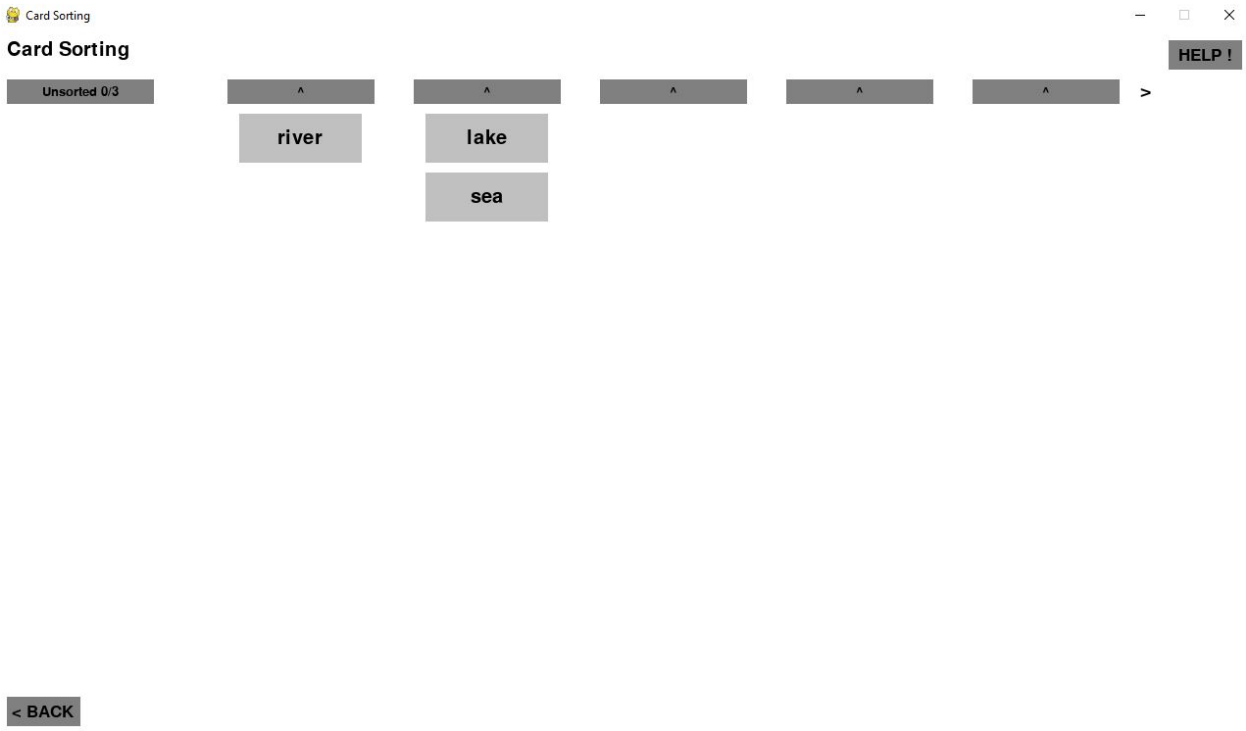

Figure: Experiment - subage: subcategory, final (no card selected, first category page)

Figure: Instructions page

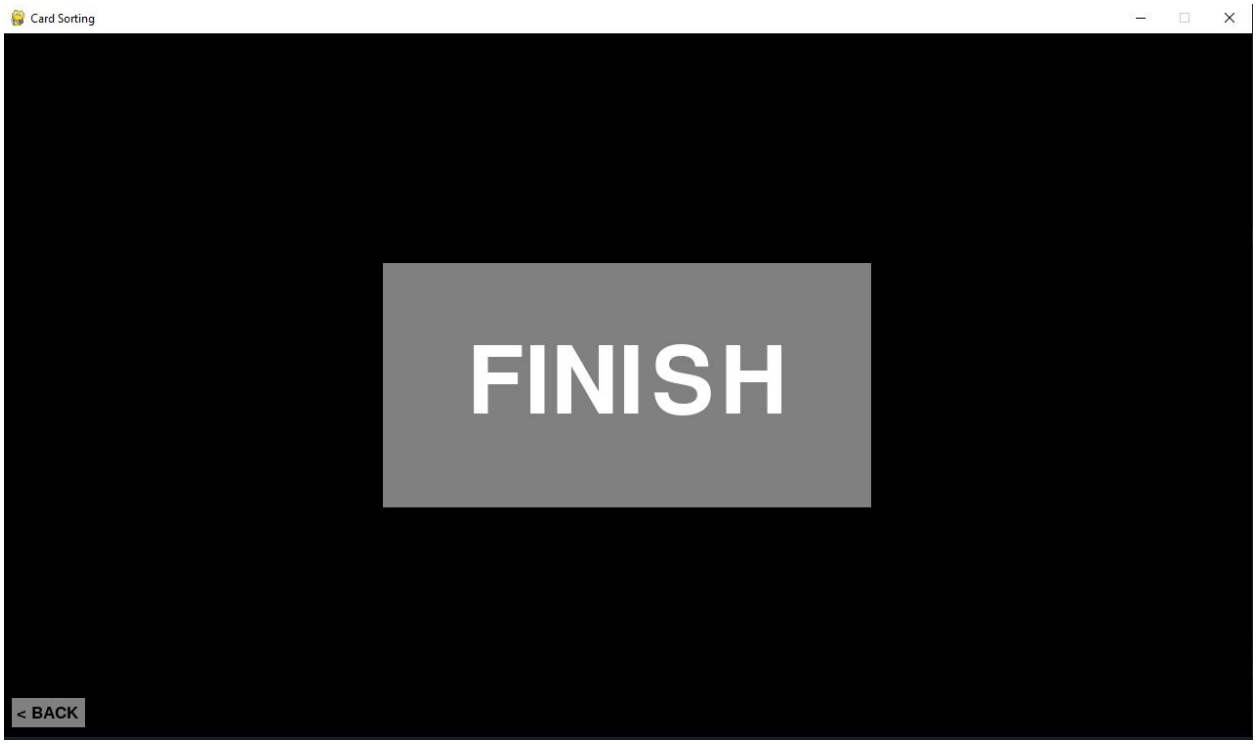

Figure: Finish page

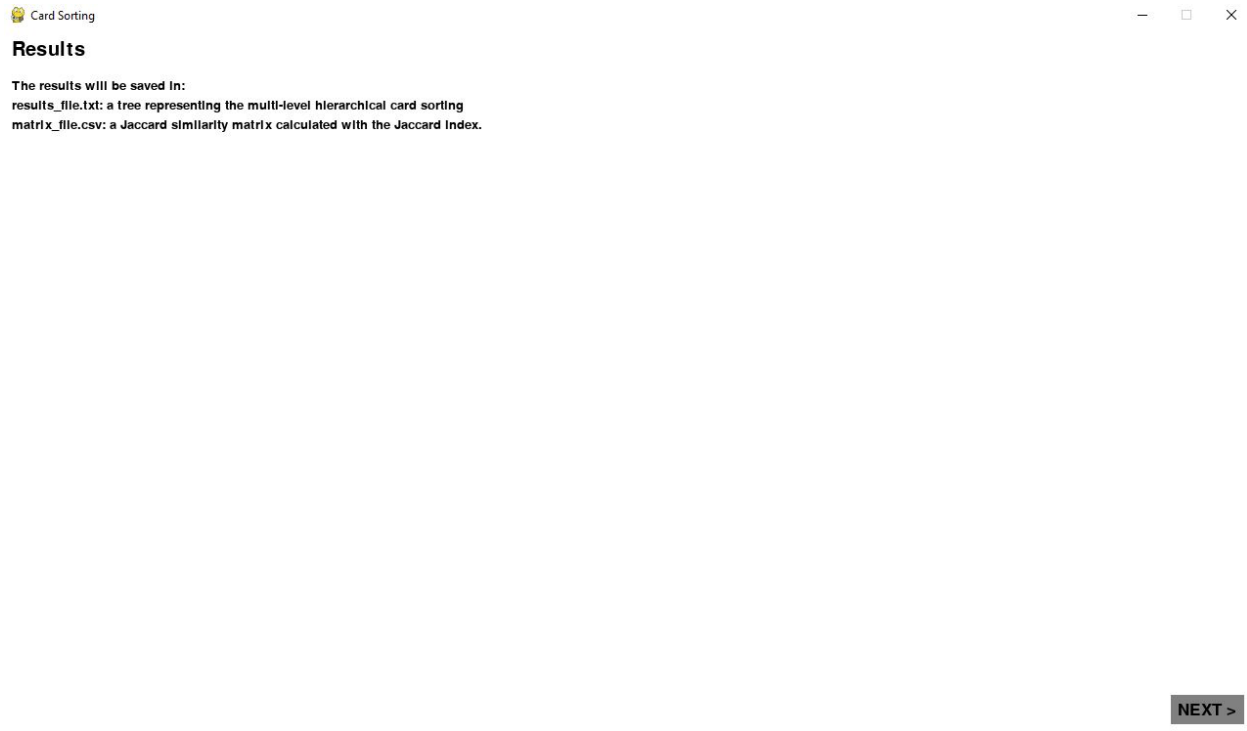

Figure: Results page

Figure: Good bye page

Appendix 5

```
: [dog, cat, wolf, lion, oak, rose, willow, river, sea, lake]
        : [river, sea, lake]
                : [river]
                : [lake, sea]
                        : [lake]
                        : [sea]
        : [cat, dog]
        : [lion]
        : [wolf]
        : [rose]
        : [oak, willow]
```

Figure: Example text result

	dog	cat	wolf	lion	oak	rose	willow	river	sea	lake
dog	1	1	0	0	0	0	0	0	0	0
cat	1	1	0	0	0	0	0	0	0	0
wolf	0	0	1	0	0	0	0	0	0	0
lion	0	0	0	1	0	0	0	0	0	0
oak	0	0	0	0	1	0	1	0	0	0
rose	0	0	0	0	0	1	0	0	0	0
willow	0	0	0	0	1	0	1	0	0	0
river	0	0	0	0	0	0	0	1	0.25	0.25
sea	0	0	0	0	0	0	0	0.25	1	0.5
lake	0	0	0	0	0	0	0	0.25	0.5	1

Table: Example matrix result

Appendix 6

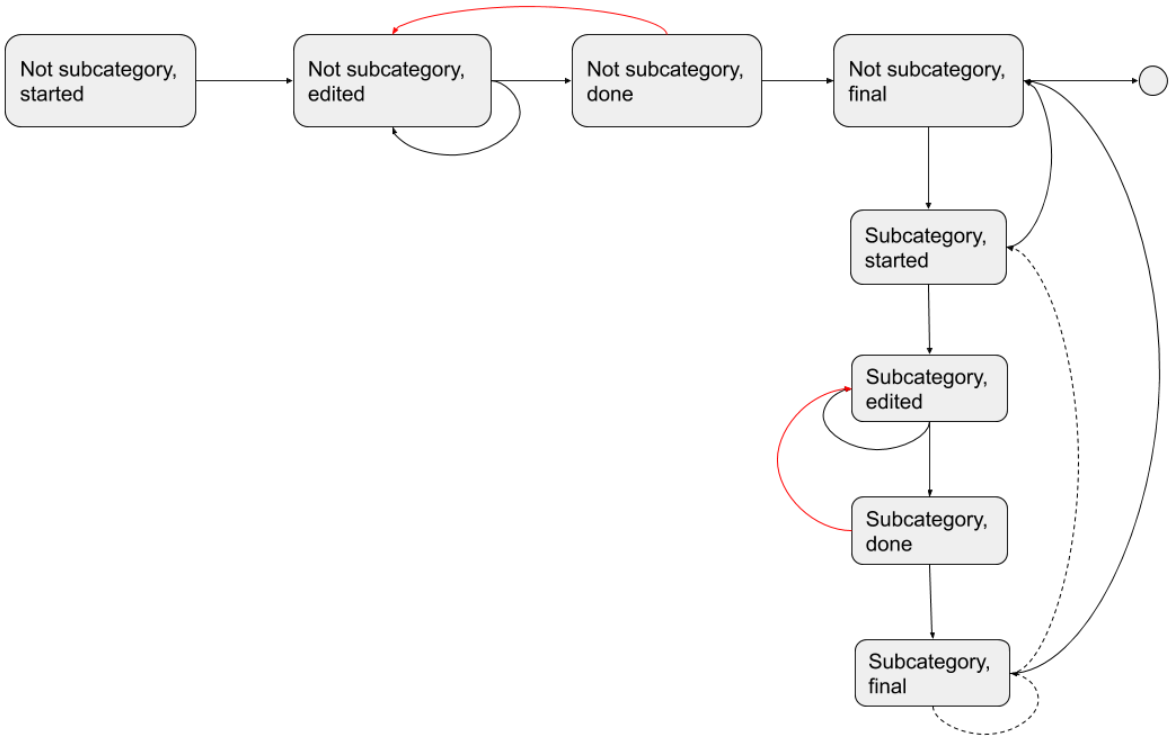

Figure: Working board states that allow removing a card from a category. The additional transitions that would be added on top of the current one are highlighted in red.