VRIJE UNIVERSITEIT BRUSSEL

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Computer Science

INVESTIGATING ALGORITHMS FOR HELPING END USERS CREATE BETTER USER INTERFACE DESIGNS

MENGYAO SONG

Academic year 2021–2022

Promoter: Prof. Dr. Beat Signer and Dr. Audrey Sanctorum
Advisor: Dr. Audrey Sanctorum

Faculty of Sciences and Bio-Engineering Sciences

Abstract

Advances in technology have greatly gone about transforming the way people interact with the world. In daily routines, people need to deal with mobile applications in the typical sense in addition to managing or monitoring wearable, portable, even implantable Internet of Thing devices. The demand for a system that can handle all these smart things in one place has been tremendously influenced by the growth of the Internet of Things and the applications. To make it more convenient to interact and manage smart devices and Internet of Things, Sanctorum developed an authoring tool, eSPACE, for the personalised management of smart devices and Internet of Thing applications. Our work extends the UI design view of the eSPACE authoring tool, which provides an environment for end users to design their own user interfaces (UIs).

The reason to extend the UI design view of eSPACE is that, although it allows end users to create UIs, there is a lack of guidance for them. Blind exploration may lead to user interfaces that do not conform to design principles and thus disengage users. Aimed at providing guidance for end users, in this dissertation, we investigate an extension to eSPACE's UI Design Tool, a graphical user interface (GUI) design tool that facilitates the UI design process and provides a recommended solution depending on the current design of the end. After dragging and dropping the UI elements that suits their demands, end users can manually select the elements to be rearranged and activate the appropriate functions according to their needs. In order to be able to organise elements flexibly and neatly, we implemented several functions, for illustration, fix Overlapping Issues, equally space elements, and align elements. Apart from this, our tool is able to automatically infer the type and the number of elements present in the solution and give real-time UI recommended suggestions based on this inference.

After analysing the systems in the related research field and simulating actual user scenarios, our requirements to be met by our tool is derived. Our tool does not only cater for the need to edit the position of specific elements in relation to each other, but can also make recommendations based on the overall layout. Compared to a traditional design tool, our design tool fulfils the requirements to design personalised applications more efficiently and with adherence to design guidelines.

Acknowledgements

Fist and foremost, I would like to express deepest appreciation to my promoters Prof. Dr. Beat Signer and Dr. Audrey Sanctorum for their invaluable patience and feedback. Whenever I encounter a technical problem that I cannot solve, they always point me in the right direction to help me overcome it. I am very grateful to them for taking time out of their busy schedules to come and meet with me every week, which helped me complete my project effectively on time. I could not have undertaken this journey without the guidance from them.

Besides, I would like to thank my family for their unstinting support, not only material but also spiritual. It has been a extremely hard years for both me and my families, but thanks to their unconditional support I have been spared from a shortage of money. Regular video communication has been the biggest relief since I've been living alone. Without that support I could not succeeded in completing my Master's degree. Maybe they won't read these words, but I still want to tell them that you are the best parents in the world.

In addition, I must thank my friends for the tremendous moral support they have provided. It is thanks to every happy moment with you that I have been able to persevere in my postgraduate studies. I will always remember the time I spent in our apartment, where I met two really lovely roommates. They are the cure for my mood swings. We seem to do everything together. We cook together, we walk together, and we watch movies together. I also want to express my gratitude to my friends for their concern and for helping me feel less alone. Whenever I face challenges in life, their suggestion always comes in handy.

Last but not least, I would like to thank this project for giving me the opportunity to meet the challenge and gain experience. For me, who had never completed a large project using JavaScript before, I did it not only for the marks, but also to increase my knowledge.

Contents

1 Introduction

In the first section, we analyse problems to derive reasons for the need for this investigation. The section detailing research questions and objectives that follows is where we outline our objectives, introduce the design of the system, and provide demonstrations of how the system works to solve problems. In the last part, the thesis structure is presented.

1.1 Problem Statement

With the explosive growth of electronic devices, not just in terms of numbers, but also in terms of diversity of types, the way we interactive with the world changes. The proliferation of the Internet of Things have greatly fostered the need to implement a system that can manage all these smart things in a single place. IoT devices interactions are already an inherent part of our everyday lives, however, it is difficult for an application to cater for everyone's different needs. For example, certain functions are redundant for specific users and can cause a negative operating experience for them. The demand for interface design goes beyond the typical graphical user interface for computer screens; new devices on the market, such as smartphones, tablets and smartwatches, also require different UI designs. Apart from this, each individual has a distinct aesthetic as well as various operational habits, such as left- and right-handedness. It has become an increasing need for many people to

create personalised user interfaces and applications themselves so that the interface can best fit the users' needs.

eSPACE [18], an end-user authoring tool prototype, provides such an environment that allows users to manage their smart things with a graphical user interface of their own design. This gives rise to another requirement: for users who are not professional designers, having guidance when designing the interface is important. Designing UIs is a complicated matter and it can be a challenging task. A successful layout must satisfy various objectives, not only aesthetic requirements but also the rationality of the design for practical use. First, for novice designers and end users who are without programming knowledge, failure to follow design guidelines can result in bad ergonomics designs. Secondly, for skilled designers, the GUI design process often involves a significant amount of repetitive, time consuming subtle tasks such as resizing, aligning, and reorganising. Overall, it is essential to provide guidance and functionalities that can facilitate design process.

1.2 Research Question and Objective

The purpose of this thesis is to facilitate the design process by providing a GUI design environment that includes design suggestions to end users and novice designers. The tool should be able to correct problems in design, as well as refine the design.

Providing inspiration for the design of graphical user interfaces (GUIs) has received much attention in recent years. There are a number of studies that look at the recommendation systems for assisting designers during the UI design process. O'Donovan et al. [15] present DesignScape, a system which aids the design process by making interactive layout suggestions. Some existing tools were developed for the purpose of providing template-based interfaces, such as Drenttel's invention[4]. While some other tools worked on grid-lines. Frisch et al. [5] contribute two tools which support layout tasks on interactive displays: interactive grids and multi-touch alignment guides. Dayama et al. [3] propose a novel optimisation approach for the generation of diverse grid-based layouts. Other influential work includes Choi et. al. [2] and Li[11]. These related works will be discussed in more details in Chapter 2.

Combining the demands derived from the requirements analysis and the experience from related work, we propose different kinds of recommendations to the user in terms of relative position refinement and style exploration, which are shown in the UI design view of the eSPACE authoring tool.

As discussed in the previous section, end users may encounter difficulties while design their application interfaces. With the guidance provided by

eSPACE, they are able to generate the user interfaces more easily. For users who know exactly how they would like their interface designed, they can use the functions offered by the tool to help them place the elements, for instance, address the overlap issue, align selected elements, spread the elements at equal intervals. For end users and novice designers, they are also able to efficiently complete designs even if they do not have a clear design concept by simply dragging and dropping the elements and applying the suggested recommendations.

1.3 Thesis Structure

This thesis will be divided into six chapters. To begin with, we will study related research areas. Here we discuss the strengths and weaknesses of these studies. We also introduce important design guidelines in Chapter 2 to point out constraints in the design process.

In Chapter 3, we briefly introduce the authoring tool we extended, eSPACE, and the solution of our strategy to streamline the UI design process.

Then it is followed by the implementation details in Chapter 4, where illustrations, algorithms and schematic diagrams are given to explain how the functionality is implemented.

Chapter 5 depicts the design environment. in this chapter, we will simulate a real user scenario and provide before and after screenshots of using the corresponding functions.

Chapter 6 draws the conclusion of this thesis and proposes methods to optimise algorithms with some directions for future improvements.

2
Related Work

In our study we are interested in two areas of research: the recommendation system and the design guidelines. In this chapter, we therefore first introduce some recommendation systems and then present some design guidelines.

2.1 Recommendation System

To date, several studies have been conducted on simplifying the process of user interface design. Users frequently choose to draw sketches in the early stages of UI design, but these low-fidelity sketches can lead to a variety of issues, including poor visual appearance, ambiguity in the user's input. Some tools, for example *Sketchplore* [22], support inferring the user's sketch into a more aesthetic layout. Xu et al. [25] propose a user interface for visualising and editing two types of spatial relationships: edge alignment and equal spacing, and editing inferred relationships between elements detected in sketches. Solving the problem of overlapping, *RUITE* [16], a UI layout refinement engine, advances existing research by converting low-fidelity sketches to higher fidelity with beautified UI layouts.

Sketchplore aims for real-time optimisation while also allowing for natural sketching flow. It provides a local optimiser for small changes and a global optimiser for large changes. The sketching tool has several contributions: First, Extend Model-based interface optimisation to interactive layouts that

can edit element sizes, positions, and colours. Second, Propose a method of dynamic optimisation that infers the user's task according to the current layout. While this paper covers a variety of design topics, it leaves out semantics and the dynamic nature of interactions that interfaces enable. Although the optimiser performs admirably in most cases, it is limited when there are more than 10 elements since it is heavily based on well-validate models rather than metrics.

Xu et al. [25] demonstrate effective layout beautification techniques that emphasise inferring relationships between layout elements. They also find a better alternative for alignment and equal spacing. They can be integrated with existing graphic editors. To edit geometric constraints, the tool requires explicit mode switching to activate or deactivate the mode. In the future, the mode-free interface can be explored to ensure a more pleasant experience.

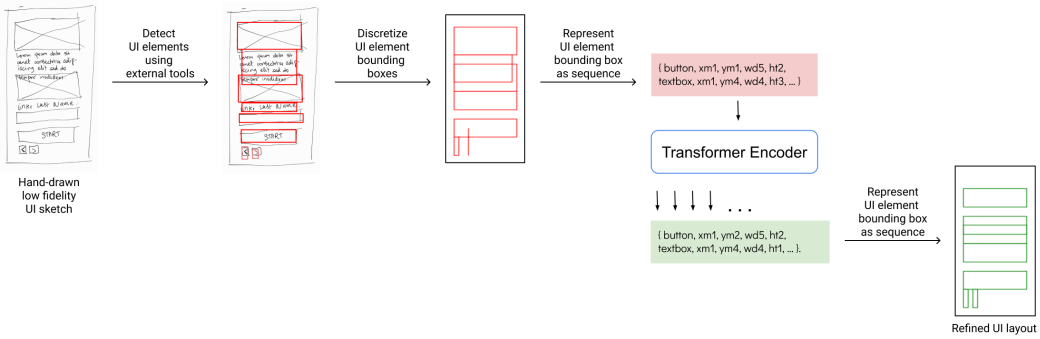

Figure 2.1: RUITE

RUITE[16], a UI layout refinement engine, represents UI element bounding box as a sequence with the transformer encoder to align an unaligned or misaligned UI layout after detecting and discretising UI elements from a hand-drawn low-fidelity UI sketch. The procedure is shown in figure 2.1. The transformer is an attention-based deep neural network model whose representation of the data is in the form of a sequence. A user interface layout, according to the authors, is defined as a list of UI elements with their respective locations and dimensions. Therefore, the power of the Transformer model can be leveraged. To train the model, a vocabulary from all the features appearing in all sequences is created, and a unique index is assigned to each feature. The maximum possible size of such a vocabulary is C + 2(H + W), where C is the total number of categories of UI elements, H and W are the height and width of the grid, respectively. It is guaranteed that RUITE does not displace elements from their original positions to obtain a realistic refined UI layout, but in the overall scheme of things it offers relatively limited functionality.

While a great deal of previous research has focused on how to generate different layout styles, *Rewire* [20] looks into reusing or editing parts of the existing screenshots from images to vector representations. The flat and unstructured screenshots can be modified as vector representations. Aimed at personalising a web page, Layout as a Service (LaaS) [10] contributes a service platform for self-optimising web layout, allowing users to produce usable personalised web layouts. Similarly, *Familiarisation* [21] also introduces methods to reorganising layouts to make elements on a new, unvisited interface more easily found. The main idea of familiarisation is to reconstruct unfamiliar layouts into designs the user recognises. Due to the general foreseeable element positions and layout frameworks, familiarised generates a more effective layout. According to the study, familiarisation can cut visual search duration by 10% and minimise eye-gaze fixations by 23%.

Rewire [20] proposes an interactive system for designers to leverage example screenshots by reusing or redrawing components of the example design. It can generate accurate vector representations of the part of the screenshots in an editable way, allowing for the reconstruction of the designs. Rewire can be seen as an exploration of intelligent design assistance. It was able to infer multiple hierarchies and shape categories, allowing the designer to choose amongst them. The system significantly relieves designers of the slow, tedious and time-consuming reconstruction process. Since it is an attempt of introducing intelligence to screenshots reconstruction, low level techniques developed in the system lead to insufficient accuracy when segmenting example designs. This issue can be improved by applying deep learning techniques or by fine tuning a pre-trained segmentation network. The weakness of Rewire is that the recognition accuracy is insufficient. Another issue is the incorrect filtering of small parts. It means that when elements are small they may be incorrectly identified or filtered out.

In contrast to the "one design fits all" approach, *LaaS* [10] was developed to support *web layouts personalisation* that can considerably improve usability and user experience for individual users without the use of manual coding or templates. The architecture of Laas is described in Figure 2.2. The core components are as follows: layout parser, event logger, design task generator, layout generator and layout adapter. To rearrange online pages into grid patterns, the authors devise a mixed integer linear programming model (MIP). Layout integrity, alignment, and functional layout are the three aspects of MIP's work. *Selection Time* and *Visual Saliency* are two design objectives adopted to efficiently represent any other objectives in the MIP. The time necessary to reach a certain element on the screen is calculated using Fitts' law. The selecting time is affected by the goal distance and size.

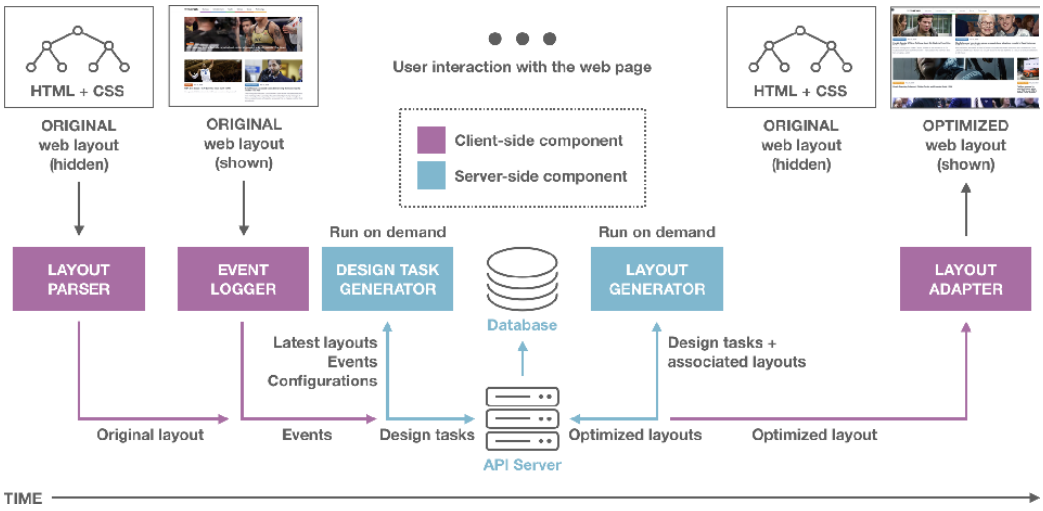

Figure 2.2: Laas architecture, including core components and interactions between them.

The saliency of an item is determined by how noticeable it is in compared to the rest of the page [17]. The saliency of an element is determined by its relative size. *Laas* allows for rapid deployment across multiple pages and machine learning techniques engagement. However, it is flawed in terms of page load times and aesthetics that could be further improved.

Our work is inspired by prior research on interactive layout design. Much of this research focuses on how user interface layouts can be refined or re-targeted to different styles and layouts automatically. Peter et al. [15] present a system that aims at providing interactive layout suggestions when users modify templates or designs for a single-page. The system aids both the refinement process and exploring alternatives in a variety of styles. Based on a mixed-integer linear programming model, GRIDS [3] introduces an approach for generating grid layouts that optimises packing, alignment, grouping, and preferential placement. It is able to complete partial layouts in addition to exploring diverse styles and exploiting local alternatives.

Grids & Guides [5] and Drenttel et al. [4] propose relatively traditional methods for layout generation. Grids & Guides provides interactive grids and multi-touch alignment guides to help with graphical object alignment. However, elements can only be manipulated based on the grid lines or arcs, making the application less useful in practical scenarios.

GRIDS [3] uses a mixed integer linear programming (MILP) paradigm to produce computational layouts based on spatial structures specified by grid lines. To tackle the computational difficulty of creating grid layouts, the

authors establish the following goals: overall alignment, rectangular outline, and placement. Particularly, they take interrelated elements into consideration, that can be manifested in the following ways:

1. If a pair of UI elements are associated, the distance between these elements should be minimised.
2. Interrelated items should be placed contiguously.
3. The specific item should be put on the canvas at specific locations.

GRIDS utilises *core MILP formulation* to guarantee a non-overlapping and non-overflowing grid, while more elaborate layouts features, such as well-aligned, need to be achieved by imposing additional conditions. It introduces two binary variables $\Gamma_{\overline{e}}^{e}$ and $\Pi_{\overline{e}}^{e}$ that can be used to slice the space around element $\overline{e}$ into two half-spaces. $\Gamma_{\overline{e}}^{e}$ indicates e is placed to the upper side of $\overline{e}$. $\Pi_{\overline{e}}^{e}$ indicates e is placed to the left side of $\overline{e}$. The two variables are sufficient to stress overlap issue via constraints. If any of the edges of a pair or more items are aligned with each other, those edges are shared by those elements. The overall alignment is ensured by reducing the total number of grid-lines utilised in the solutions. This means in poorly aligned layouts, there are more grid-lines. GRIDS additionally provides preferred placement for elements, such as a fixed position for a specific element or relative to the canvas. GRIDS builds upon a procedure to guarantee that it is able to systematically generate a wide range of distinct solutions in respect to one another.

According to the evaluation feedback from professional designers, GRIDS might give instant value to novices by enabling grid-based layout generation. The areas for further optimisation can be discussed under three headings, which are: (1) The development of efficient interaction strategies for specifying groups; (2) The adaptation of the MILP model to incorporate Gestalt principles; (3) Detect user's preferred style to obtain more relevant layouts.

In the methods developed by Drenttel et al., the elements are arranged to fill the area provided by the template. For a complicated layout generation, this is a relatively rigid and limited approach.

Prior studies that was presented by Clemens et al. [26] have compared gird-bag layout and constraint-based layout empirically. The comparison results allow for the selection of the most appropriate model in a variety of situations when building the UI generation system.

Some studies [8] contribute the solutions to the keyboard optimisation by developing integer programming as a complementary approach. These studies show that integer programming's non-random search approach improves results and ensures their quality. Because advances in keyboard design can

be applied more broadly to UI design problems, these studies have inspired our solutions for spatial search.

These studies have helped us to understand the direction of development in the relevant fields and have allowed us to arrive at the objectives to be achieved.

2.2 Potential Design Guidelines

In this section we introduce some design guidelines that could be used to generate some UI recommendations.

Gestalt psychology is a school of psychology. There is a large volume of published studies describing the role of Gestalt principle. It was created by three German psychologists in the 1920s. According to Gestalt psychologists, organisms comprehend complete patterns or configurations, not just individual components. Gestalt is defined as a unified structure, configuration, or layout with specific properties that are greater than the simple sum of its individual parts. "*The whole is more than the sum of parts.*" [1] can be used to the summarise the adage.

The following is another explanation of Gestalt Theory by Max Wertheimer: "*The fundamental "formula" of Gestalt theory might be expressed in this way: There are wholes, the behaviour of which is not determined by that of their individual elements, but where the part-processes are themselves determined by the intrinsic nature of the whole. It is the hope of Gestalt theory to determine the nature of such wholes.*" [24]

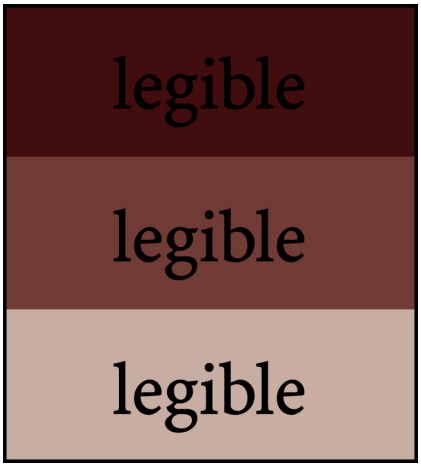

Figure 2.3: Figure/ground contrast affects legibility.

L. Graham et al. [6] propose a study to investigate how Gestalt Theory influences interactive media design. Unexpected interpretations by the user may be introduced if designers disregard Gestalt Theory when implementing UI designs. According to L. Graham et al., this part explains the gestalt laws of perception and discusses how they can be applied to interactive media design.

Figure/ground: The law makes it easier for users to tell apart items from their surroundings, such as images, text, that means images and text must be visible to be understood. Different contrast levels can improve or detract readability (see Figure 2.3). The typical utilisation of this law is to offer feedback by visual changes in colour, due to the fact that the original items must display "ideal" figure/ground gestalt in various situations.

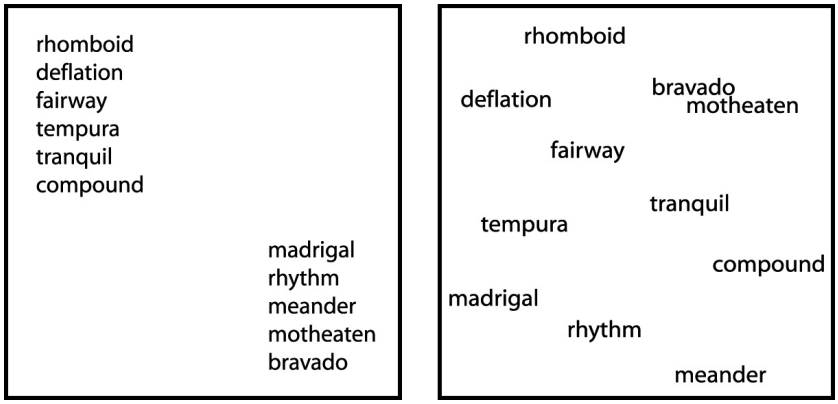

Figure 2.4: The law of proximity dictates that items located close together seem part of a group (left), while items that are not close together are perceived as separate (right).

Proximity: Element that are distant from the others are viewed as separate, while elements that are close to each other are considered to be segregated wholes (see Figure 2.4). For example, a text close to a button might be perceived as a description of the button. Irregular spacing may lead to unexpected interpretations by the user.

Similarity: Items with similar visual features are viewed as belonging to a group if they have the same size, colour, and type, even though they locate far apart (see Figure 2.5).

According to the gestalt law of similarity and proximity, in our tools, the same type of items located near to each other and the items with the same colour tend to be moved, scaled, or aligned together.

To date, several studies have investigated human movement. Fitts' law [12], which is mostly utilised in ergonomics and human-computer interaction (HCI),

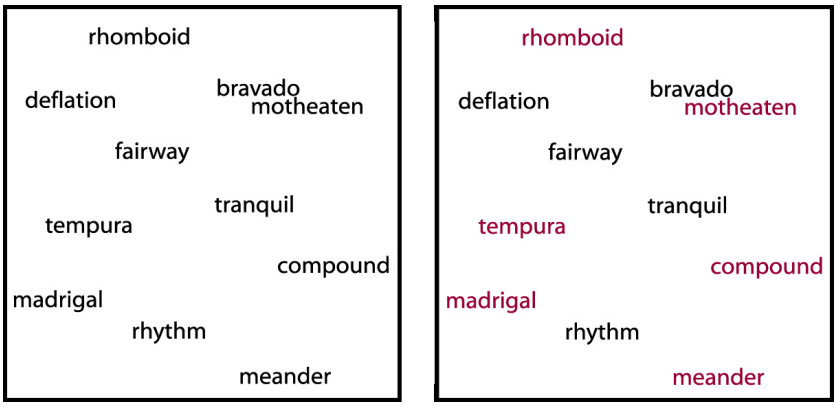

Figure 2.5: The words in the left square are scattered randomly and little perceptual grouping is apparent. Changing some of the words in the right square to red makes them similar, and they seem part of a group.

is a predictive model of human movement that was initially developed by Paul Fitts. Derived from Shannon's Theorem 17, Fitts' law reveals that human movement can be modeled by analogy to the transmission of information. Following the guidance of this law, researchers have been able to predict the time required to position a cursor and hit a target. Besides the cursor, Fitts' law also applies to the case for devices such as the joystick, touch pad, track-ball, helmet-mounted sight, and eye tracker. Understanding this law allows designers to create more effective buttons, forms, lists, and other interactive elements. Fitts' law is one of the most robust and extensively adopted models because it proposes that the difficulty of a task could be measured using the information metric bits.

A motor tasks' index of difficulty (ID) is the quantity of bits transferred during the execution of a movement assignment, and the index of performance (IP) refers to the information capacity of the human motor system, that is, the rate of transmission. IP is computed by dividing ID by the movement time (MT) to complete a motor task. The equation is of the form:

$$IP = ID/MT. \tag{2.1}$$

The following was proposed as the indicator of difficulty for a motor task, loosely based on Shannon's logarithmic expression:

$$ID = log_2(2A/W). \tag{2.2}$$

A denotes the movement distance in this equation. The width of the target measured along the axis of motion is referred to as W. Due to the motion's

final point must be within $\pm W/2$ of the target's centre, W can also be thought of as the allowed error tolerance in the final position. According to this equation, doubling the target distance or halving the size raises task difficulty (ID) by one bit. As a result, ID offers a single but helpful measure of the combined influence of two physical aspects of movement activities.

A useful variation of Equation 2.1 places MT on the left as the dependent variable and ID as the independent variable:

$$MT = ID/IP. \tag{2.3}$$

MT can also be determined by regressing on ID, that means the relationship between MT and ID is linear. In this case, the equation is expressed as follows:

$$MT = a + bID. \tag{2.4}$$

where the intercept (a) and slope (b) are regression coefficients, and they are empirically determined constants. According to the formula above, Fitts' law is expanded as follows:

$$MT = a + blog_2(2A/W). \tag{2.5}$$

The law, however, is inherent to 1-dimensional (1-D) tasks. The task of pointing to 2D targets seems to be more common as the advancement in human-computer interaction technology. Considering the width and height of the target, MacKenzie[13] has made significant contributions to the body of tasks utilising Fitts' Law to evaluate 2-dimensional assignments. The issue with traditional Fitts' law applications to two-dimensional target acquisition tasks is addressed in this study. While the shape of the target is not a circle but a rectangle, the approach angle should be taken into account (Figure 2.6); otherwise, some unexpected interpretations of task difficulty (ID) may emerge.

When $A:W$ ratio falls below 1:2, Fitts formulation yields a negative rating for tasks' index of difficulty, which leads to an obviously theoretical issue. Mackenzie addressed this issue by introducing Shannon formulation by always providing a non-negative ratio for ID.

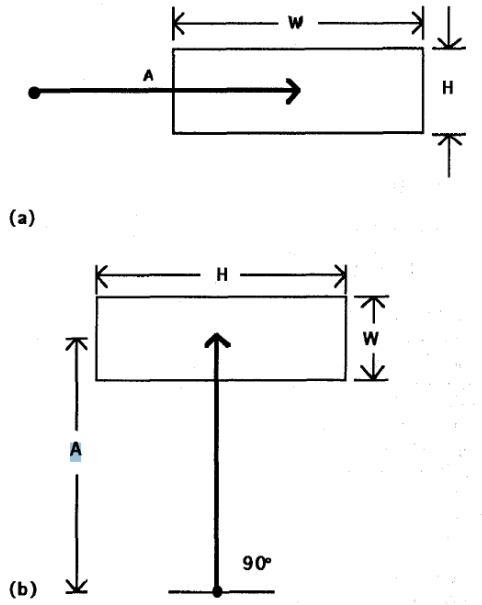

Figure 2.6: Fitts' law in 2D. The roles of width and height reverse as the approach angle changes from 0° to 90°.

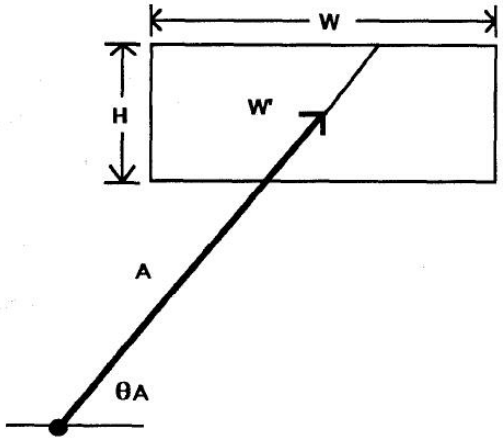

Figure 2.7: The target width is the width of the target along an approach vector.

Mackenzie also introduces two new models to enhance the performance of Fitts' law. The first strategy is the W' model, which replaces W with the target's extent along an approach vector through its centre (Figure 2.7). The second approach, known as the SMALLER-OF model, uses the smaller

of the width or height of the target as W. The previous research developed models that were good at predicting human pointing performance. These studies, however, do not take into account the absolute ambiguity of finger touch.

With the guidance of these design guidelines, we are able to generate recommendations that are in line with design specifications. To illustrate, by following Gestalt Theory, it is possible, firstly, to make the text distinctly different from the background to ensure readability; secondly, to make elements of the same type or group have the same characteristics, such as the same colour, size and shape; and thirdly, to such elements together to form a visual part of a group. By following Fitts' law, we are able to calculate the proper size of elements to avoid unreasonably sized elements.

2.3 Summary

In this section, the studies in the related field have been introduced. This investigation of related studies enables us to comprehend what the existing tools achieve and what needs to be improved. By examining other research in the field of UI design and combining it with research on the generation of graphical design recommendation systems, we derive the objectives that our tool should achieve. We use a constrained optimisation strategy, which is inspired by these layout beautification techniques, to ensure alignment and even distribution.

3 Design

This thesis works on the extension of eSPACE [18], an end-user authoring tool prototype, which will be introduced in this chapter. After introducing the different *views* provided by eSPACE, we continue to discuss the design of our solution to facilitate the UI design process for end users by adding recommendations to the UI design.

3.1 eSPACE Authoring Tool

With the continuous development of technology, more and more smart devices are being introduced into people's lives. However, to manage these devices, users normally use the decentralised controllers like a variety of applications. The lack of a user-friendly control of such smart devices and the Internet of Things (IoT) is a factor that greatly affects the use of smart things. A tool is needed to adapt to evolving user demands.

Aimed at enhancing user experience with their smart devices and IoT appliances, eSPACE enables users to manage all of their smart things in a single application rather than having this control dispersed over many locations. eSPACE (end-user Smart PIACE) authoring tool is implemented for the design and development of applications for managing cross-device and IoT appliances. End users can create their own Cross-Device (XD) and IoT management application using the visual tools provided by eSPACE.

The authoring tool is structured into four views: the *home, UI design, interaction* and *rules view*. An overview of all the user-defined applications, rules, devices and other users are presented on the *home view* (Figure 3.1). The *home view* can be regarded as a dashboard-like overview where users are allowed to group their applications. The *rules view* is utilised to define interaction rules. By clicking **Add Rule...** button, the *rules view* is opened for users to create rules. To edit existing rules, users can select the **Edit** button followed by the rule and enter the *rules view* (Figure 3.2).

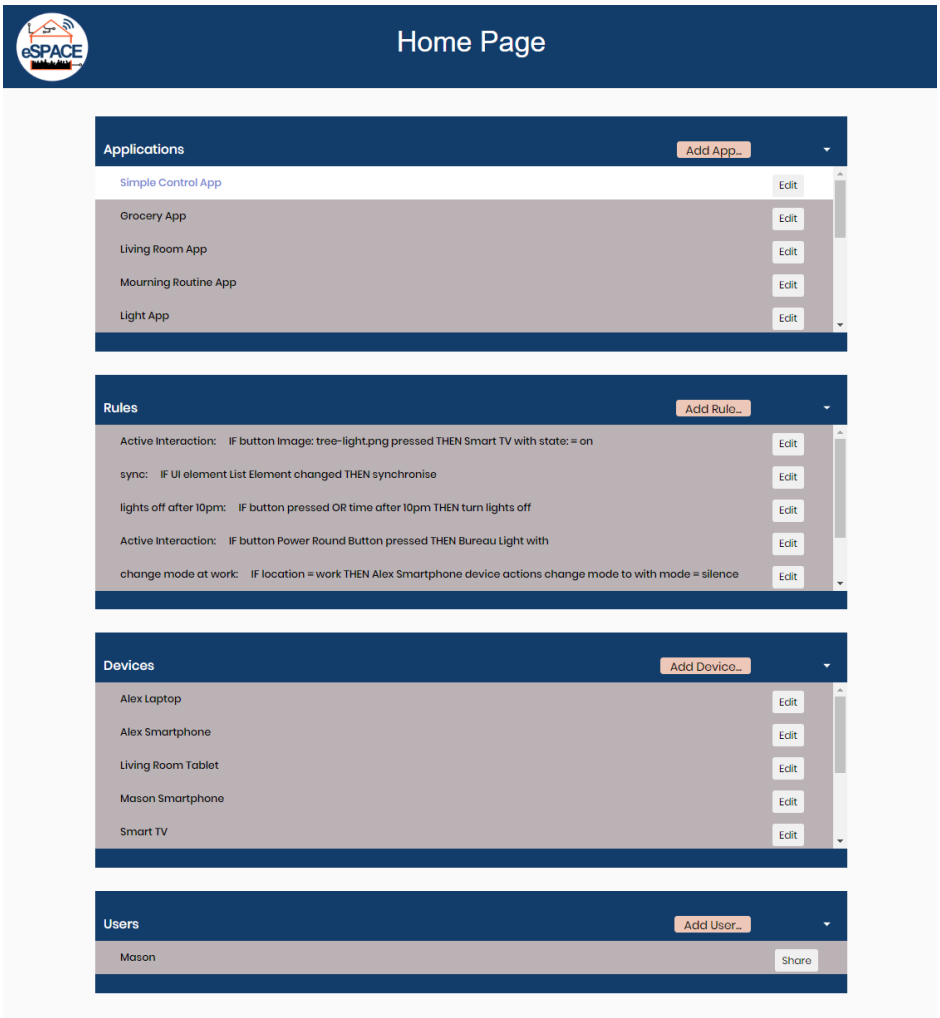

Figure 3.1: *Home view with user cursor hovering over the Simple Control App*

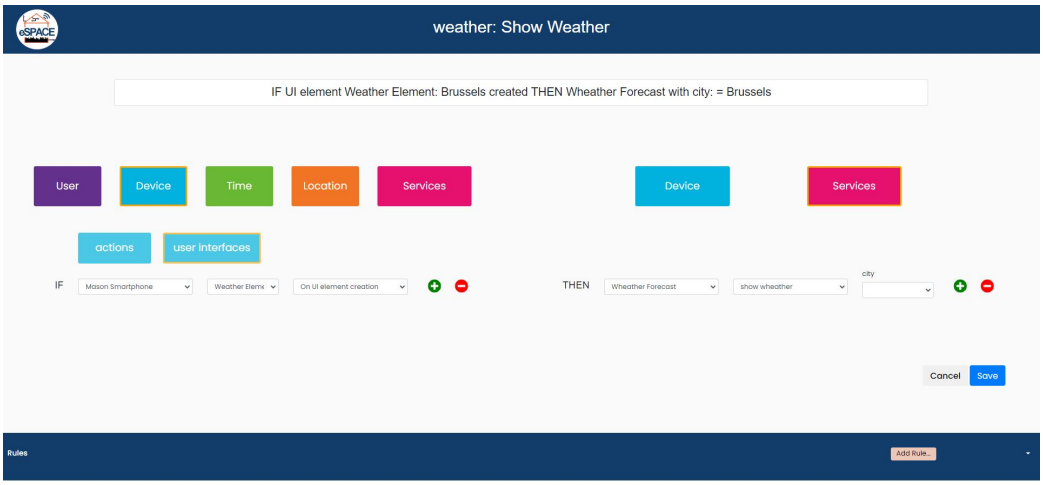

Figure 3.2: *Rules View*

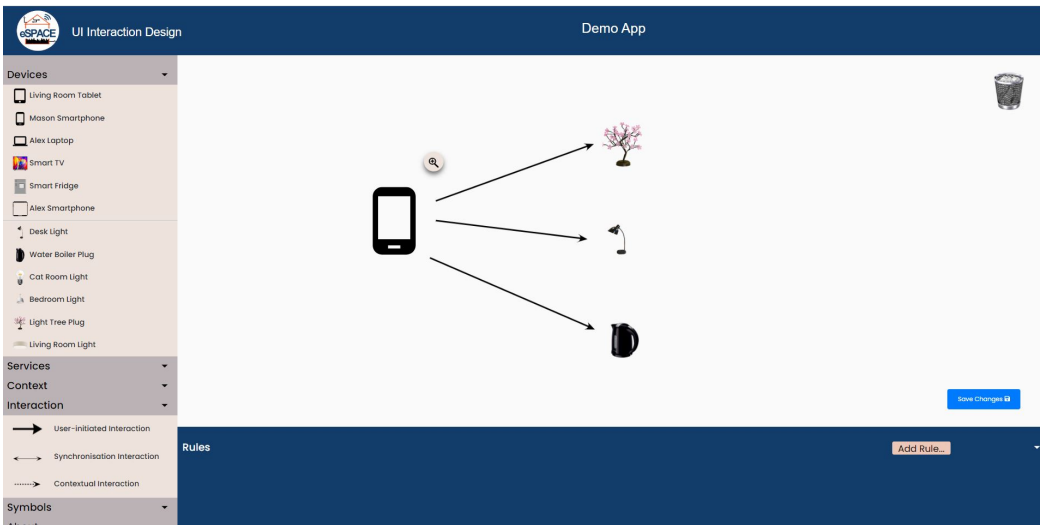

Figure 3.3: *Interactions View*

Next, end users can visualise interactions across devices with the *interaction view*. By selecting its **Edit** button in the **Applications** drop down, any currently existing application can be edited. The associated application's *interaction view* will then be displayed. Figure 3.3 depicts an example interaction, which could be interpreted as "when home, synchronise my pictures between my phone and my computer". By processing the magnifying glass shown above a device in the interaction view that someone can go to *UI design view*. Our work focuses on extending this UI design view. Before

the extension work started, the original view is shown in Figure 3.4, and the modified view will also be shown in Chapter 5. In *UI Design View*, users are able to create personalised GUI for their applications by dragging and dropping elements from the UI Elements list. Figure 3.5 provides a simple example of the application users may create using the eSPACE authoring tool.

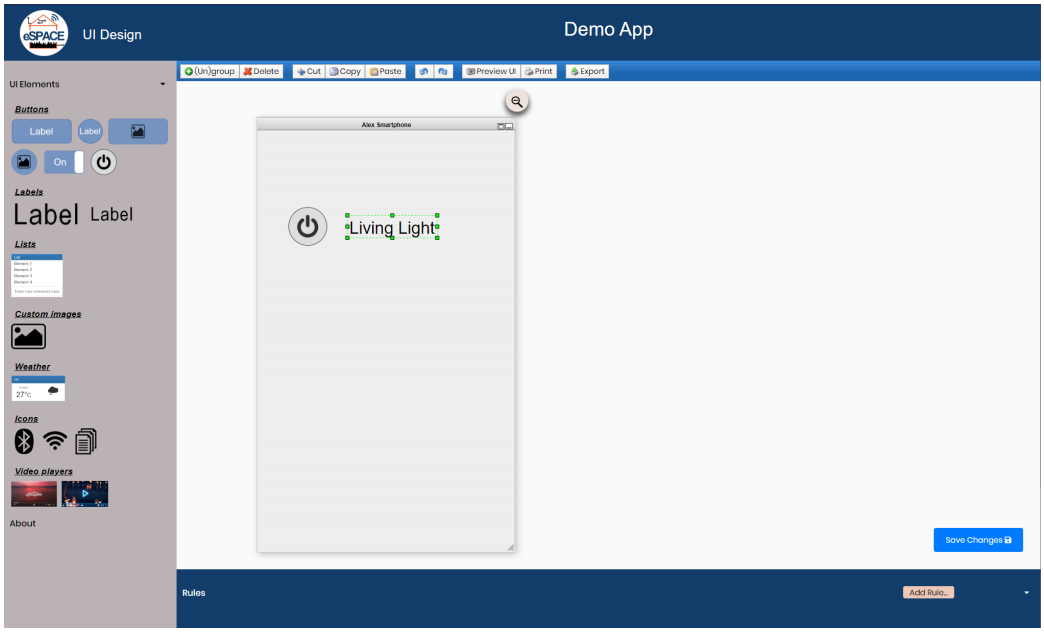

Figure 3.4: *UI Design View*

The technologies utilised to implement the eSPACE authoring tool are given below:

- web technologies: JavaScript, HTML5, CSS3 and some additional JavaScript libraries (e.g. canvasutilities[1]);
- the technologies for managing the page layout and structure: Bootstrap[2] and CSS Grid Layout[3];
- the technologies for retrieving data from the server: JQuery API[4]

[1] `http://dbp-consulting.com/scripts/canvasutilities.js`
[2] `https://getbootstrap.com`
[3] `https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout`
[4] `https://api.jquery.com`

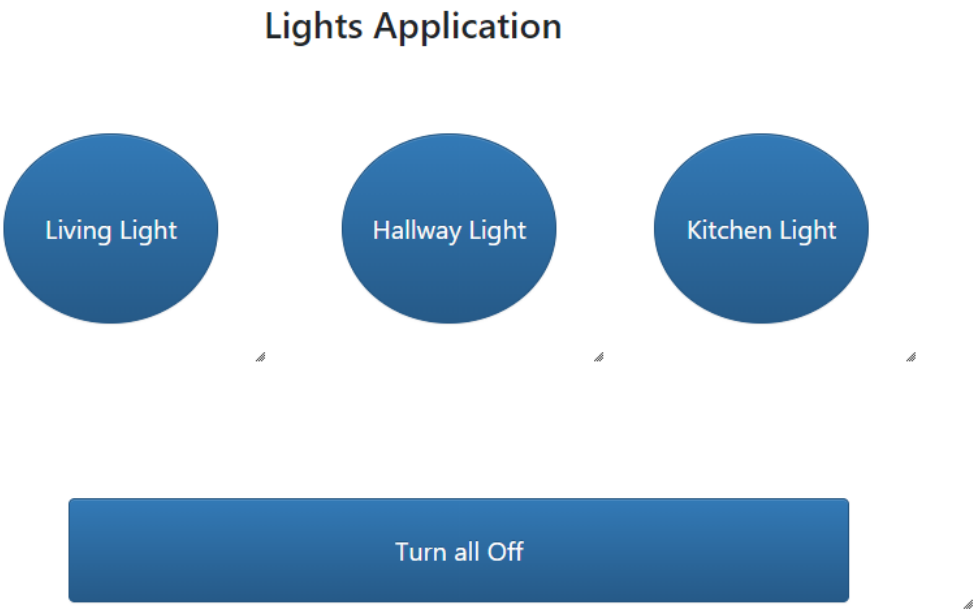

Figure 3.5: Simple lights application

- the JavaScript library used for UI design view: the mxGraph diagramming JavaScript library[5] which uses SVG and HTML for rendering).

3.2 Design of Solution

Using eSPACE, users are allowed to design graphical user interfaces for their applications in the UI design view. As few studies have focused on recommended solutions for user GUI design, the main purpose of this thesis is to provide suggestions while end users designing user interfaces.

As a tool that needs to be geared towards non-specialist designers, it is essential to ensure the aesthetically pleasing and clarity of GUI design. The interface should be visually clear and guarantee there is no occlusion of UI elements.

Suggestions for the user interface can be divided into two types: First, recommendations with higher priority: for design issues that would affect the operation, such as overlapping issue and weak readability due to low contrast, will be fixed if detected. Second, optional recommendations: for design issues that would not affect the operation, such as elements not aligned or cluttered layout, it is up to the user to decide whether to accept suggestions to fix them.

[5] `https://jgraph.github.io/mxgraph`

Combining our knowledge of related work and the requirements derived from our use case scenario, we arrive at several functional requirements and present them in the following sections.

3.2.1 Fix the Problem of Overlapping

There should be a recommendation about avoiding overlapping elements. When two or more UI elements have overlapping issues, that is, when some elements are in front of other elements, the user should get a recommendation that solves this overlap problem. The recommendation should guarantee that overlaps are resolved with minimal changes in the position of elements. Since the overlapping elements may lead to side impacts on the operation, we should check for the occurrence of overlapping issues after each adoption of any other recommendations. If overlaps between elements are detected, the corresponding procedures that are based on the recommendations adopted in this step should be adopted to solve these overlapping issues.

3.2.2 Equally Spacing Elements

Recommendations should include the equal spacing of certain UI elements. When more than two elements are selected, the recommendation should provide a solution that the elements be distributed equidistantly. Whether they are divided vertically or horizontally at equal intervals depends on the comparison of their rightmost edge minus their leftmost edge and their topmost edge minus their bottom-most edge. The detailed algorithm will be explained in Chapter 4.

3.2.3 Align Elements

The third functional requirement is that recommendations should provide alignment of selected elements. Depending on the distribution of the selected elements on the interface, these elements should be left-aligned or top-aligned. The relative order of positions between these elements may also be changed in this step if they are not fixed. To be specific, if the relative position of elements is flexible, the relative positions are arranged in the following pattern: top-align the elements and put the longer height on the left; left-align the elements and put the longer width on top.

3.2.4 Fix Wrong Colour Combination

In addition to the functional requirements for the position of the elements, we also set out the functional requirements for ensuring colour harmony. Wrong colour combinations are defined as background colours and element/-text colours being too close together or some elements of the design having too high a colour contrast. The former results in low readability, while eye strain is caused by the latter, which makes engagement much more difficult. To fulfil this functional requirement, we need to perform a reading of the element colours, transform them into a colour form that can be further analysed, and perform an analysis of the colour combinations.

3.2.5 Explorations of Different Styles of Layout

The recommendations that provide explorations of various styles of UI layout have been introduced to assist with UI design. Design tools supporting automated layout adaptation are very promising for the design process. Generally, it is the case that people's creativity is constrained, and they become stuck with their existing ideas and are unable to consider other layouts. To get around this, these recommendations are employed to generate a variety of styles based on the original layout. In these recommendations, the elements should be properly aligned, equally spaced, and should not overlap.

The detailed implementations and algorithms of these recommendations are presented in Chapter 4.

4 Implementation

In the previous chapter we identified the functional requirements for our solution. In this chapter we will discuss the implementation of the recommendations we added to the eSPACE authoring tool. First we explain the coordinate system that is used for all recommendations and the algorithms this system are based on. Then we explained the following implementation of functionalities in details: fix the issue of overlapping, space elements evenly, align selected cells. In this section, the process of thumbnail generation and switching between solutions is also included. After that we present the exploration recommendation.

4.1 Technology and Coordinate System

The system is implemented using JavaScript as programming language and Spring Boot as framework. The JavaScript language version is ECMAScript 6. The tool has been implemented using the *mxGraph* JavaScript library[1]. HTML5, CSS3 and Bootstrap are used to manage the page layout and structure. As mentioned in Chapter 3, our solution extends the eSPACE authoring tool and provide recommendations during the design process when end users create a new UI.

[1] `https://jgraph.github.io/mxgraph`

We use the same coordinate system as the HTML canvas coordinate system which defines the origin point at the top left corner of the canvas. The X axis is positive rightwards and the Y axis is positive downwards. The *scalable* property of the canvas is set to *unscrollable*.

As shown in Figure 4.1 Each element in the canvas possesses the following geometry information: x_e, y_e, $Width_e$, $Height_e$. x_e, y_e represent the location of left and top edges of individual element e respectively. $Width_e$, $Height_e$ represent the actual width and height. The non-scrollable canvas restricts the geometry information x and y can not have negative values.

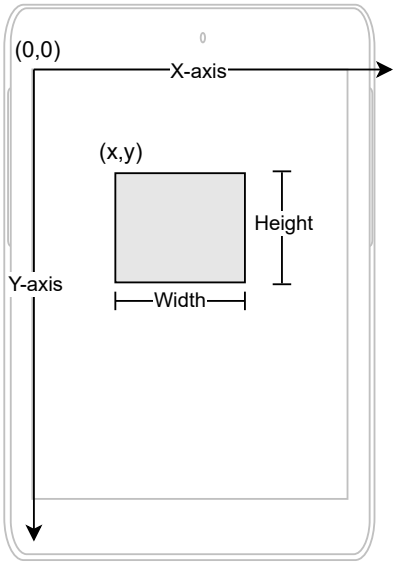

Figure 4.1: The X and Y axis of a canvas and the geometry information of an element.

The following part of this thesis moves on to describe in greater detail the specific implementation algorithms used for each type of recommendation. The implementation is divided into two main sub-groups: refinement and exploration of the possible designs.

4.2 Refinement

This section focus on the local layout optimisation. Rather than focusing on the overall layout of the canvas, it concerns the way to implement different functionalities on elements themselves. For example, if end users want to fix

a position error between several elements, the functionalities introduced in this section will help without major changes to the layout.

In the following sections, the detailed implementation are given. The section starts with the most important function that fixes the issue of overlapping user interface elements. It follows by the function of spacing selected elements evenly. The third one describes the function for aligning elements.

4.2.1 Fix Overlapping Issues

In graphic design, the use of overlapping can create depth and dominance. For instance, text overlaps with graphics, images overlap with backgrounds and graphics overlap each other to create a complex graphic. However, when it comes to graphical user interfaces, overlap is a constant source of issues. It is never a wise move to have elements overlapping in an application because overlapping often leads to occlusion of certain UI elements, mistakenly touching and reduced usability of the software [14, 23].

While designing GUIs, end users may sometimes accidentally overlap elements without realising it. Especially when there are a plenty of elements located on the canvas, placing a new element without overlapping it with others can be a tough task. Owing to this, a function that may assist in resolving this problem is crucial for the system.

The algorithm to fix the problem of overlapping element positions is given in Algorithm 1. The algorithm was inspired by a *stackoverflow* article[2].

The variable *hasIntersection* controls the *while* loop. When this value is true, it means that there are overlapping problems in the canvas. The function *findIntersection(rect,rectList)* returns a list of rectangles if any other elements intersects with the element *rect*. Consequently, the loop ends if the length of the result equals to zero, that means *hasIntersection* is judged to be *False*.

The default value for the distance per move *movementFactor* is set to 30 according to the common component sizes. Setting the step size to 30 can prevent the movement from being too considerably large and disrupting the layout overall, as well as too slightly and leading to needless many loops. For the given list of elements *rectList*, the variable *surroundingRect* records the leftmost point as x, the topmost point as y, the total width as *width*, and the total height as *height*.

In order to avoid infinite loops in the while loop function, the variable *maxIteration* is introduced. With each loop, the value of *maxIteration* re-

[2]`https://stackoverflow.com/questions/3265986/an-algorithm-to-space-out-overlapping-rectangles`

Algorithm 1 Fix the problem of overlapping

```
hasIntersection ← True
movementFactor ← 30
surroundingRect ← surroundingRect(rectList)
while hasIntersections do
  for rect in rectList do
    moveX ← 0
    moveY ← 0
    intersectRects ← findIntersection(rect, rectList)
    if intersectRects.length > 0 then
      for rPirme in intersectRects do
        xTrans ← (rect.centerX − rPrime.centerX)
        yTrans ← (rect.centerY − rPrime.centerY)
        if xTrans < 0 then
          moveX -= movementFactor
        else
          moveX += movementFactor
        end if
        if yTrans < 0 then
          moveY -= movementFactor
        else
          moveY += movementFactor
        end if
      end for
      xTrans1 ← (rPrime.centerX − surroundingRect.centerX)
      yTrans1 ← (rPrime.centerY − surroundingRect.centerY)
      if xTrans1 < 0 then
        moveX -= movementFactor
      else
        moveX += movementFactor
      end if
      if yTrans1 < 0 then
        moveY -= movementFactor
      else
        moveY += movementFactor
      end if
      rect.x+ = moveX
      rect.y+ = moveY
    end if
    hasIntersections ← (findIntersection(rect, rectList).length ≠ 0)
  end for
end while
```

duces by 1. The loop breaks when the value reaches 0. The occurrence of break means the function was in infinite loops due to exceptions, and end users need to move overlapping elements manually under such situation.

To begin with, the following operations need to be performed for each of the elements, noted as *rect* in the below:

Initialise two variables *moveX* and *moveY*, which represent the x-axis shift and the y-axis shift, to 0. If the length of the result of *findIntersection* is not 0, store the intersected list of elements in *intersectRects*. Iterate over the elements the list *intersectRects* one by one, noted as *rPrime* in the iteration.

Algorithm 2 Equally space elements in the horizontal direction

```
surroundingRect ← surroundingRect(rectList)
widthSum ← 0
space ← default
widthNow ← 0
for rect in rectList do
    widthSum = widthSum + rect.width + space
end for
widthSum− = space
rectList ← orderByX(rectList)
if surroundingRect.width < widthSum then
    for rect in rectList do
        rect.x+ = widthNow
        widthNow+ = rect.width + space
    end for
else
    space ← (surroundingRect.width − widthSum)/(rectList.length − 1)
    for rect in rectList do
        rect.x+ = widthNow
        widthNow+ = rect.width + space
    end for
end if
```

The second place is to decide the direction the element should move. A decrease in *moveX* and *moveY* means a corresponding shift in the negative direction of the directional axis, and the increase means a shift in the positive direction. To be clear, the reduction in *moveX* stands for the move left. The increase in *moveX* stands for the move right. The reduction in *moveY* stands for the move up. The increase in *moveY* stands for the move down. Back to the implementation, if *rPrime* locates at the left of *rect*, the x-axis movement *moveX* is reduced by the *movementFactor*, otherwise, increased by

movementFactor. The y-axis movement *moveY* is calculated in the same way. If the centroid of *rPrime* is on the top side of *rect*'s centroid, *moveY* reduces the *movementFactor*, that is, *rect* moves upwards by *movementFactor*.

After iterate intersected elements, add a vector to *moveX* and *moveY* proportional to the vector between the centre of *surroundingRect* and the centre of *rect*. Compare the central point of *rect* with the centroid of surrounding rectangle *surroudingRect*. Similar to previous steps, *rect* moves toward left if *rect* is on the left half of the *surroundingRect*, and moves upwards if *rect* is on the top half of the *surroundingRect*. Repeat all these steps until nothing overlaps. An example of the process is shown in Figure 4.2.

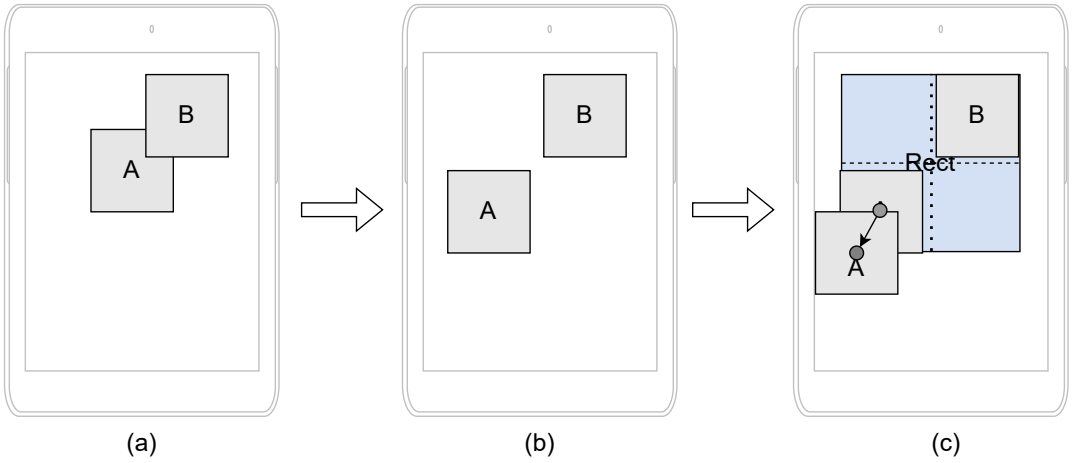

Figure 4.2: The process of fixing the problem of overlapping: (a) the original layout. Element A and B overlap. Adjust the position of A to solve the problem; (b) the centroid of A is on the lower left side of B. Hold B still and move A down to the left; (c) compare the centroid of A and the centroid of *surroundingRect*(light blue rectangle). Move A down to the left since its centroid is to the lower left of the centroid of *surroundingRect*.

4.2.2 Equally Spacing Elements

One of the most fundamental yet crucial components of any successful design is appropriate spacing of the user interface elements. It aids in information organisation and establishes the design's rhythm, structure, and hierarchy. Space in graphic design refers to the distance between various compositional design elements. Space also known as white or negative space. In graphic design, space is utilised to divide or link elements in the design layout, that means, space is utilised to help users identify groups of elements and make

a logical order. Adding large space emphasises the differences between the various visual parts of the composition, while adding more narrower space between them produces more relationships between the items. As we mentioned in Chapter 2, in the Gestalt Theory, the law of proximity states that elements located close together are more likely to be regarded as a part of a group, while dispersed elements are more likely to be perceived as separate.

The recommendation system focuses on equidistant spacing distribution since it is an error-free type of element spacing layout in design principle and it is less likely to visually mislead users [7]. Notably, consistent padding also improve readability and make the design more aesthetically pleasant. The system provides tools to space evenly between objects horizontally or vertically on canvas. The algorithm for achieving equal distances in the horizontal direction is shown in Algorithm 2.

In order to make the algorithm clear, the definition of the variables explained below: The parameter *rectList* is a list of selected elements to be evenly spaced. The *surroundingRect* , as mentioned in the previous section 4.2.1, stands for the smallest rectangle that can enclose all elements. The *widthSum* is the minimum distance required that all elements can be placed properly. The *space* is the minimum space between two elements. The *widthNow* records the x-axis coordinates where the next element should be placed.

To obtain an equidistant distribution of elements, as illustrated in the horizontal direction, the very first stage is to calculate the geometry information (in this case, the *width*) of the surrounding rectangle. The *widthSum* is calculated by taking the sum of the width of selected cells and the *(n-1)* times space, where n is noted as the length of selected cells. In order to keep the relative order between several elements unchanged, the elements in the *rectList* are sorted from left to right according to the x-coordinate using the bubble sort algorithm. Even if the element overlap problem has been solved, the narrow space may lead to inconvenient handling, such as accidental touching. The process of reposition the elements is shown in Figure 4.3. To determine whether there is adequate room, it is necessary to compare the *width* of surrounding rectangle and the *widthSum*. The $surroundingRect.width < widthSum$ stands for that there may be overlapping elements or elements that are too close to each other. Based on the results of the comparison, relocate elements sequentially according to the strategy below:

$$x_n = x_{n-1} + width_{n-1} + space \tag{4.1}$$

If $surroundingRect.width < widthSum$ happens, we adopt the default space as *space*, otherwise, we adopt the calculated space as space.

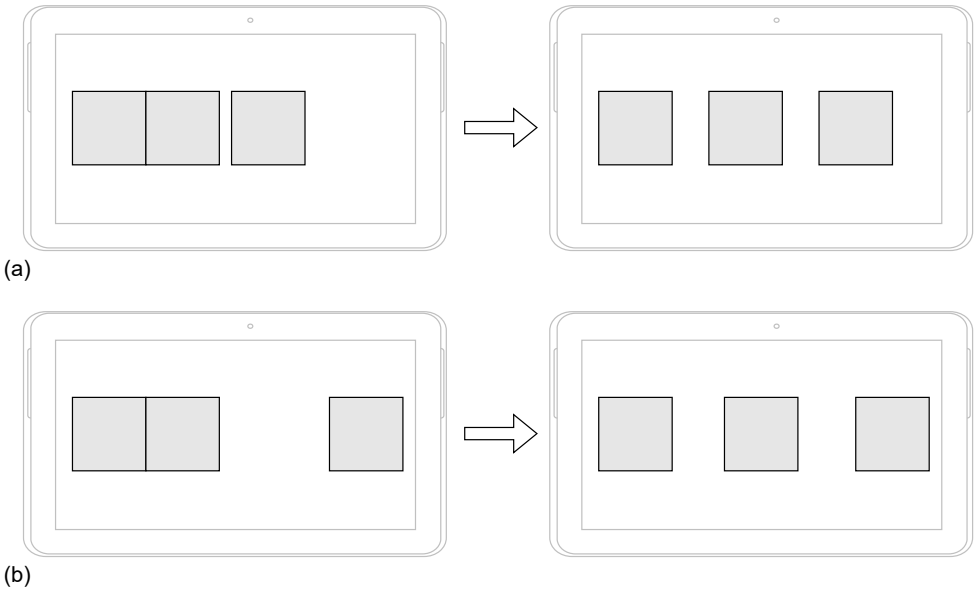

Figure 4.3: (a)no adequate room, introduce the default space(set here to 30) as the minimum space between elements; (b)when the space is large enough, the elements are distributed equally.

With the help of spacing elements equally, users are easy to generate an aesthetic and user-friendly graphic user interface.

4.2.3 Align Elements

In graphic design, alignment is crucial because of the ability to balance the elements so that they are aesthetically pleasing, enables designers to organize things in a way that matches how people naturally scan the page and establishes a visual link between similar elements. As illustrated by the Gestalt theory of similarity, since the aligned elements share edges, they can be considered to have the same properties and are therefore considered to be in the same group. It is never a smart decision to arrange things randomly while working with graphic design. All of the components can be joined and brought together into a solid, cohesive framework by applying the alignment principles. When the rules of alignment are not properly followed, graphic user interfaces appear disorganized, random, and visually irrational.

Edge alignment and center alignment are the two fundamental alignment principles. Edge alignment, a widely used technique, places pieces against a boundary that naturally corresponds to their outside edges. It produces a very comfortable, pleasant, safe, and traditional design. This system uses

edge alignment to maintain the items flush against the left or top boundary.

Despite understanding the significance of alignment, the user still finds it challenging to properly align all of the objects because of the numerous edges that exist in the canvas. Consequently, it is crucial to offer consumers wise alignment recommendations. Faced with different scenarios, two distinct functions have been developed to specify the alignment, which is explained in Algorithm 3. In doing so, *mxGraph.alignCells*, a function provided by *mxGraph* library, is fired to align the given cells with the parameter *mxConstants.ALIGN_LEFT* or *mxConstants.ALIGN_TOP* passed to it according to the given alignment.

The first case applies when there are only two elements, and their relative positions are unrestricted. To formulate appropriate visual groupings, The function both specifies the alignment and their relative positions for the list elements. The alignment is decided according to the shape and size of the two elements. Following the rules provided by Kim and Foley [9], who propose the rules providing professional support in user interface presentation design, the decision tree for shape and size comparison for the two rectangular items is built and shown in Figure 4.4.

In the decision tree, We first need to decide whether the width and height of the elements are similar. The similarity ratio *gamma* is introduced to compares the similarity of the widths and heights of the elements in the upper layer of the decision tree. If $width_2 > width_1$ and $width_1 > \gamma * width_2$, or vice versa, if $width_1 > width_2$ and $width_2 > \gamma * width_1$, the widths of cells *1* and *2* are regarded as being similar, and the same is true for judging their heights. In this tool, the default γ is set as 0.618, which is know as the golden proportion in the field of aesthetics [7, 19]. Designers are able to adjust this parameters according to the actual situation to gain an appropriate effect. In the lower layer, the decision tree decides the alignment and placement of the two chosen cells with the maximum widths and heights as the criterion. After the cells are aligned, two different *equal spacing* function fires. If the alignment is *mxConstants.ALIGN_LEFT*, distribute the chosen elements evenly and vertically,otherwise, specify equal horizontal spacing for elements. The second case applies when there are only two elements whereas the relative position is fixed or when more than three elements are selected. In this situation, items selected is regarded as a huge rectangular, whose geometry information *x, y, width, height* are recalculated as following:

x The leftmost point of the selected items

y The uppermost point of the selected items

Width The rightmost point of the selected element minus the leftmost point of the selected element

Height The uppermost point of the selected element minus the bottom-most point of the selected element

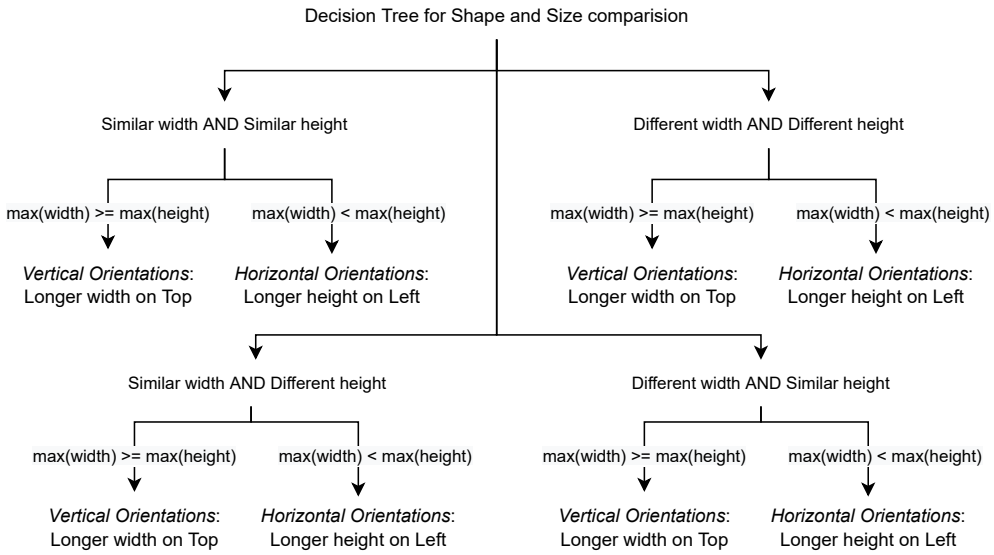

Figure 4.4: The layout rule decision tree for analyzing two rectangular items based on the comparison of shape and size.

After the rectangle is generated by *surroundingRect()*, compare the length of *width* and *height* to decide the alignment for the selected elements. If $width > height$, assign alignment as $mxConstants.ALIGN\_TOP$, and call the horizontal even spacing function. Align the items to the left and space them vertically and equally while $height > width$. An example is given in Figure 5.3.

The technical issue arises after the selected items are aligned that the aligned cells may intersect with the other items. If call the fix overlapping function after the align process, some elements that were just aligned may be relocated and lose their proper aligned character. To address this problem, using *findIntersection()* to scan the aligned elements and, if intersections occur, produces a list of intersected element IDs *intersects*. Determine whether the intersection point occurs in the upper or lower half of the new rectangle formed by the aligned cells while the intersected element list is not empty. For the case that the intersection point is positioned in the lower half of the rectangle, move down all the elements whose y-coordinate is lower than that

Algorithm 3 Align Elements

```
if rectList.length < 3 then
    Compare similarity of widths and heights
    if max(width) > max(height) then
        alignment ← ALIGN_LEFT
        Place longer width on top
        Space elements horizontally and equally
    else
        alignment ← ALIGN_TOP
        Place longer height on left
        Space elements vertically and equally
    end if
else
    rect ← surroundingRect(rectList)
    if rect.height > rect.width then
        alignment ← ALIGN_LEFT
        Space elements horizontally and equally
    else
        alignment ← ALIGN_TOP
        Space elements vertically and equally
    end if
end if
graph.alignCells(align, rectList)
```

point, otherwise, move up all elements whose y-coordinate is higher than that point to make over. *offset* is the distance of the intersection point from the top/bottom edge plus the default spacing, which is set as 30 in this thesis. The *alignCells()* has already been explained in the previously mentioned algorithm 3, the algorithm to fix overlap issues is shown in Algorithm 4.

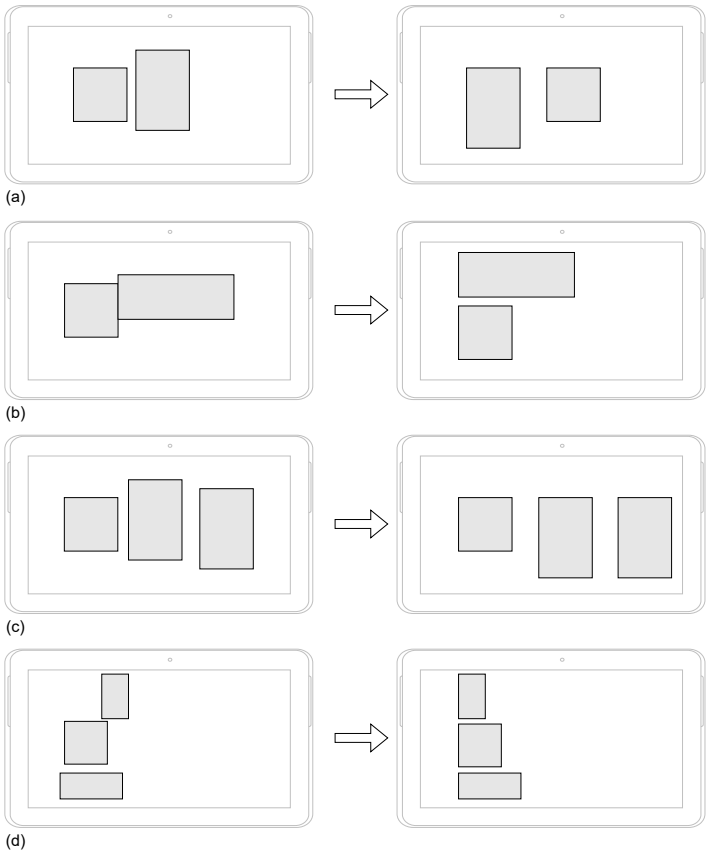

Figure 4.5: The different scenarios of alignment: (a) top align the elements and put longer height on left; (b) left align the elements and put longer width on top; (c)$width > height$, top align the elements and space them evenly and horizontally; (d)$height > width$, left align the elements and space them evenly and vertically.

4.2.4 Thumbnail Generation and Switching Between Solutions

The proposal of this thesis is to provide recommendations for end users, therefore, thumbnails are essential for an intuitive view. The thumbnails

provide more easily viewable recommended solutions and allow end users to decide exactly which solution they want to apply. Clicking on the recommendations fires the event that applies the recommended solution selected to the current canvas. By clicking on the different thumbnails, the user can switch between several solutions on the canvas and decide whether to adopt the solution by observing how it is actually applied on the canvas.

Algorithm 4 Fix overlap after aligning elements

```
alignCells()
for rect in rectList do
  intersects ← findIntersection()
  if intersects.length ≠ 0 then
    checkIntersectPosition()
    offset ← (top/bottom edge − (intersection point.y + spacing))
    for rect in the elements below/upper than the point do
      rect.y+ = offset
    end for
  end if
end for
```

To explain how the thumbnails work, it is first necessary to explain the rationale for generating recommendations. If the layout is modified directly on the original canvas, it will definitely affect the user's operation. Therefore, to generate a recommended solution, the original graph *originGraph* is cloned and saved in a new graph *newGraph*. The modifications are applied on *newGraph*. When users click the recommendation thumbnail, the corresponding graph replaces *originGraph* by the following steps: (1) select all elements in *originGraph*; (2) remove selected elements; (3) clone elements from *newGraph* to *originGraph*; (4) refresh the graph to show updates.

Before all else, create a container *recommendation* and a draw pane *svgPane* in scalable vector graphics (SVG) format. Passing the contents of the canvas to *svgPane*. Empty the contents of *recommendation* then append *svgPane* to it. Set the scale factor to 0.3, that is, the height and width of thumbnails are 0.3 times the height and width of the canvas. Due to the fact that the thumbnails only work on the current graph, the following process need to be done to get thumbnails of recommendations: (1) generate a temporary graph *tmp*; (2) clone current graph to *tmp*; (3) clone *newGraph* to the current graph; (4) take screenshots; (5) clone *tmp* to the current graph. The canvas has actually been changed twice, in spite of that the user will not be aware of these changes since the canvas *graph* is not refreshed.

4.3 Exploration

In addition to the optimisation of selected elements, end users also demand recommended solutions on the overall layout. The recommendation aims to rearrange existing elements to obtain a more diverse range of solutions. The solutions are presented as thumbnails, giving users a visual representation of the results. If it suits their application, users can employ the recommendations, or at least draw inspiration from the different suggestions. In this section, the implementation of recommended solution is explained.

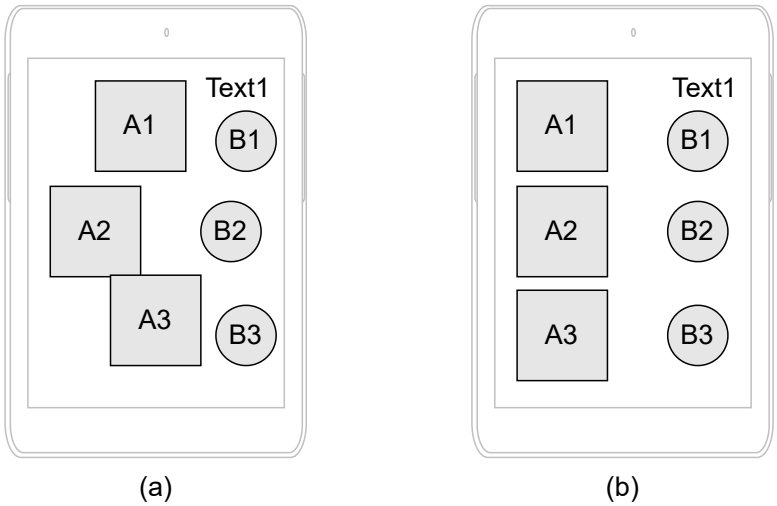

Figure 4.6: An example of the recommended solution: (a) the original layout; (b) the recommended solution.

As the main purpose of the authoring tool is to serve the user in order to better manage smart things, we assume a common application layout where a control button and a label are clustered together, the label is used to identify the device corresponding to the button and the button is used to control the corresponding device. The label can be presented in different forms, such as text box, image. Similarly, the button can be presented as slider, button, switch, etc. Faced with so many combinations, we decided to calculate the type and number of elements and to align the elements with different types and the same number of elements. This way we can get neatly arranged pairs of elements, for example, when there are three labels, three buttons and one text box, they will be arranged as three rows of paired labels and buttons. As shown in Figure 4.6, *A1, A2* and *A3* stands for three elements of the

same type. *B1, B2* and *B3* stands for another three elements of the same type. *Text1* represents a third type of element. The elements *A1, A2, A3* and *B1, B2, B3* are aligned separately while *Text1* remains in the original position.

The generation is activated by *Start Recommend* button, and a cloned *graph* will be edited in the following process. The first step is to address the issue of overlap. Due to the properties of the function *fixOverlapping*, elements are spread out towards fewer elements. To decide how to generate a recommended solution, it is essential to know what elements are present on the canvas, both in terms of type and number. Since the type is stored in the property *style*, which is a *String* class, the type is parsed using regular expressions such as *icon, button, image, etc.* Next, a hashmap is used to store the type and number of elements, where the type is key and the number is value. Iterate through the hash table and if there are other types that have the same value as that type, push all those types into the array. Align them separately, using the type as a classification. Lastly, call the function *equalSpace* to space the elements evenly. Once a suggestion has been generated, a screenshot is taken, which is used as a thumbnail and passed to *Recommendation 1, 2* or *3* for the user to choose from.

With the recommended solutions, we believe that visual suggestions can facilitate the design process along with giving end users who are stuck for ideas some inspiration.

5 Demonstration

In this chapter, we present the result of the work done in the previous chapter by simulating a scenario where users are using the UI design view to design applications for managing their smart devices as well as Internet of Things devices. To demonstrate the effect of each function explained in *Implementation* chapter visually, before and after views of using the corresponding functionality will be shown, respectively.

It starts with an introduction to the UI design view, which is depicted in Figure 5.1. eSPACE provides a drag-and-drop design environment for the design activity. The interface of the UI design view can be divided into four parts: (1) a left sidebar that provides UI elements; (2) a toolbar on top of the design space provides easy copy/paste, undo/redo, deletion, and grouping of UI elements; (3) a *window* element represent the device screen; (4) a right sidebar consists of three recommended solutions and six buttons representing the corresponding functions.

In the UI design view, end users can design their personalised graphical user interfaces by utilising the UI elements provided in the left side bar. The UI elements can be simply dragged and dropped into the canvas. By allowing for precise text, style, and size modification, UI design view can generate elements in a variety of styles. To be noted that although the button *Fix Wrong Colour Combination* is added, we did not implement the functionality to actually fix the wrong colour combination occurring in the layout, instead,

the button now simply return the colour of each elements.

The first button on the right sidebar corresponds to the function *fix the overlap issue*, which is shown in Figure 5.2. The left figure represents the layout before fixing the overlap issue. It can be seen that there are two labels and one icon that are overlapping each other. The right figure stands for the layout after fixing the overlap issue, where the intersecting elements are spread out.

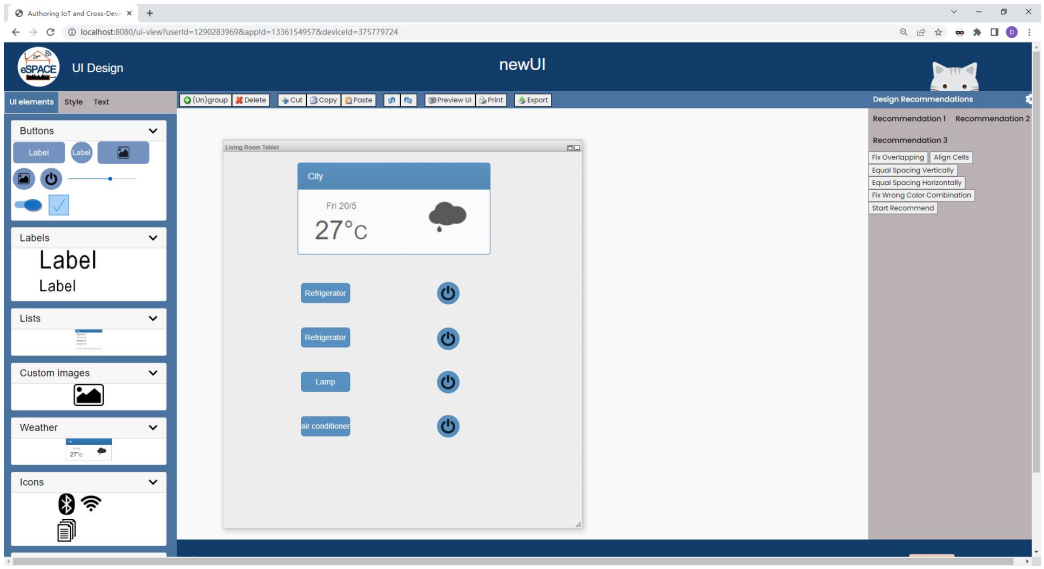

Figure 5.1: UI design view

Next, we move on to the GUI where we solved the overlap problem in the previous step. The second button is utilised to align selected elements and is illustrated in Figure 5.3. First select and align the three elements on the left, then move on to the three elements on the right. As a result, two columns of left-aligned elements are generated.

And then we will show another typical circumstance that users could experience. As shown in the left figure of Figure 5.4, there are batches of elements distributed at different intervals. We start by defining the number of elements as n. Since the default spacing times (n -1) plus the sum of their heights is greater than the difference between the highest and lowest point, that is, $\sum_{i=1}^{n} height_i + (n-1) * default\_spacing > lowest\_point - highest\_point$, the top and bottom elements are not moved, but only the middle elements are relocated to achieve an equidistant distribution. The result of the shifting is shown in the right figure of Figure 5.4. As the elements on the right do not satisfy this condition, that is, $\sum_{i=1}^{n} height_i +$

$(n-1) * default\_spacing < lowest\_point - highest\_point$, the interval between them are set to the default spacing. With this function, as shown in Figure 5.5, end users can save time by eliminating the need to position each element one by one and simply fix the position of the elements at the far ends to achieve the desired even distribution. The function equally and horizontally space elements works the same way as illustrated in Figure 5.6.

When end users want to get recommendations, they can click the *"Start Recommend"* button to start generating recommended solutions according to the current UI design (see Figure 5.7) and generate the corresponding thumbnail in the recommendation sidebar. In the original solution, there are four label buttons, four power buttons and one text label on the canvas. The *"Recommendation 1", "Recommendation 2",* and *"Recommendation 3"* labels in the recommendation sidebar will be replaced by the thumbnails by the corresponding solutions. As mentioned earlier, since only one constraint is currently in place, only the *Recommendation 1* will be replaced now. *Recommendation 2* and *Recommendation 3* are reserved for the further study of other recommendations. Users can click the recommendations to apply the solution. As shown in Figure 5.8, once the recommendations clicked, it will be applied to the current layout, and the elements are relocated according to the algorithm. Even though the elements were originally scattered across the canvas, they have been properly arranged to give a neat layout.

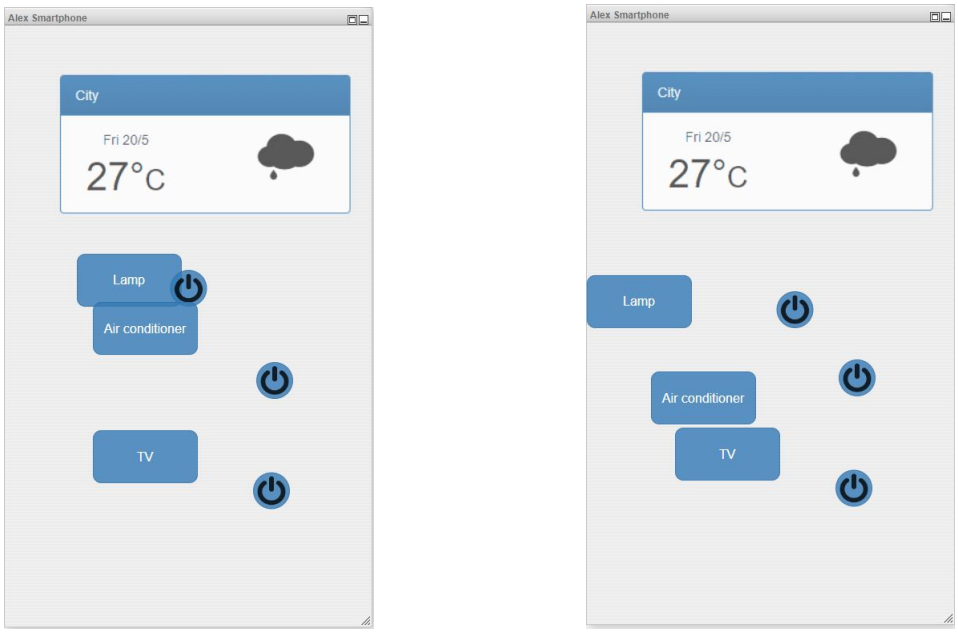

Figure 5.2: Left: Layout before fixing the overlap issue. Right: Layout after fixing the overlap issue.

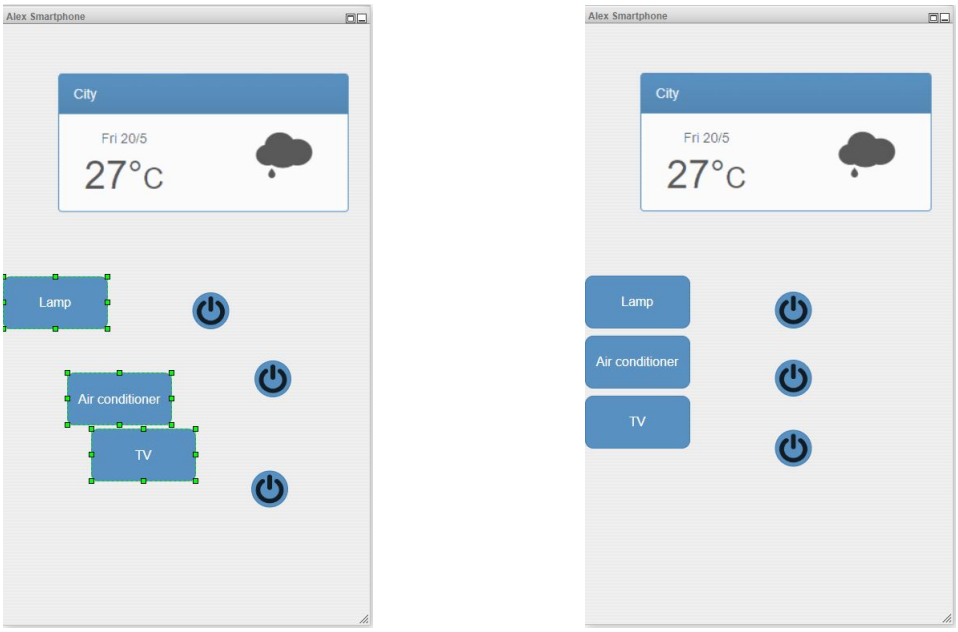

Figure 5.3: Left: Layout before aligning cells selected. Right: Layout after aligning the elements of the left and right columns separately.

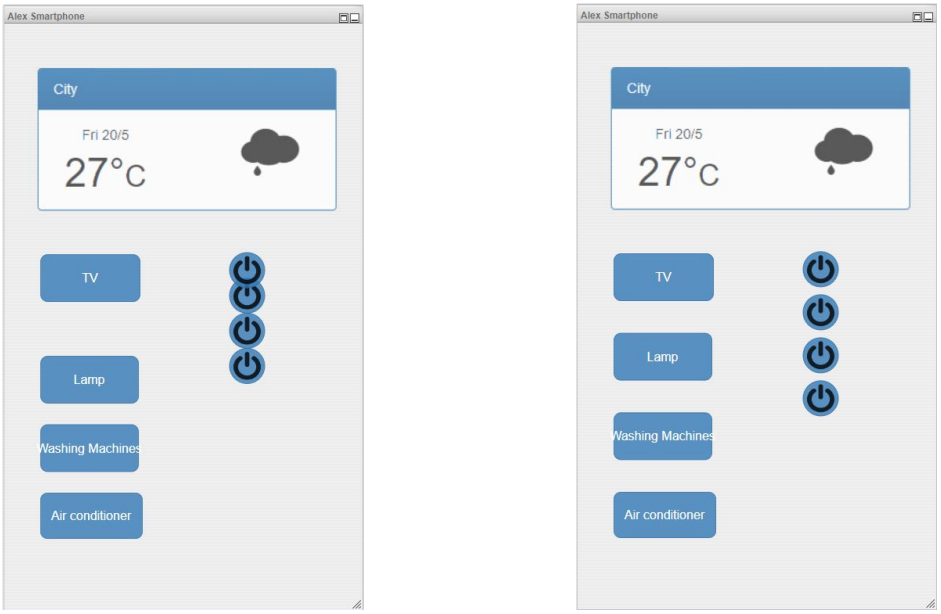

Figure 5.4: Left: Layout before equal spacing of elements. Right: Layout after equidistant spaced distribution of elements vertically.

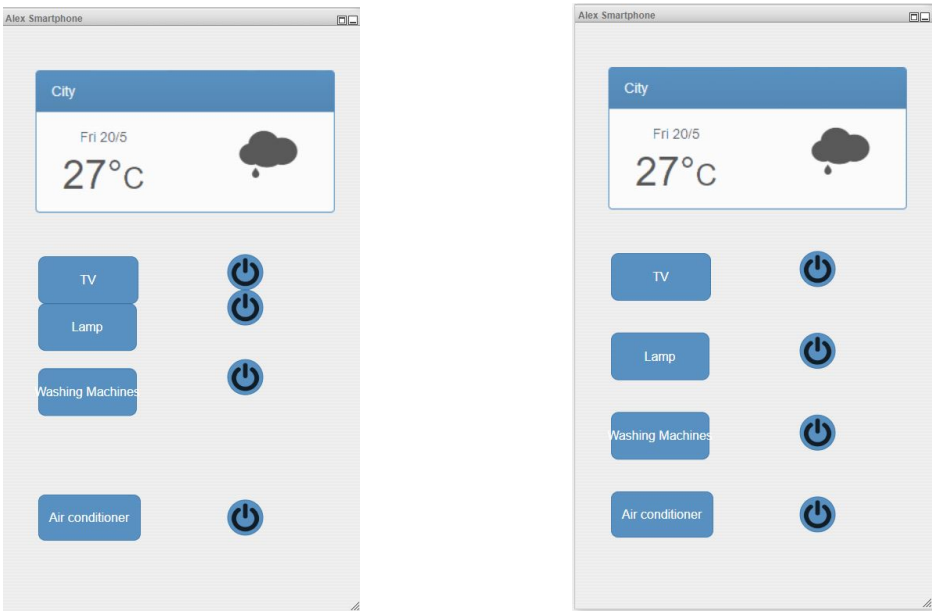

Figure 5.5: Left: Layout before equal spacing of elements, where the positions of the elements at both ends in each of the two columns are aligned. Right: Layout after equidistant spaced distribution of elements vertically.

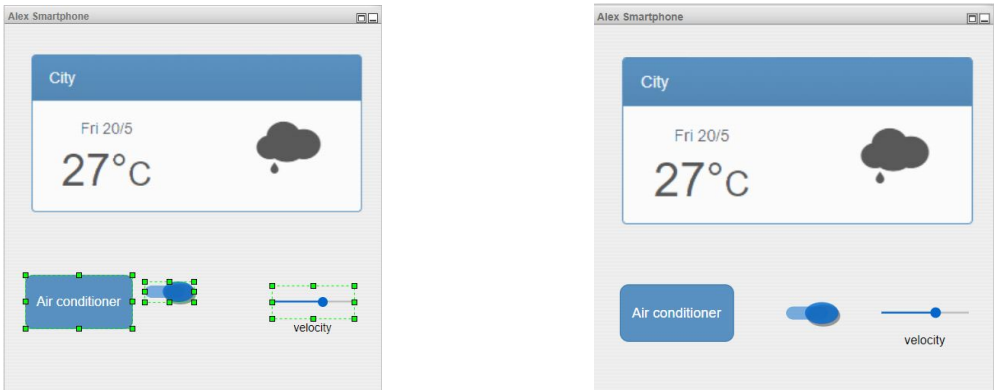

Figure 5.6: Left: Layout before equal spacing of elements. Right: Layout after equidistant spaced distribution of elements horizontally.

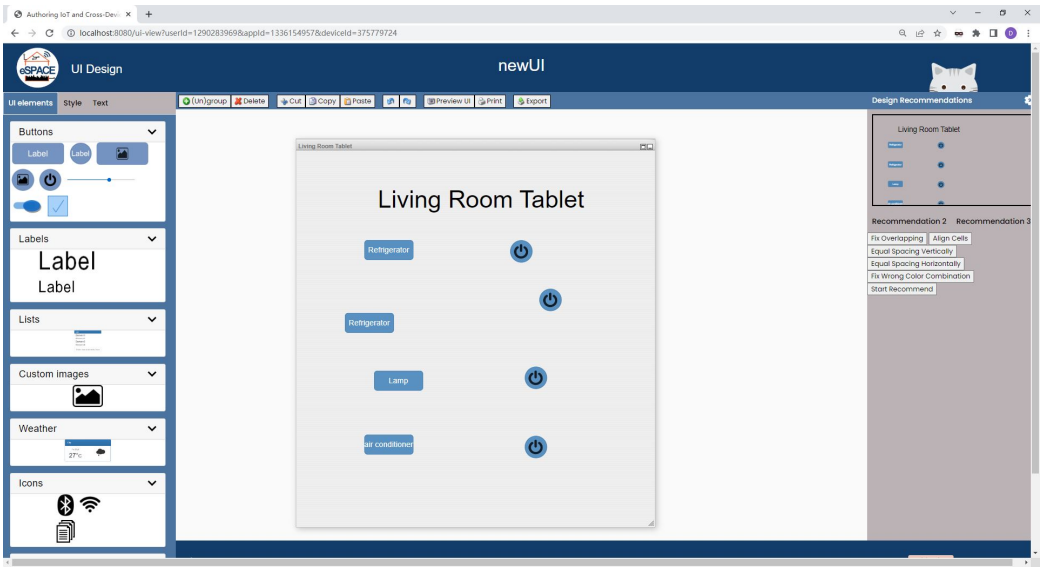

Figure 5.7: A recommended solution generated by clicking *"Start Recommend"* button.

Figure 5.8: Resulting UI after applying the recommended solution to the canvas.

6 Conclusion and Future Work

In this chapter, we are going to critically analyse the accomplishment we have achieved and continue with the limitations we have spotted in the practical application of the tool. Finally, we propose some future work and improvements.

6.1 Conclusion

With smart devices becoming an essential part of our lives, it is becoming increasingly essential to create software that can manage all smart devices in one place. To enhance Internet of Thing devices interactions as well as to meet end users' evolving demands, eSPACE provides UI design views that allow end users to design their own IoT applications.

Due to the fact that end users and novice designers lack professional design experience, they might run into problems when it comes to designing GUIs. Therefore, it is essential to implement effective and feasible tools for providing GUI recommendations. Based on the analysis conveyed, we sought to extend UI design view of eSPACE for end users to design the user interfaces of their applications.

This dissertation has presented recommendations that can help end users to solve the following problems that may arise during the design process: the problem of overlapping, uneven spacing between elements, clutter caused by

unaligned elements. To be specific, the recommendation that can fix overlapping issues is given if a situation arises where some elements are in front of others. This recommendation, in addition to solving the overlap problem, also allows for a more dispersed arrangement of elements. When the users want an even distribution between specific elements, the recommendation provides a layout with a group of evenly distributed and aligned elements. For the need to align elements, the recommendation horizontally or vertically according to their current position.

Apart from these recommendations for local optimisation, by adding constraints, the UI design view also offers recommended solutions that generated based on the present layout. This recommendation would result in a major change to the overall layout.

In order to provide visualised feedback of the recommendations, we implemented thumbnails for UI design views. The thumbnails are able to get screenshots of the optimised layout without changing the current canvas. The optimised layout is displayed by thumbnails, which provide a visual representation that allows the user to choose whether to adopt the suggestion or not. End users are allowed to accept the recommendations by clicking on the thumbnails, or return to the original design if they are not satisfied with the recommendation.

In Chapter 5 we demonstrated that the UI design tool developed based on eSPACE authoring tool can assist end users design their applications to improve their experience in cross device interaction.

6.2 Future Work

Unavoidably, there are some limitations to our solution. The methods for manipulating elements are not flexible enough. For instance, only left-alignment and top-alignment are possible. On the other hand, as far as colour correction is concerned, we have only managed to extract the colours of elements without further analysis and processing. According to the figure/ground law of Gestalt Theory, we could implement the detection of background colour and element colour, element background colour and element font colour to guarantee the readability. These limitations indicate the direction for our future work.

As a tool aimed at assisting graphical user interface design process, our tool still need improvements in the following aspects:

(1) Templates can be provided at the beginning of the design process so that users may adopt these templates and edit them according to their demands. Templates can facilitate the design process by reducing the time

spent on recreating elements and reducing the amount of work involved in dragging and dropping elements, users only need to make minor modifications to the original template, and this method can also provide inspiration for users who are out of ideas.

(2) So far, this thesis has only developed one recommended optimisation to use, namely, that it renders better when there are a consistent number of elements of different types in the layout. Therefore, more different styles of exploration should be offered, depending on the elements already on the canvas. However, building on this thesis, future research should propose more explorations to provide the user diverse choices, contribute to the completion of the partial layouts, and present a better experience of use.

(3) Optimise algorithm: the majority of the algorithms we presently employ are derived based on the geometry information of the individual elements themselves. As a result, the way of rearranging the elements is relatively homogeneous. As it is hard to speculate on all possibilities for different combinations of elements, in the future, heuristic approaches should be introduced into the tool, such as mixed integer programming (MXIP), to cope with a variety of situations with different original layouts.

(4) To get how well the UI design view meets current needs and to get feedback on the use of this view, we need to conduct user research in the future.

We believe that platforms like eSPACE allow us to experiment with novel ways to engage with smart things, and that the design UI tool assists users to manage smart devices in a more efficient, convenient and personalised way.

Bibliography

[1] John B Best. *Cognitive Psychology.* West Publishing Co, 1986.

[2] Jeong-Hyeon Choi, Ho-Youl Jung, Hye-Sun Kim, and Hwan-Gue Cho. Phylodraw: a phylogenetic tree drawing system. *Bioinformatics*, 16(11):1056–1058, 2000.

[3] Niraj Ramesh Dayama, Kashyap Todi, Taru Saarelainen, and Antti Oulasvirta. Grids: Interactive layout design with integer programming. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020.

[4] William Drenttel and Jessica Helfand. Method and system for computer screen layout based on a recombinant geometric modular structure, October 17 2006. US Patent 7,124,360.

[5] Mathias Frisch, Sebastian Kleinau, Ricardo Langner, and Raimund Dachselt. Grids & guides: Multi-touch layout and alignment tools. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1615–1618, 2011.

[6] Lisa Graham. Gestalt theory in interactive media design. *Journal of Humanities & Social Sciences*, 2(1), 2008.

[7] Allen Hurlburt. Layout: The design of the printed page. *IEEE Transactions on Professional Communication*, PC-26(4):213–213, 1983.

[8] Andreas Karrenbauer and Antti Oulasvirta. Improvements to keyboard optimization with integer programming. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 621–626, 2014.

[9] Won Chul Kim and James D Foley. Providing high-level control and expert assistance in the user interface presentation design. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems*, pages 430–437, 1993.

[10] Markku Laine, Ai Nakajima, Niraj Dayama, and Antti Oulasvirta. Layout as a service (laas): A service platform for self-optimizing web layouts. In *International Conference on Web Engineering*, pages 19–26. Springer, 2020.

[11] Kuo-Bin Li. Clustalw-mpi: Clustalw analysis using distributed and parallel computing. *Bioinformatics*, 19(12):1585–1586, 2003.

[12] I Scott MacKenzie. Fitts' law as a research and design tool in human-computer interaction. *Human-computer interaction*, 7(1):91–139, 1992.

[13] I Scott MacKenzie and William Buxton. Extending fitts' law to two-dimensional tasks. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 219–226, 1992.

[14] Aaron Marcus. Principles of effective visual communication for graphical user interface design. In *Readings in human–computer interaction*, pages 425–441. Elsevier, 1995.

[15] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. Designscape: Design with interactive layout suggestions. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 1221–1224, 2015.

[16] Soliha Rahman, Vinoth Pandian Sermuga Pandian, and Matthias Jarke. Ruite: Refining ui layout aesthetics using transformer encoder. In *26th International Conference on Intelligent User Interfaces-Companion*, pages 81–83, 2021.

[17] Ruth Rosenholtz, Yuanzhen Li, Jonathan Mansfield, and Zhenlan Jin. Feature congestion: a measure of display clutter. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 761–770, 2005.

[18] Audrey Sanctorum and Beat Signer. espace: Conceptual foundations for end-user authoring of cross-device and iot applications. 2020.

[19] M_J SEGHERS, JJ Longacre, GA Destefano, et al. The golden proportion and beauty. *Plastic and Reconstructive Surgery*, 34(4):382–386, 1964.

[20] Amanda Swearngin, Mira Dontcheva, Wilmot Li, Joel Brandt, Morgan Dixon, and Amy J Ko. Rewire: Interface design assistance from examples. In *Proceedings of the 2018 CHI conference on human factors in computing systems*, pages 1–12, 2018.

[21] Kashyap Todi, Jussi Jokinen, Kris Luyten, and Antti Oulasvirta. Familiarisation: Restructuring layouts with visual learning models. In *23rd International Conference on Intelligent User Interfaces*, pages 547–558, 2018.

[22] Kashyap Todi, Daryl Weir, and Antti Oulasvirta. Sketchplore: Sketch and explore with a layout optimiser. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, pages 543–555, 2016.

[23] William L Verplank. Graphics in human-computer communication: Principles of graphical user-interface design. *Xerox Office Systems*, 1985.

[24] Max Wertheimer and Willis D Ellis. A source book of gestalt psychology. *Hartcourt, Brace and Co, New York*, 1938.

[25] Pengfei Xu, Hongbo Fu, Takeo Igarashi, and Chiew-Lan Tai. Global beautification of layouts with interactive ambiguity resolution. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 243–252, 2014.

[26] Clemens Zeidler, Johannes Müller, Christof Lutteroth, and Gerald Weber. Comparing the usability of grid-bag and constraint-based layouts. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, pages 674–682, 2012.