A Reinforcement Learning Approach to Determine Horizontal Spaces in Typefaces

N M Ranathunga

Index No. : 13000985

Supervised by

Dr. M I E Wickramasinghe

Submitted in partial fulfillment of the requirements of the
B.Sc. in Computer Science (Hons) Final Year Project in Computer Science (SCS4124)

University of Colombo School of Computing
Sri Lanka
May 24, 2018

Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate: N M Ranathunga

Signature of Candidate
May 24, 2018

This is to certify that this dissertation is based on the work of Mr. N M Ranathunga under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Supervisor: Dr. M I E Wickramasinghe

Signature of Supervisor
May 24, 2018

Abstract

Typeface spacing is a hard problem. It takes countless hours of manual labour to achieve an aesthetically pleasing font, one frequently encounters in digital media. The amount of space between two letters (inter-letter space) significantly contributes to the aesthetically pleasing nature and readability of the typeface. Although inter-letter spacing defines the texture and feel of a typeface and when done accurately yields aesthetically balanced, and an appealing typeface. Setting spacing in a typeface is a tedious and time-consuming task. Hence this research presents an exploratory study investigating potential of reinforcement learning models to fully automate the typeface spacing process.

The proposed reinforcement learning model, first of it's kind was able to achieve good accuracies even with a simple reward function. Some of the visual differences were subtle. Thus, we conclude that reinforcement learning models can indeed be used to model the typeface spacing problem and as one of the first attempts to apply reinforcement learning models in this particular problem domain, this study lays the foundation to future research and studies.

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Dr. M I E Wickramasinghe for the enormous support and guidance provided to me throughout the year to make this research a success. And also I wish to express my gratitude to Mr. Ayantha Randika for the support given to me by providing preprocessed data required for this research. I would also like to thank the evaluation panel for providing me with feedback and showing me the right direction.

I would like to thank my friends for accepting nothing less than excellence from me. Last but not the least, I would like to thank my family for supporting me spiritually throughout writing this thesis and my my life in general.

Contents

Declaration i

Abstract ii

Acknowledgement iii

Contents v

List of Figures vi

List of Tables vii

1 Introduction 1

1.1 Background to the Research . . . 1

1.1.1 Typeface . . . 1

1.1.2 Typeface Designing Process . . . 2

1.1.3 Typeface Spacing (Type Metrics Engineering) . . . 2

1.1.4 Horizontal Spacing Using Bearing Values . . . 3

1.1.5 Kerning . . . 4

1.1.6 Reinforcement learning . . . 4

1.2 Research Aim and Research Questions . . . 5

1.3 Justification for the research . . . 5

1.4 Methodology . . . 6

1.5 Outline of the Dissertation . . . 7

1.6 Delimitations of Scope . . . 7

2 Literature Review 8

2.1 Existence of Spacing Models . . . 8

2.2 Logical Spacing Models . . . 9

2.3 Algorithmic Spacing Models . . . 10

3 Research Design 11

3.1 Dataset . . . 12

3.2 Preprocessing . . . 12

3.3 Action and State Representation 13
3.4 State Space Representation 14
3.5 State Space Reduction 16
3.6 Reward function Design 16

4 Implementation **19**
4.1 Glyph Preprocessing 19
4.2 State representation 19
4.3 Reward Function 21
4.4 Q Learning Algorithm 21

5 Results and Evaluation **22**

6 Conclusion **26**

Appendices **28**

A Spacing Models **29**
A.1 Walter Tracy's method[1] 29
A.2 Miguel Sousa's method[2] 30

B Code Listing **32**
B.1 Glyph Intialization 32
B.2 preprocessing 34
B.3 State Representation 35
B.4 Action Representation 36
B.5 Reward Function 36
B.6 Q Learning 38

[1]The system is described in Tracy, p. 72. The present description was adapted from the book.

[2]Miguel Sousa uploaded a description of his method to Typophile, an online forum related to typeface design. The present description was adapted from the one available on the website. ¡http://typophile.com/node/15794¿

List of Figures

1.1 type designing process 2
1.2 equal spacing vs optical spacing 3
1.3 Bounding box and side bearings 3
1.4 Kerning applied to “AV” character pair 4
1.5 Traditional approach vs. proposed approach 7

3.1 Reinforcement Learning Model 11
3.2 Q Learning equation 12
3.3 Glyph preprocessing 12
3.4 Initial state representation 13
3.5 State representation after ’left shift’ action to glyph “y” 14
3.6 Sample state 14
3.7 State space representation for word ‘AAABBABB’ 15
3.8 Generalized state space representation 16
3.9 Distance calculation between characters 17
3.10 Volume of space representation 17

5.1 Spacing sample with all capital letters 23
5.2 Spacing sample with all simple letters 24
5.3 Spacing sample with mixed simple and capital letters 25

A.1 Standard spaces for uppercase letters in Walter Tracy’s method 29
A.2 Standard spaces for lowercase letters in Walter Tracy’s method 30

List of Tables

3.1 State Representation 14

4.1 State Table 20

5.1 Spacing sample with all capital letters 23
5.2 Spacing sample with all simple letters 24
5.3 Spacing sample with mixed simple and capital letters 25

Chapter 1

Introduction

In typeface designing, defining inter letter spaces is an important task since it affects the look and feel of a text body. However at present, the designers manually set character spacing by first considering character pairs followed by placing the characters in a text paragraph. This process is both time consuming and labour intensive even in fonts with small character sets. Therefore automating the spacing process will save a significant amount of time and money for typeface designers. With this objective, there have been several models (Celso, 2005; Vargas, 2007) built to formulate a function to do the spacing in a logical manner and there have been several approaches to automate the process. Most of these approaches only address the typefaces based on Latin alphabet and do not formulate a generalized model to reach the set objective of the typeface spacing problem. To the extent of our knowledge, these contemporary methods are not used in practical scenarios and typeface designers always revert back to manual spacing, to space their designs. Therefore, this research focuses on developing a more accurate and intuitive way of automating character spacing process using a reinforcement learning approach, in a manner that it becomes aesthetically pleasing to humans and is acceptable to type designers as a spacing tool.

1.1 Background to the Research

1.1.1 Typeface

In typography, a typeface (also known as font family) is a set of one or more fonts each composed of glyphs that share common design features. Each font of a typeface has a specific weight, style, condensation, width, slant, italicization and other visual properties. For example,"ITC Garamond Bold Condensed Italic" means the bold, condensed-width, italic version of ITC Garamond. It is a different font from "ITC Garamond Condensed Italic" and "ITC Garamond Bold Condensed," but all are fonts within the same typeface, "ITC Garamond". Designers of typefaces are called type designers and are often employed by type foundries. In digital typography, type designers are also called font developers or font designers.

1.1.2 Typeface Designing Process

Type designing is the art and process of designing typefaces. For the purpose of this research, the term typeface design will include font design. Figure 1.1 represents a high level abstraction of the process involved in designing a typeface.

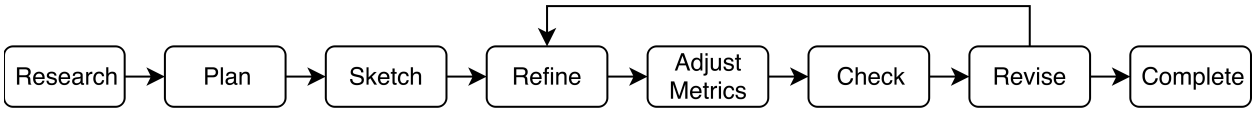

Figure 1.1: type designing process

The typeface designing process starts with research where the type designers studies about design work of others and derives new art forms. In this stage, few initial sketches are made based on inspiring work that the designer come across. Next stage involves with planning, here a set of guidelines are made about the work to be done. Initial sketches are improved. Most type designers prefer to start sketching a typeface design by hand at first. Once they are satisfied with few simple characters or even the whole alphabet, then the design is digitized.

Refining the design always involves with a chosen software. Commercial softwares like Fontlab, Robofont and Glyph are more commonly used in the industry. In this stage, hand-sketched designs are digitally improved and the alphabet is included with extra characters like punctuations etc. Then it comes to type metrics engineering. It involves with adjusting spaces between characters, words, lines etc. This phase is crucial to the usability of a typeface. Many great typeface designs go unused as they are poorly spaced. This research mainly involves with this phase of type designing and will be discussed in much greater depth in latter sections.

Checking how the designs performs in relation to real context is a must. If it performs poor, revisions are necessary until it does perform good. The design is completed once the designer is satisfied with the outcome of the newly designed typeface.

1.1.3 Typeface Spacing (Type Metrics Engineering)

Each glyph consists not only of the shape of the character, but also the white space around it. The type designer must consider the relationship of the space within a letter form (the counter) and the letter spacing between them. Designing type requires many accommodations for the quirks of human perception, ”optical corrections” required to make shapes look right, in ways that diverge from what might seem mathematically right. The purpose of defining the letter space is to make them visually equally distant from each other within words, sentences and paragraphs creating an even value of grey, without darker or lighter areas (Banjanin and Nedeljkovic, 2014). Since the human visual perception is different from mathematical distance, there is no universal formula to do the spacing. Figure 1.2 illustrates the difference between equal spacing and the optical spacing of characters.

Figure 1.2: equal spacing vs optical spacing

This process of adjusting space between characters is known as “fitting”. In typeface designing; character spacing is done using two techniques. One technique is assigning spaces to each character, and other technique is assigning spacing values to character pairs.

1.1.4 Horizontal Spacing Using Bearing Values

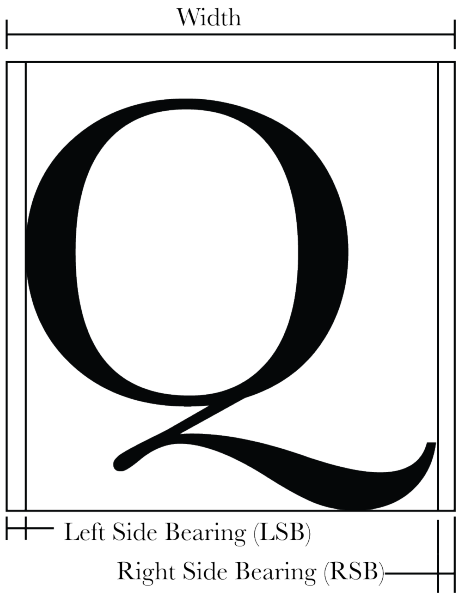

Figure 1.3: Bounding box and side bearings

Within a computerized typesetting system, every character and symbol is conceived of as existing in a box. This bounding box determines the space each letter takes up on a printed page. The space between the character image itself and the edge of the bounding box, called as the side bearing determines how far that character's image will be from the image of a character set next to it. Left Side Bearing (LSB) is the distance between leftmost stroke of the letter and the left edge of the bounding box. Right Side Bearing (RSB) is the distance between rightmost stroke of the letter and the right edge of the bounding box. Collectively, LSB and RSB are known as bearing values. These terms are illustrated in Figure 1.3.

In general, overall appearance of a text body is defined by these spaces unless font uses kerning (refer Section 1.1.5). Bearings values differ from character to character. Determining this type of spacing setting for a whole typeface manually is a tedious task. In an ideal

situation designer has to design all the characters in a font, place it different text bodies and find out an optimal setting by adjusting spacing of all the characters multiple times. This is hugely time consuming and labour intensive. Therefore, designers decides on a value by comparing only some key characters and apply it to all the other characters. As a result, an optimal spacing value cannot be derived only from this spacing method.

1.1.5 Kerning

In typography, kerning is the process of adjusting the spacing between pairs of characters to create a perception of uniformity to achieve a visually pleasing result. It could be considered the hardest part in the spacing process. Most novice type designers tend to skip this part due to it's difficulty. But kerning is a critical aspect in creating a good typeface. For example (Figure 1.4), the letters"AV" would require kerning to get slant edge of "A" closer to the slant edge of "V". With kerning, the space between these character pairs is much more balanced with the spacing of the rest of the font. In a well-kerned font, the two-dimensional blank spaces between each pair of characters have a visually similar area. With different shapes, the blank spaces perceived by human eye is different. Even the spacing between some character pairs are mathematically equal with others, human eye does not perceive it as same. In such instances eye judgment is more important than any mathematical parameters when arranging different shapes in an equally distributed manner (Vargas, 2007).

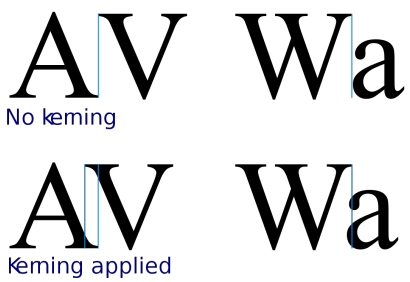

Figure 1.4: Kerning applied to "AV" character pair

1.1.6 Reinforcement learning

Reinforcement learning (RL) is an area of machine learning inspired by behaviourist psychology, concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. It allows machines and software agents to automatically determine the ideal behaviour within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behaviour; this is known as the reinforcement signal. There are many different algorithms that tackle this issue. As a matter of fact, Reinforcement Learning is defined by a specific type of problem, and all its solutions are classed as Reinforcement Learning algorithms. In the problem, an agent is supposed decide the best action to select based on his current state. When this step

is repeated, the problem is known as a Markov Decision Process. Reinforcement Learning allows the machine or software agent to learn its behaviour based on feedback from the environment. This behaviour can be learnt once and for all, or keep on adapting as time goes by. If the problem is modelled with care, some Reinforcement Learning algorithms can converge to the global optimum; this is the ideal behaviour that maximises the reward.

1.2 Research Aim and Research Questions

Research Aim

Typeface spacing is still a tedious task that requires countless hours of manual labour. Typeface designing has become costlier due to the fact that the designers have to spend more time on spacing rather than designing the fonts. We aim to make the typeface designing process much easier and quicker by providing a fully autonomous solution for inter-letter spacing for all kinds of typefaces. This research is a one step forward in the direction of that ultimate aim.

Research Questions

1. Is it possible to design a reward function to do character spacing of a font, in a manner which complies with its design while preserving it's aesthetically pleasing nature?
2. Can reinforcement learning be applied in determining optimal spacing values for typefaces, which is a problem based on human intuition?
3. Possibility of a reinforcement learning system in quantifying an aesthetically pleasing design?

Research Objectives

1. To suggest a reinforcement learning model which is capable of generating spacing values in a given format for a given typeface according to its design parameters.
2. To identify the reward function for the problem of typeface spacing.
3. To determine the best learning algorithm for the specified problem from reinforcement learning techniques such as Temporal Difference Learning (Sutton, 1988), SARSA (Sutton, 1996), Q Learning (Watkins, 1989) etc.
4. To compare the performance of manual spacing with autonomous spacing of typefaces.

1.3 Justification for the research

To this date typeface spacing remains a hard problem. There are hardly any automated spacing models whose algorithms are disclosed to the public. Most of the literature available

regarding the spacing problem is mostly based on logical models that were implemented targeting manual spacing. Main problem associated with these logical models is that , they are based on the fact that they are spaced manually by humans. So these methods have been implemented in a manner which is efficient for humans and not for computers. All the literature related to this field tends to follow a similar approach that was introduced even before 1960's and there is a lack of novel ideas to see the problem in a different perspective. This seems to be a major drawback for the advancement of new typeface spacing models. In this research we hope to lay a foundation to the algorithmic spacing of typefaces by introducing a reinforcement learning model. It's our objective to create a platform where typographic experts can learn new spacing rules by testing it out on the proposed model.

1.4 Methodology

At present, the designers manually set character spacing by first considering individual characters and then by character pairs followed by placing the characters in a text paragraph. Designers first set bearing values (1.1.4 General Horizontal Spacing) for each and every character in the alphabet. Then only kerning values are decided based on their classes. In this process every character pair is not kerned due to its labour intensive nature. As a result traditional approach follows a step by step process in order to avoid the hardship of kerning each and every pair. (E.g. For an alphabet with 'n' letters there are about n^2 kerning pairs). But for a computer it is not a challenging task. Therefor a more direct approach is proposed. According to this approach, every character pair is kerned automatically using reinforcement learning without having to find individual bearing values for each character. Both Right side and left side bearing values are set to zero for easy computation. Figure 1.5 represents a high level representation on how typeface horizontal spacing is done traditionally and according to the proposed approach. The proposed model will be discussed in further details in chapter 3.

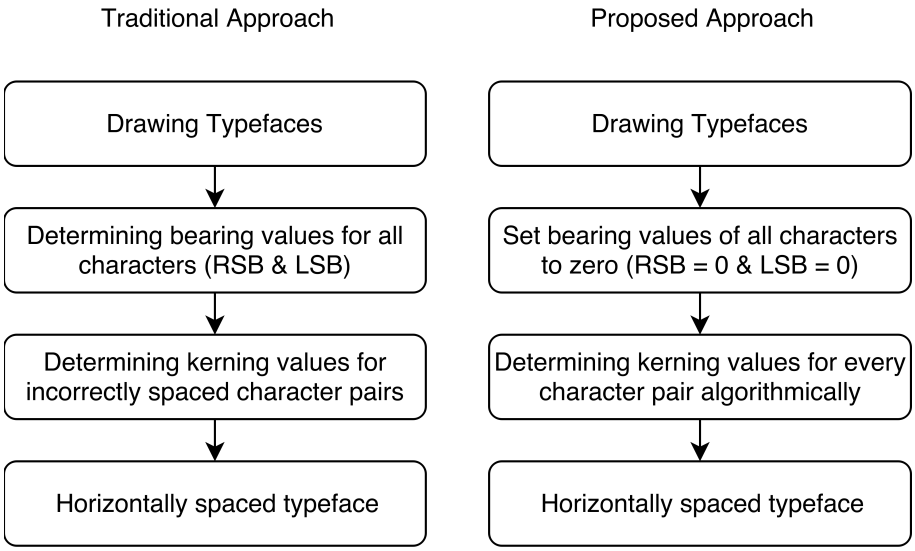

Figure 1.5: Traditional approach vs. proposed approach

1.5 Outline of the Dissertation

Chapter 2 contains a background of the thesis and the literature review of the study. It presents relevant information regarding existing spacing models and problems associated with the existing models. In chapter 3 a detailed description of research methodology will be provided. Chapter 4 contains all the details related to the implementation. Chapter 5 will demonstrate results of each experiment using visual representations.The final chapter contains the conclusion and future work. According to the discussion, the conclusions will be provided as well as some potential avenues of investigation will be pointed out. At last, the work that has to be implemented in the future will be explained.

1.6 Delimitations of Scope

One of the key objectives of this research is to lay a foundation to the development of algorithmic spacing model of typefaces. To achieve the set objective it is required to to keep the scope of our research manageable to a period of one year. As a result some delimitations on the scope of the research have been imposed. Although the proposed model doest put any restrictions on the type of alphabet, in-order to ease up the implementations latin character set will be considered for testing and evaluations. Assuming the aesthetic aspects are universal, system should be able to perform on other character sets as well. But scripts such as Devanagari, Bengali and Arabic will not be considered due to the existence of conjuncts, half letter forms etc.

Chapter 2

Literature Review

Typeface spacing is considered one of the most important stages in typeface designing process such that, a perfectly designed typeface with poor spacing will have no value at all. According to Vargas(2007, p. 4), "The objective of typeface spacing is to make all the glyphs equally distant from each other inside a word through optical adjustments, creating comfortable textures in texts". As it is important it also is a tedious task. There has been a number of attempts to automate the spacing process or at least to make a mathematical formula or a logical model to do the task. Although there are several logical models to do the task of spacing, they are either restricted to latin alphabets or performs poor with different types of alphabets.

2.1 Existence of Spacing Models

Before 1950's people did not believe that spacing can be performed using a pre specified model. It seems that correct spacing seems to be a combination of reasonable judgement of the eye and the aspects of the design of the glyphs.

David Kindersley (1956) said that the judgement of correct spacing does not depend in the eye mechanism by itself. "This is the important thing - the eye - how does it balance, how does it space; yet this is not all, because what we know of spacing seems directly to contradict the simple interpretation of the image on the retina. The cerebral cortex perhaps only uses the retinal image and then blends this information with experience received from the other senses".

W.A. Dwiggins was probably the first to mention that there is a possible set of rules in spacing in the 1940's. Although he mentions that letter spacing can be established based on grouping the letters with similar shapes, he does not state any logical method to convey his idea.

David Kindersley had mixed ideas of the existence of a spacing model. Although he said spacing is connected to information that we obtained through cerebral cortex at beginning, later in his essay he mentions "... somewhere deeper than I could see for the moment there was a set of rules that could be applied to all alphabets, and perhaps all symbols that were

arranged laterally, and that these rules if closely parallel to the function of the eye would achieve good spacing".

2.2 Logical Spacing Models

In 1956 Kaech, proposed his method for letter spacing. He took letter "O" as reference letter for arranging the width of all other as well as for their inner spaces. He talked about "golden mean" and defines the quality of rhythm as a result of perfect relations between those measures.

Kindersley also attempted several systems for spacing letters, by defining their "optical centers" through a photo-electric cell device (Kindersley, 1966), or by searching their "centre of gravity" by eye. He began with spacing capital letters "O" and "I" in string "OIIIO" and when satisfactory results was achieved, he placed all other characters into the place of second "I" and defined their side bearings.

Walter Tracy developed a system for determining letter spacing for Roman alphabets. His method is probably the most in influential and well-known up to now, since it is reproduced in many typeface design publications. Walter Tracy suggests his method of defining letter spacing taking into consideration inner letter space (counter). He started with capital letter "H", measuring space between two vertical stems and then give the left and right side bearing around quarter of that value (half of counter on letter "H" between two letters with vertical stems; e.g. "HH"). He set value of right and left side bearing in word "HHHH" and then put letter "O" between them and adjust its side bearings. When these values are defined (these are called standards) other characters receive their side bearings according to values achieved from standard letters. For small letters he starts with letter "n" defining its left side bearing value as half of its counter and right side bearing value as little bit less than left (because of its rounded right corner). Then he adjusted side bearings for letter "o" in word "nnonn", "nnonon" and 'nnoonn". According to tracy's method amount of white space for other characters in the alphabet are calculated according to values achieved from standard letters. (see Appendix A.1)

Recently, portuguese type designer Miguel Sousa also developed a reliable method while creating his typeface Calouste20. (see Appendix A.2). Harry Carter suggested a method for spacing letters in which counters in letter "m" and ligature "ffi" define an interval between all other strokes.

Carter (1984) stated that "the letters with double upright strokes (n, u, h,) should have a wider interval than m, and similarly, the whites of d,o,p are a little wider than the white in "n". Spacing of "m", "n" and 'o" are the key to provide a proper spacing for all the other characters?.

Main issues with the above mentioned logical models are that they are built based on the latin alphabet and contains rules based on reference characters. Therefor a typeface cannot be spaced unless the reference letter is present in the alphabet. And also these logical

methods are built with the intention that humans are the one's who are going to space the designs. Therefor these models tend to follow a procedure which is easier for humans to perform rather than computers. They follow a two step approach where a determining side bearing is followed by kerning of letter pairs. So most of the rules mentioned in these models are inefficient with a computational model and new ways of algorithmic representations are needed for the models to become computationally effective. In this research we expect to present a computationally effective model to address this problem of spacing.

2.3 Algorithmic Spacing Models

Currently there are only few software tools developed, incorporating above methods to automate the spacing process. "Kernagic"[1] is an open source tool developed as a semi-automatic font spacing tool.

Furthermore, there are services focused on spacing of typefaces and, in the perspective of type designer, these services can also be viewed as automations of type spacing process."Ikern"[2] and "autokern"[3] are such services that are currently available. They provide kerning and spacing typefaces as a service. Their spacing algorithms are not disclosed to the public.

To the best of our knowledge there have been only one attempt to fully automate the spacing problem. This was a neural network approach to determine the horizontal spaces in typefaces proposed by Randika (2016). The developed CNN classifier achieved only a maximum of 47% accuracy in spacing fonts when compared with the manual process. However, this was not enough to capture the subtleties of font spacing thus was not able to surpass the quality of manual spacing. In the neural network approach, it was only able to determine bearing values of glyphs and no kerning values were determinable.

Although computer science is evolving rapidly in the 21^st century , the area of typography seems to be left off. There is hardly any literature related to algorithmic spacing. Few of the existing models are also not disclosed to the public. This seems to be a major drawback for the advancement of new typeface spacing models. In this research we hope to lay a foundation to the algorithmic spacing of typefaces.

[1] https://github.com/hodefoting/kernagic

[2] http://ikern.com/k1

[3] https://fontforge.github.io/autowidth.html

Chapter 3

Research Design

This chapter provides the design methodology of the project, which provides a solid design for the prototype. As discussed in the literature review no previous study has developed a reinforcement learning model to automate the spacing process. So it is required to model the typeface spacing problem to a reinforcement learning problem as shown in Figure 3.1. Here the interpreter is responsible for interpreting the feedback given from the environment to a reward and a new state.

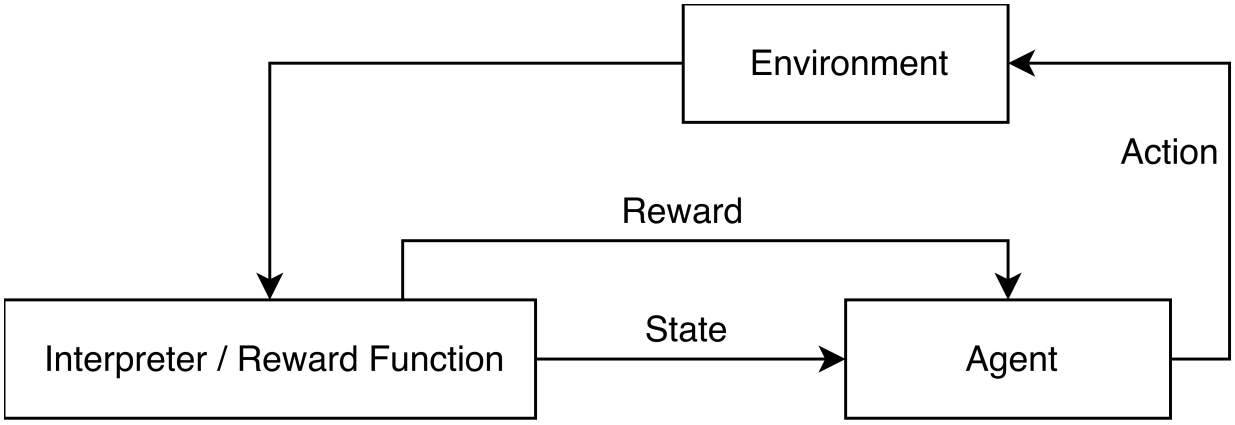

Figure 3.1: Reinforcement Learning Model

When designing the RL system, as we do not have any model of the environment it is necessary to select a model free reinforcement learning algorithm to solve this problem. As a result we have chosen Q-learning, which is one of the model free reinforcement learning techniques. The main logical reasoning behind choosing q learning as the learning algorithm for the model rather than other model-free RL algorithms is that, it could get the work done in the simplest way without complex procedures. The core of the algorithm is a simple value-iteration-update (Figure 3.2). It assumes the old value and makes a correction based on the new information.

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} \right)$$

Figure 3.2: Q Learning equation

Reinforcement learning is widely tested on gaming platforms due to the fact that they have a predefined reward function (score calculated by game rules) and a state transition function (Mnih et. al, 2013). So it is required to model the reward function and the state change function according to the typeface spacing problem before applying any reinforcement learning agent.

3.1 Dataset

We have obtained a dataset of 75 typefaces with 52 characters (A-Z, a-z) for each typeface. These typefaces were selected representing both serif and sans serif classes. Cursive scripts and surreal typefaces were not included to the dataset. Obtained images have been previously preprocessed to a 56 x 56 pixel greyscale images.

3.2 Preprocessing

Left and right marginal spaces of every glyph is removed in the preprocessing stage. Figure 3.3 shows a glyph after being preprocessed. Height of the image is kept unchanged to obtain a uniform image when concatenating multiple glyphs.

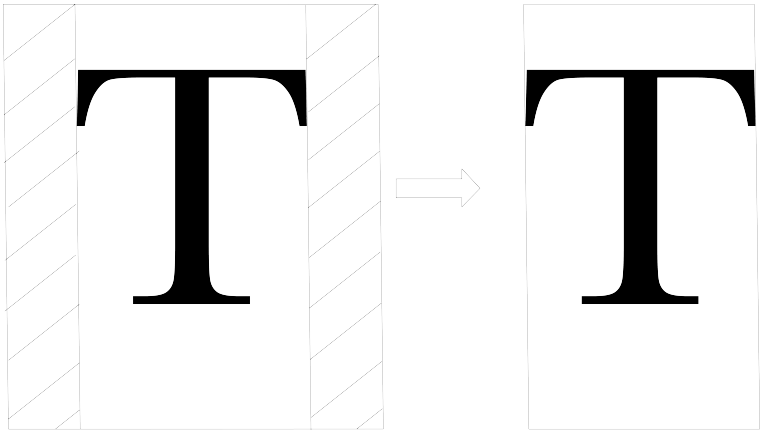

Figure 3.3: Glyph preprocessing

3.3 Action and State Representation

In this model a state is designed as a concatenation of two or more preprocessed glyphs. Let's consider how the word "Type" is represented as a state in a typeface whose alphabet consists of characters {T , y , p , e}. All the glyphs are concatenated in a particular order to form a single state. Initial state for the stated example is shown in Figure 3.4 where,

W_i - Maximum width of the i^{th} glyph (Distance between leftmost stroke and the rightmost stroke of the glyph)

B_i - Base distance of the i^{th} glyph (Distance between starting axis and leftmost stroke of the glyph)

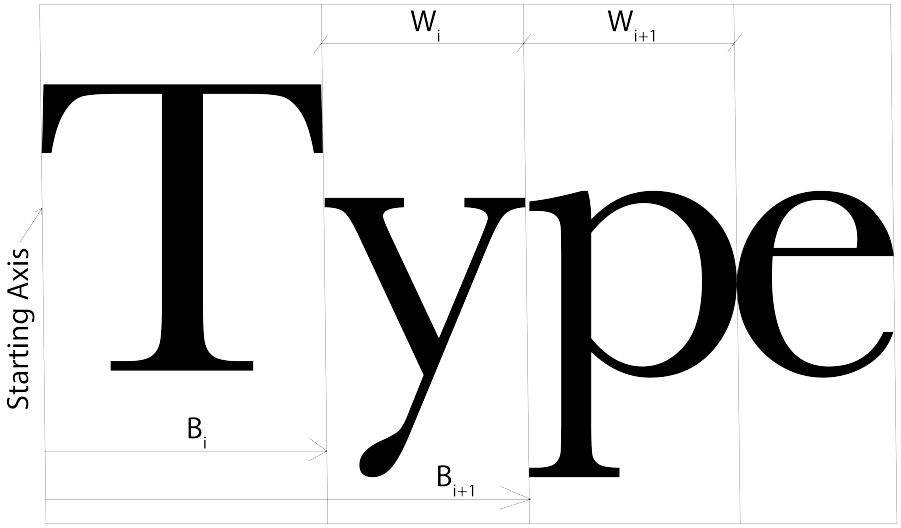

Figure 3.4: Initial state representation

Once a particular glyph is selected an action is performed on the selected glyph. Only actions that the RL algorithm can perform is, it can shift the selected glyph to right or left. Number of pixels that have been shifted left or right is considered the kern value (K_i). Actions of the model are defined as follows:

Right Shift: Shift the selected glyph to the right side by one pixel. This will result in a right shift to all the glyphs followed by the selected glyph.

$B_{i(new)} = B_{i(old)}$ +1 ; i represents the selected glyph and all the glyphs followed.

$K_{i(new)} = K_{i(old)}$ +1; i represents only the selected glyph

Left Shift: Shift the selected glyph to the left side by one pixel. This will result in a left shift to all the glyphs followed by the selected glyph.

$B_{i(new)} = B_{i(old)}$ -1 ; i represents the selected glyph and all the glyphs followed.

$K_{i(new)} = K_{i(old)}$ -1; i represents only the selected glyph

Figure 3.5 represents how state will be represented after applying a left shift to glyph 'y' in the initial state.

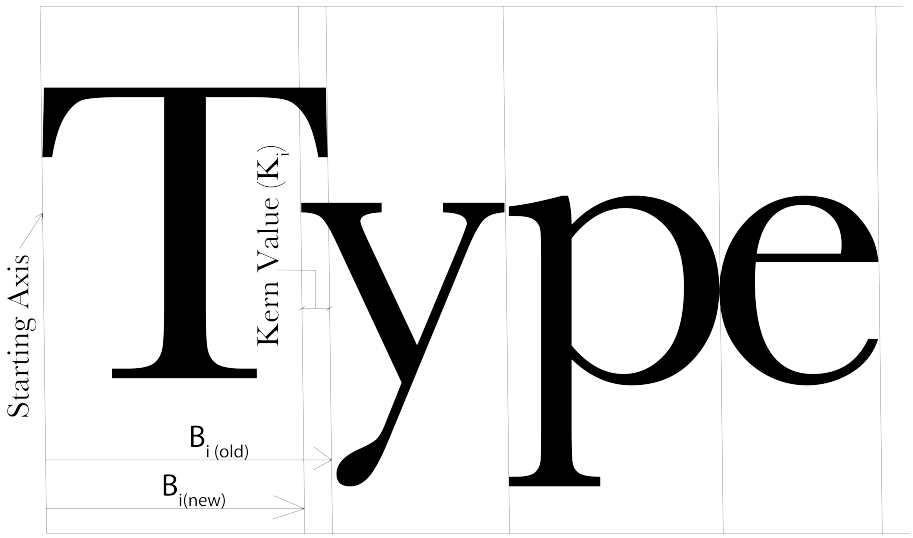

Figure 3.5: State representation after 'left shift' action to glyph "y"

Every state is represented as a raster image using matrix representation. To generate the state, a table is maintained with information of glyph position id, glyph (matrix), max width (Wi), base distance (Bi) and kern value (Ki) as shown in Table 3.1

Table 3.1: State Representation

Glyph Position ID	Glyph (Matrix)	Max Width (W_i)	Base Distance (B_i)	Kern Value (K_i)
1	T	W_1	B_1	K_1
2	y	W_2	B_2	K_2
3	p	W_3	B_3	K_3
4	e	W_4	B_4	K_4

3.4 State Space Representation

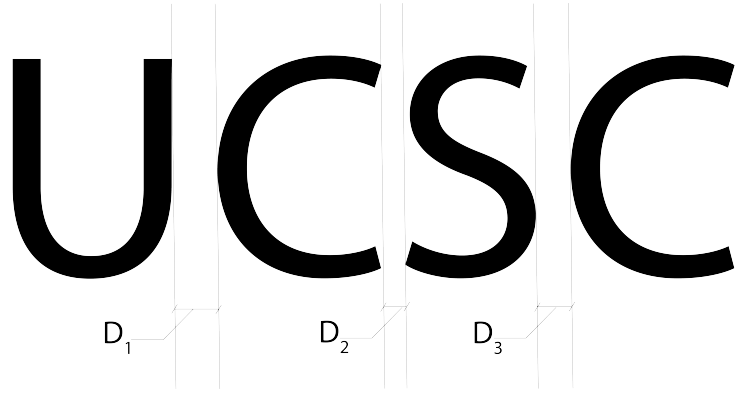

Figure 3.6: Sample state

In order to make the explanations simple, any state will be represented using a simple vector. The above sample state shown in Figure 3.6 will be represented as [D_1, D_2, D_3] This notation will be used throughout the paper to represent any state. Here D(i) is defined as the minimum distance between glyph(i) and glyph(i+1).

Generally, a state will be represented as [D_1,D_2,.....,D_i,.....,D_{n-1}] ; where 'n' is the number of glyphs in the state.

Before understanding a general state space representation, let's consider a typeface for a simple language with only two characters in it's alphabet.

L = $\{w|w \in \{A, B\}^*\}$

There are 2^2 (n^2 ; n - no of glyphs in alphabet) glyph pairs for the given typeface. (AA,AB,BA,BB). It is required to form a single word containing all glyph pairs to find all the kerning values needed for the alphabet. Therefore word 'AAABBABB' which was formed by concatenating all glyph pairs will be considered the input word. This word will be of size 2*n^2. Initial state will be denoted by vector [0,0,0,0,0,0,0] of size (2*n^2)-1 according to above mentioned notation.

For each and every glyph except the glyph at the first position can be shifted left or right. Therefore for every new state, there will be 2*((2*n^2)-1) number of total possible states transitions (see Figure 3.7).

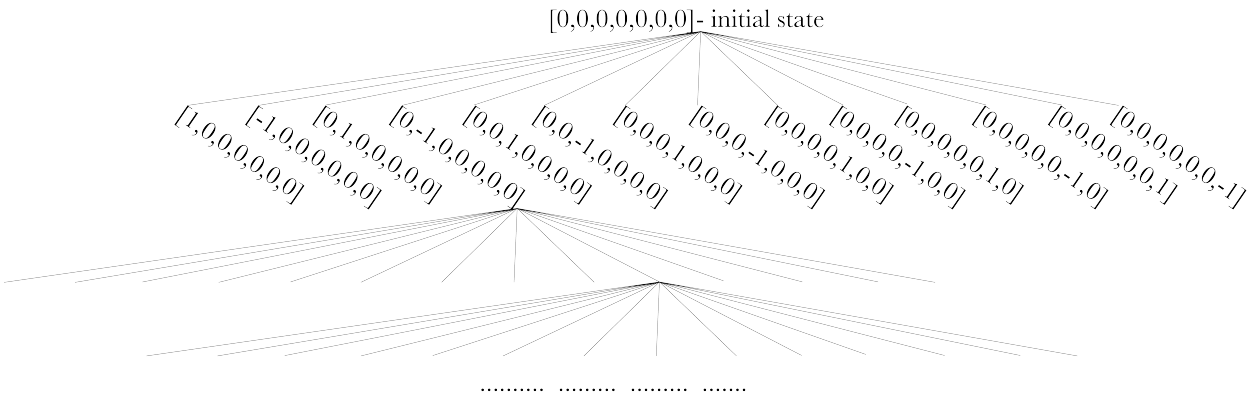

Figure 3.7: State space representation for word 'AAABBABB'

A generalized state space representation is shown in Figure 3.8, where n is the number of glyphs in the alphabet.

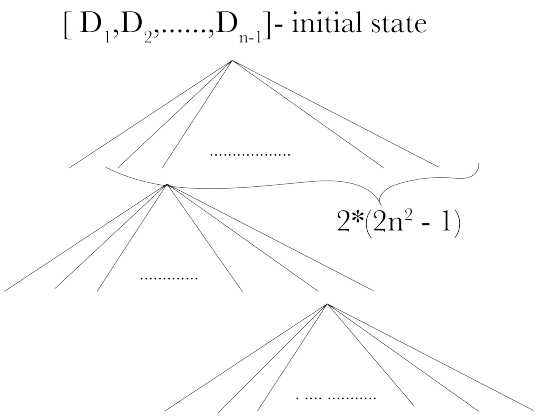

Figure 3.8: Generalized state space representation

If we were to use a brute-force approach to solve this problem it will be computationally expensive for even the smallest alphabets in existence. Hence a reinforcement learning approach is used to solve the stated problem.

3.5 State Space Reduction

D_i which is defined as the minimum distance between glyph(i) and glyph(i+1) could take any value in theory, but when we consider practical implementations this value definitely lies within a range for all alphabets. Hence limiting D(i) to a specific range would reduce the state space. As the minimum distance between two glyphs is always greater than 0 and mostly lower than the maximum width out of all the glyphs, it can be reduced to a range of $0 \leq D_i \leq Max(W_i)$.

This range specification is not considered a hard rule of the proposed model. But this rule will hold for most of the alphabets, even for non-latin alphabets. It's important to note that state space reduction is only optional in the proposed model. But for the ease of implementations in this research, a reduced state space will be followed. This step could be completely eliminated by using neural networks with reinforcement learning. But it will be not discussed in this research due to scope limitations.

3.6 Reward function Design

There are no mathematical rules which defines a good spacing between letters. When humans are given such problems what we intuitively does is that, we try to make it less worse. Same principles can be applied in designing a reward function. We can define rules that are known to be bad rather than implementing rules that are not known to be good. These rules are given a penalty such that the RL agent will take actions to minimize those penalties.

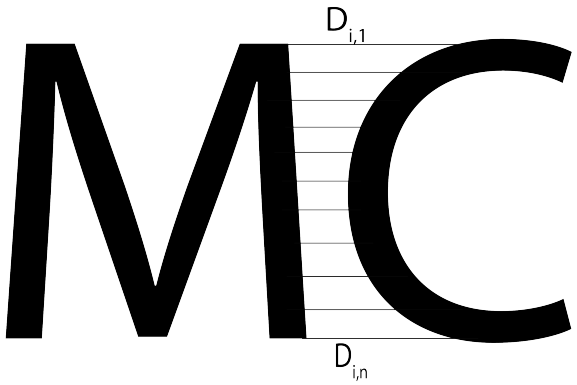

Figure 3.9: Distance calculation between characters

Max and min conditions for the spacing values are defined as the base conditions of the reward function. As denoted in Figure 3.9, Di and Dj are distances measured at each row of the image matrix from the boundary line until a black pixel is found. Distance are obtained in terms of number of pixels.

Minimum distance D_i between two letters are defined as Min($D_{i,1}$,$D_{i,2}$,...$D_{i,j}$,...$D_{i,n}$).

Maximum distance between any two letters are defined as Max(W_1.....,W_n) , where W_i is the maximum width of glyph(i).

Rule 1: If $D_i < 0$ then a penalty is awarded.

Rule 2: If $D_i > Max(W_1.....,W_n)$ then a penalty is awarded.

Uneven space volumes between characters leads to a badly spaced font. According to (Vargas, 2007) objective of a spacing model is to make all the glyphs equally distant from each other inside a word through optical adjustments, creating comfortable textures in texts by balancing internal and external white spaces in letterforms. To implement this idea a volume calculation model is designed such that a penalty is given for characters with bad space volumes.

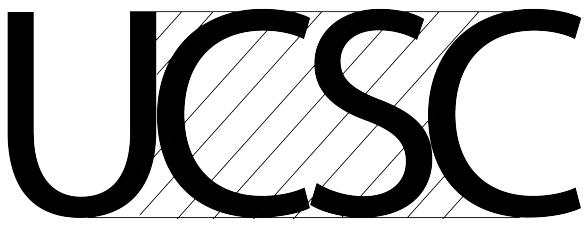

Figure 3.10: Volume of space representation

Figure 3.10 shows how space volumes are represented in the word “UCSC” when it’s in initial state with kerning value initialized to zero. Here volume will be measured in terms of no of pixels in the space area. Volume, V_i will be defined as the no of pixels contained

within two consecutive glyphs. As the model is required to find badly spaced characters, a measure of deviation is defined.

$mean = (\sum_0^n V_i)/n$

$deviation_i = abs(V_i - mean)$

Rule 3: If $deviation_i > 0$ then a penalty is awarded proportional to the deviation.

By giving a penalty in proportion to the deviation, It is expected to adjust the spacing of the characters in a manner that volume will spread evenly across the word. For example deviation between glyphs 'C' and 'S' is the largest resulting in a heavy penalty. To reduce the penalty RL agent have to make 'C' and 'S' closer or make other letters apart. If 'C' and 'S' are made closer, it will fire the condition $D_i < 0$ and result in a heavy penalty. Therefore RL agent will try to make other letters apart instead making C' and 'S' closer. This will result in a much better spaced word.

As the reward function completely based on penalties, to encourage exploration away from the initial state, a penalty is given when current state is same as the initial state.

Rule 4: If $currentstate = initialstate$ then a penalty is awarded.

This is not be considered the finalized reward function. One of the main objectives of this research is to create a platform where other researchers can experiment on different rules related to spacing. This reward function lays the foundation needed in creating a basic spacing model. This model will be able to be bridge the knowledge gap between researchers in the field of typography and computer science. So that typographers with little background of computer science could develop rules and accomplish a desired aim or result.

New rules have to be defined in a manner that the agent will converge to a single optimal state. If it diverges we have to refine the rule set. As future work we would like to suggest that, If algorithm converges to a single state and that state is not optimal in the eyes of the designer, we can design a interrupt function which would give a penalty by human intervention. This will help to include properties of human aesthetic mind that cannot be described in terms of rules.

Chapter 4

Implementation

This chapter mainly focuses on the implementation of the proposed reinforcement learning model which is described in chapter 3. C++ have been used as the primary coding language and OpenCV[1] computer vision library has been used as the primary image processing library.

4.1 Glyph Preprocessing

Our dataset consists of glyphs of size 56*56. In the preprocessing stage, left and right marginal spaces were removed from each and every glyph of the alphabet. Each glyph is converted to a 56*56 matrix and was truncated in order to remove whitespaces in right and left margins of the glyph. Algorithm 1 stated below states the marginal removal algorithm.

Algorithm 1 Glyph Preprocessing

1: `Traverse each column from left to right of the glyph matrix until a column with black pixel is found. We define this as start column.`
2: `Then traverse each column from right to left of the glyph matrix until a column with black pixel is found. We define this as end column.`
3: `Trim the old matrix to a new matrix with the range(start column,end column).`

Full code is listed in Appendix B.2.

4.2 State representation

very state is represented as a raster image using matrix representation. To generate the state, a table is maintained with information of glyph position id, glyph (matrix), max width (Wi), base distance (Bi) and kern value (Ki) as shown in Table 4.1 . This data is sufficient to generate any state of the model.

[1] http://opencv.org

W_i - Maximum width of the i^{th} glyph (Distance between leftmost stroke and the rightmost stroke of the glyph)

B_i - Base distance of the i^{th} glyph (Distance between starting axis and leftmost stroke of the glyph)

K_i - Number of pixels that have been shifted left or right is considered the kern value

Table 4.1: State Table

Glyph Position ID	Glyph (Matrix)	Max Width (W_i)	Base Distance (B_i)	Kern Value (K_i)
1	glyph(1)	W_1	B_1	K_1
..	..	..	..	..
..	..	..	..	..
n	glyph(n)	W_n	B_n	K_n

Here Glyph position ID, Glyph (Matrix) and Max Width columns are initialized at the beginning and no change is done to these variables after initialization. On the other-hand Base Distance and Kern value is updated dynamically with each action as represented by Algorithm 2.

Right Shift: Shift the selected glyph to the right side by one pixel. This will result in a right shift to all the glyphs followed by the selected glyph.

$B_{i(new)} = B_{i(old)}$ +1 ; i represents the selected glyph and all the glyphs followed.

$K_{i(new)} = K_{i(old)}$ +1; i represents only the selected glyph

Left Shift: Shift the selected glyph to the left side by one pixel. This will result in a left shift to all the glyphs followed by the selected glyph.

$B_{i(new)} = B_{i(old)}$ -1 ; i represents the selected glyph and all the glyphs followed.

$K_{i(new)} = K_{i(old)}$ -1; i represents only the selected glyph

Algorithm 2 State and action representation

if $direction = right$ **then**
    $B_{i(new)} = B_{i(old)} + 1()$
    $K_{i(new)} = K_{i(old)} + 1()$
else
    $B_{i(new)} = B_{i(old)} - 1()$
    $K_{i(new)} = K_{i(old)} - 1()$
end if

Full code is listed in Appendix B.3 for state representation and in Appendix B.4 for action representation.

4.3 Reward Function

We have implemented a simple reward function as a foundation to this model.

Rule 1: If $D_i < 0$ then a penalty is awarded.

Rule 2: If $D_i > Max(W_1.....,W_n)$ then a penalty is awarded.

Rule 3: If $deviation_i > 0$ then a penalty is awarded proportional to the deviation. ;$deviation_i = abs(V_i - mean)$, $mean = (\sum_0^n V_i)/n$

Rule 4: If $currentstate = initialstate$ then a penalty is awarded.

Full code is listed in Appendix B.5.

4.4 Q Learning Algorithm

For our learning algorithm , we'll be implementing Q-learning(Algorithm 3). The two key steps in the above pseudocode are steps 3 and 5. We will be following an an ϵ-greedy policy to choose actions based on the current Q-value estimates (step 3). In this policy, the action is selected greedily with respect to the Q-value estimates a fraction (1 - ϵ) of the time (where ϵ is a fraction between 0 and 1), and randomly selected among all actions a fraction ϵ of the time.

The update rule: Q(s,a):=Q(s,a) + α [r+ γ maxa'Q(s',a') - Q(s,a)]

Algorithm 3 Q Learning algorithm

1: `Initialize Q-values Q(s,a) arbitrarily for all state-action pairs`
2: **for** `<untill stopped>` **do**
3: `Choose an action (a) in the current world state (s) based on current Q-value estimates Q( s , . )`
4: `Take the action (a) and observe the the outcome state (s') and reward (r)`
5: `Update Q(s,a):=Q(s,a) + α [r+ γ maxa'Q(s',a') - Q(s,a)]`
6: **end for**

Full code is listed in Appendix B.6.

Chapter 5

Results and Evaluation

This chapter presents the outcomes of the research work described in the previous chapter.We will analyze the results obtained from the proposed under different conditions. As There are different shapes and type of letters, we will consider how the the proposed model performs with variations. It's difficult to provide a numerical comparison of the spacing values. Therefor a visual comparison is provided for the reader to understand the differences between the manually and automatically kerned typefaces.

We also have provided a spacing value table for each and every visual representation. As manually spaced characters consists of Bearing values, we have converted those bearing values to a single kerning value for easy comparison using the below equation,

Target Kern Value(i) = RSB(i) + LSB(i+1) + Manually Kerned Value(i)

where 'i' is the glyph position id.

Performance of the proposed model is evaluated based on all capital characters, all simple characters and mixed simple and capital letters. We can observe that model performs extremely well for all capital letters (Figure 5.1 , Table 5.1). For all simple letters we can observe that spacing values are little higher than the expected (Figure 5.2 , Table 5.2). It performed poorly when capital latters and simple letter are mixed (Figure 5.3 , Table 5.3). It is important to understand that this is reward function carries the foundation in buiding a better spacing model. It is satisfactory that model has been able to output such fair space values with only a limited number of rules.

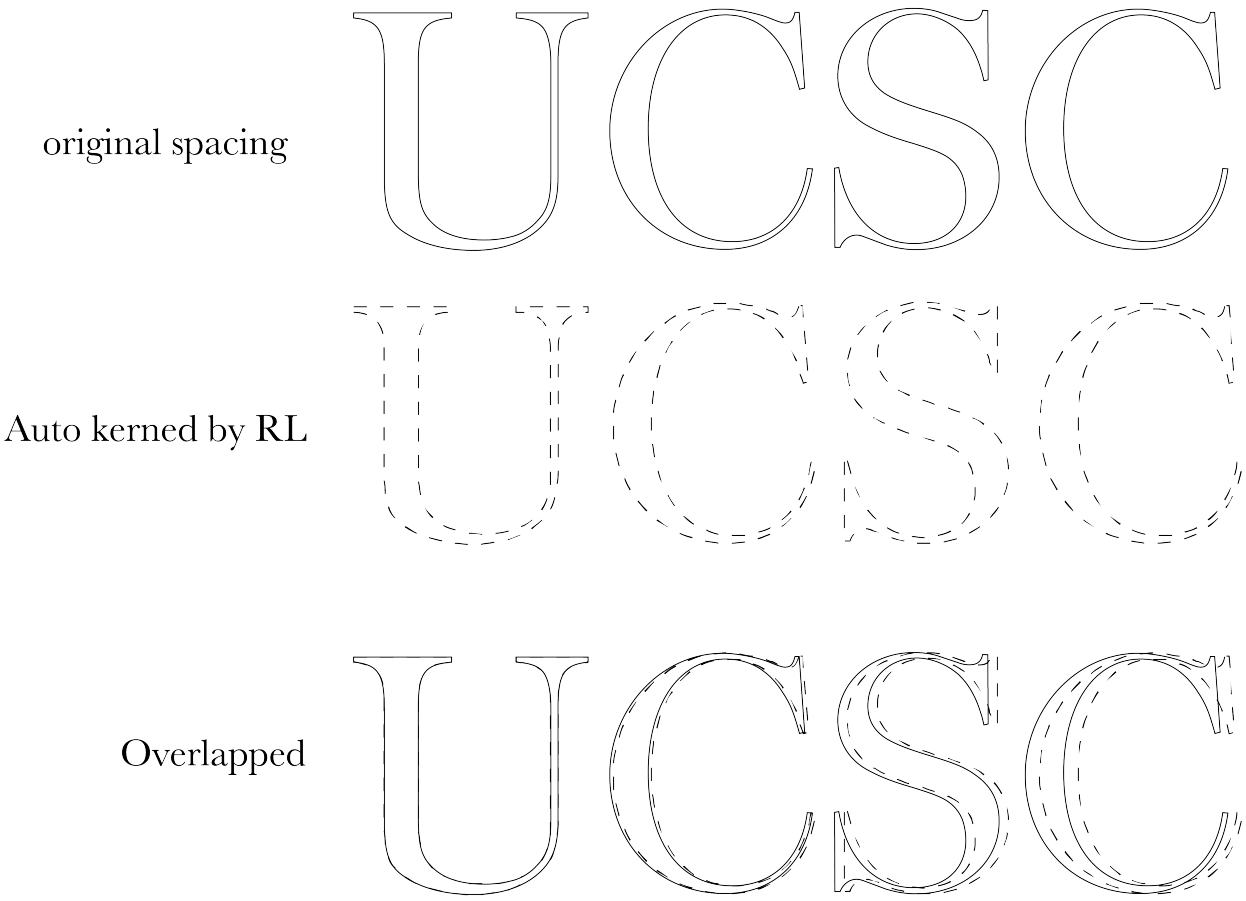

Figure 5.1: Spacing sample with all capital letters

Table 5.1: Spacing sample with all capital letters

Name		U	C	S	C
Width(units)		725	627	509	627
Original Typeface (Manual)	LSB(units)	26	42	40	42
	RSB(units)	26	28	39	28
	Kern(units)	0	0	0	
Target Output (RSB,LSB=0)	LSB(units)	0	0	0	0
	RSB(units)	0	0	0	0
	Kern(units)	68	68	81	
Actual Output (By RL)	LSB(units)	0	0	0	0
	RSB(units)	0	0	0	0
	Kern(units)	74	79	90	

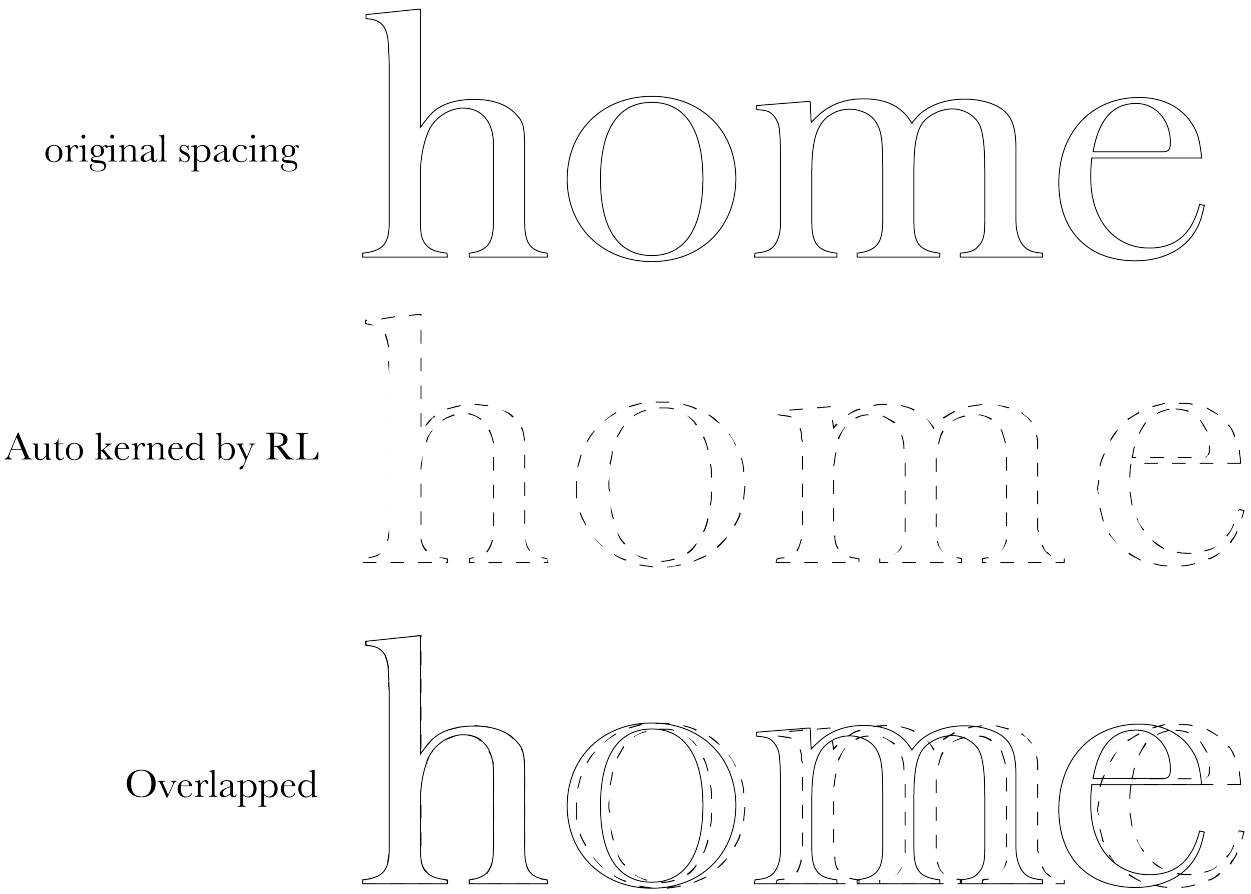

Figure 5.2: Spacing sample with all simple letters

Table 5.2: Spacing sample with all simple letters

Name		h	o	m	e
Width(units)		550	502	858	433
Original Typeface (Manual)	LSB(units)	12	36	19	36
	RSB(units)	23	36	12	23
	Kern(units)	0	0	0	
Target Output (RSB,LSB=0)	LSB(units)	0	0	0	0
	RSB(units)	0	0	0	0
	Kern(units)	59	55	48	
Actual Output (By RL)	LSB(units)	0	0	0	0
	RSB(units)	0	0	0	0
	Kern(units)	74	67	67	

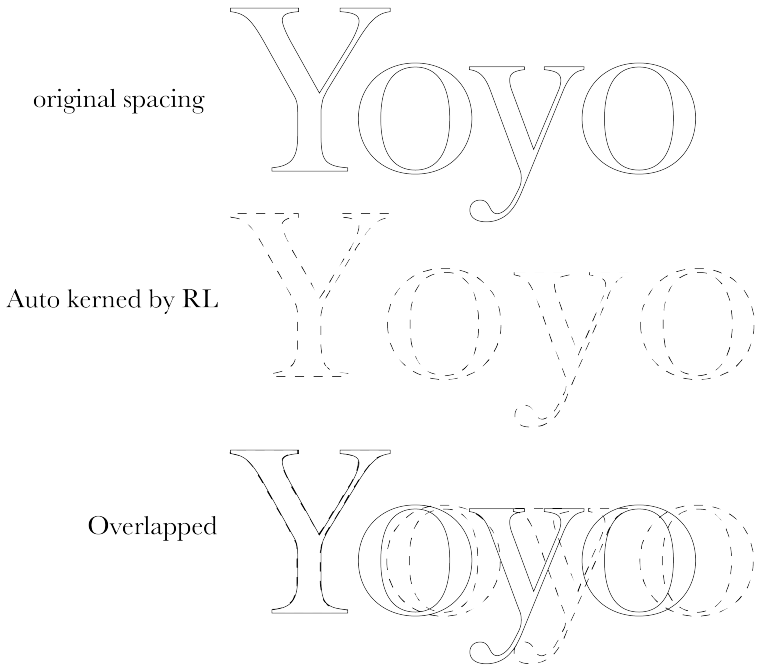

Figure 5.3: Spacing sample with mixed simple and capital letters

Table 5.3: Spacing sample with mixed simple and capital letters

Name		Y	o	y	o
Width(units)		550	502	858	433
Original Typeface (Manual)	LSB(units)	-63	36	-32	42
	RSB(units)	-62	36	-37	43
	Kern(units)		-115	-15	0
Target Output (RSB,LSB=0)	LSB(units)	0	0	0	0
	RSB(units)	0	0	0	0
	Kern(units)		-141	-11	5
Actual Output (By RL)	LSB(units)	0	0	0	0
	RSB(units)	0	0	0	0
	Kern(units)		-5	30	88

Chapter 6

Conclusion

Imitating the human perception and intuition is an extremely hard task for a computer model. As per the related work, solving the typeface problem has never been attempted with a reinforcement learning learning approach. This research has been able to achieve promising results, which can lead to better insights and better solutions for the problem.

The reward function of the proposed model will be an ideal playground for other researchers and typographers to test the spacing rules that they have in their minds, This platform will help to achieve much accuracies in font spacing in future. As a field with only little literature, this research will motivate other researchers to keep on finding the optimal spacing function. Although it seems a unimaginable challenge, if we are to come close to that ultimate goal it would be a huge step in the world of typography.

However, despite these difficulties, the researcher has been able to achieve results with significance potential for further improvements. Furthermore, the reinforcement learning models can be suggested as an effective machine learning model with potential to discover this relationship in typefaces and solve the typeface spacing problem.

References

Celso, A. (2005). Rhythm in type design. M.A. University of Reading.

Kaech, W. (1956). Rhythm and Proportion in Lettering. Switzerland, Olten: Otto Walter Ltd, p.67

Lawler, B. (2006) "Typographic Terms" in The Official Adobe Print Publishing Guide, Second Edition: The Essential Resource for Design, Production, and Prepress. San Francisco: Adobe Press.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Sutton, R.S., 1988. Learning to predict by the methods of temporal differences. Machine learning, 3(1), pp.9-44.

Sutton, R.S., 1996. Generalization in reinforcement learning: Successful examples using sparse coarse coding. Advances in neural information processing systems, pp.1038-1044.

Tracy, W. (2003). Letters of Credit: A View of Type Design. Boston: David R Godine, pp. 70-80.

Randika, A. (2016). 'Neural Network Approach to Determine the Horizontal Spaces in Typefaces', thesis, University of Colombo School of Computing, Srilanka.

unifiedfontobject.org, (2012). UFO3. [online] Available at: http: //www.unifiedfon tobject.org/versions/ufo3/index.html [Accessed 16 Apr. 2017].

Vargas, F. (2007). Approaches to applying spacing methods in seriffed and sans-serif typeface designs. M.A. University of Reading.

Watkins, C.J.C.H., 1989. Learning from delayed rewards (Doctoral dissertation, University of Cambridge).

Appendices

Appendix A

Spacing Models

A.1 Walter Tracy's method[1]

Uppercase

1. The first step is to set the spacing for the H. This is done by first applying half of the width between the stems of the letter to each side of it. Then the spacing is refined through the word 'HHHH'.

2. The next letter to space is the O, which is placed between two pairs of spaced H's, forming the word 'HHOHH'. The side bearings of the O are adjusted until the word is balanced. Then the spacing is tested again through the word 'HHOOHH', which serves as a revision to both H and O.

3. With the spaces of H and O adjusted, the other glyphs are spaced as indicated in Figure A.1.

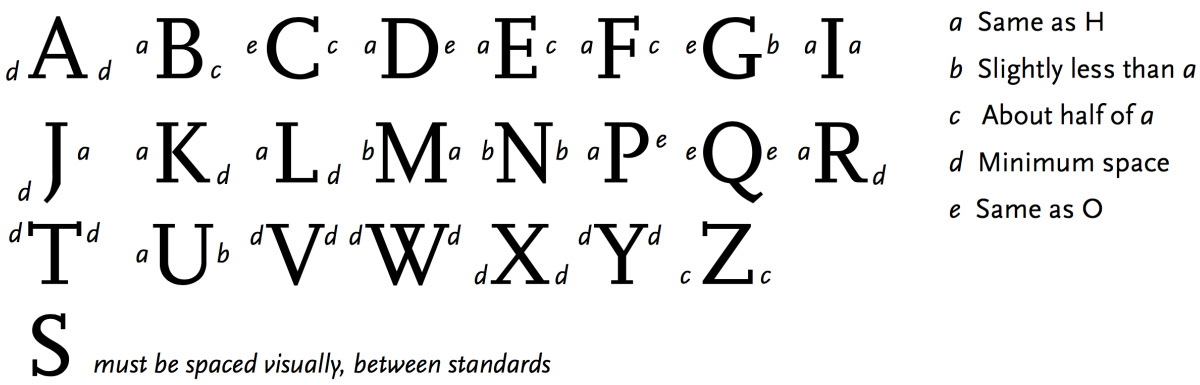

Figure A.1: Standard spaces for uppercase letters in Walter Tracy's method

[1]The system is described in Tracy, p. 72. The present description was adapted from the book.

Lowercase

1. The standards are n and o. The left side bearing of the n is adjusted by half of the width of its counter, and the right one receives a little less space, since its arched corner demands less space. The spacing is then refined through the word 'nnnn'.

2. The o is adjusted by setting the words 'nnonn', 'nnonon' and 'nnoonn'.

3. With the spaces of n and o well regulated, the rest of the glyphs are spaced as indicated in Figure A.2.

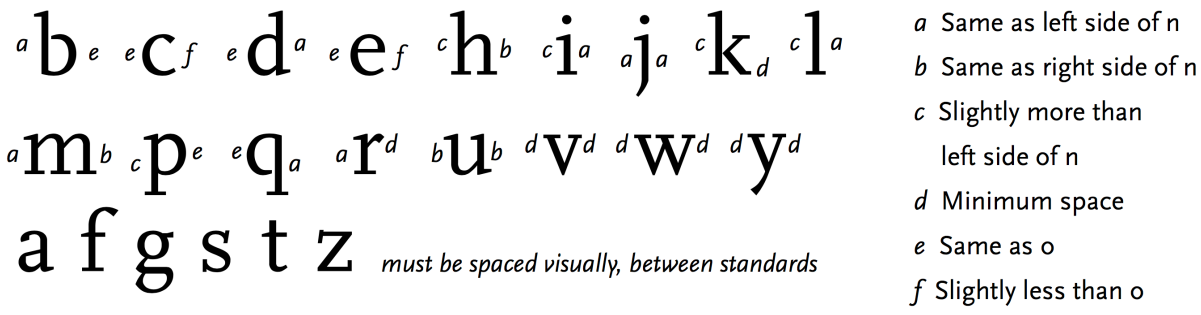

Figure A.2: Standard spaces for lowercase letters in Walter Tracy's method

A.2 Miguel Sousa's method[2]

The system divides the lowercase alphabet in three groups of letters:

First group: b d h i l m n o p q u

The amount of space on both sides of the letters are related to, at least, one side of another element in the same group. Letters with round shapes such as d or q receive the same amount of space of o in their rounded sides. Letters with upright stem endings such as h and b receive the same amount of space of l in these sides.

Second group: a c e f j k r t

The letters in this group each have one side with similar shapes (and spaces) to letters of the first group, but their other side has no relation to any character in the first group.

[2]Miguel Sousa uploaded a description of his method to Typophile, an online forum related to typeface design. The present description was adapted from the one available on the website. ¡http://typophile.com/node/15794¿

Third group: g s v w x y z

The spaces of these letters have no direct relation to any other character. Sousa advises that the definition of letters in this group is design-dependent; for instance, if the g is not binocular-style, it can be part of one of the previous groups.

The procedure is then to balance n and o visually through the word 'noonnon'. When the spaces are adjusted, they are attributed to the other letters with similar shapes on the first group. The necessary adjustments and corrections are made through words containing only these first group letters, generated by the adhesiontext tool. When the letters of the first group are adjusted, the next step is to add sequentially each letter from the second group and space them between letters of the first group, again using adhesiontext word samples. The process is repeated with elements of the third group. Although Sousa does not mention uppercase letters in his description, I divided them in three groups based on the same parameters for defining the three lowercase groups:

First group: B D E F H I N O Q Second group: C G J K L P R ?Third group: A M S T U V W X Y Z

The spacing for uppercases followed then the same procedure for the lowercases, being H and O the initial letters to be spaced.

Appendix B

Code Listing

B.1 Glyph Intialization

```
class Glyph
{
private:
    static int glyphId;
    int id;
    int maxWidth;
    cv::Mat glyph;

public:
    Glyph(){
        id = glyphId;
        glyphId++;
    }

    void setGlyphMat(cv::Mat glyphInput){
        int startcol = 0;
        int endcol = glyphInput.cols ;

        // clean left margins of the glyph
        for (int i = 0; i < glyphInput.cols; i++) {
            int count = 0;
            for (int j = 0; j < glyphInput.rows; j++) {
                count += (int)glyphInput.at<uchar>(j,i);
            }

            if (count != 0){
                startcol = i ;
```

```
        break;
    }
}
// clean right margins of the glyph
for (int i = glyphInput.cols -1 ; i >= 0; i--) {
    int count = 0;
    for (int j = 0; j < glyphInput.rows; j++) {
        count += (int)glyphInput.at<uchar>(j,i);
        if(count != 0)
            break;
    }
    if (count != 0){
        endcol = i + 1;
        break;
    }
}

//maxWidth calculation
maxWidth = endcol - startcol +1;
maxWidthArr[id] = maxWidth;

//trimming the margins
glyph = glyphInput.colRange(startcol, endcol);
glyphMatArr[id] = glyph;


//calculate spacemaps
int distance;
for (int i = 0; i < glyph.rows; i++) {
    distance=0;
    for (int j = 0; j < glyph.cols; ++j) {
        if((int)glyph.at<uchar>(i,j) == 255 || j== glyph.cols-1){
            if(j== glyph.cols-1){
                glyphLeftSpaceMap[id][i] = ++distance;
            }

            else{
                glyphLeftSpaceMap[id][i] = distance;
            }
            break;
```

```
            }
        distance++;
        }
    }

    for (int i = 0; i < glyph.rows; i++) {
        distance=0;
        for (int j = glyph.cols-1; j >= 0; j--) {
            if((int)glyph.at<uchar>(i,j) == 255 || j== 0){
                if(j== 0){
                    glyphRightSpaceMap[id][i] = ++distance;
                }
                else{
                    glyphRightSpaceMap[id][i] = distance;
                }
                break;
            }
            distance++;
        }
    }
}
```

B.2 preprocessing

```
    int startcol = 0;
    int endcol = glyphInput.cols ;
    // clean leftside of the glyph
    for (int i = 0; i < glyphInput.cols; i++) {
        int count = 0;
        for (int j = 0; j < glyphInput.rows; j++) {
            //check wether any black pixel is present
            count += (int)glyphInput.at<uchar>(j,i);
        }
        if (count != 0){
            startcol = i ;
            break;
        }
    }
    //clean rightside of glyph
```

```
        for (int i = glyphInput.cols -1 ; i >= 0; i--) {
            int count = 0;
            for (int j = 0; j < glyphInput.rows; j++) {
                //check wether any black pixel is present
                count += (int)glyphInput.at<uchar>(j,i);
                if(count != 0)
                    break;
            }
            if (count != 0){
                endcol = i + 1;
                break;
            }
        }
        glyph = glyphInput.colRange(startcol, endcol);
```

B.3 State Representation

```
int maxWidthArr[alphabetSize];
int baseDistanceArr[alphabetSize];
int kernValueArr[alphabetSize]={0};
cv::Mat glyphMatArr[alphabetSize];

cv::Mat gernerateState(){
    cv::Mat currentState = glyphMatArr[0];
    cv::Mat gapMat;
    cv::Mat leftPart,rightPart,commanPart,prevState,temp;
    for(int i=1;i<alphabetSize;i++){
        if(kernValueArr[i-1]==0){
            //cout << glyphMatArr[i].type()<<endl;
            cv::hconcat(currentState, glyphMatArr[i], currentState);
        }
        else if(kernValueArr[i-1] >0){kernValueArr[i-1],CV_8U);
            cv::hconcat(currentState, apMat, currentState);
            cv::hconcat(currentState, glyphMatArr[i], currentState);
        }
        else if(kernValueArr[i-1] <0){
            //cv::hconcat(currentState, glyphMatArr[i], currentState);
            leftPart = currentState.colRange((currentState.cols-1)-(
                ↪ kernValueArr[i-1]*-1) , (currentState.cols-1));
```

```
            rightPart = glyphMatArr[i].colRange(0, (kernValueArr[i-1]*-1));
            addWeighted( leftPart, 1, rightPart, 1, 0.0, commanPart);
            leftPart = currentState.colRange(0, (currentState.cols-1)-(
                ↪ kernValueArr[i-1]*-1));
            rightPart = glyphMatArr[i].colRange((kernValueArr[i-1]*-1),
                ↪ glyphMatArr[i].cols-1);
            cv::hconcat(leftPart, commanPart, currentState);
            cv::hconcat(currentState, rightPart, currentState);
        }
    }
    return currentState;
}
```

B.4 Action Representation

```
void action(int glyphId, bool direction){
    if(glyphId > 0 && glyphId<alphabetSize){
        if(direction){ //true -right
            kernValueArr[glyphId-1] += 1;
            baseDistanceArr[glyphId] += 1;
        }
        else if(abs(kernValueArr[glyphId-1] )+2 < glyphMatArr[glyphId].cols
            ↪ ){
            kernValueArr[glyphId-1] -= 1;
            baseDistanceArr[glyphId] -= 1;

        }
    }
}
```

B.5 Reward Function

```
void generateStateMap(){
 //maximumSpaceBearing Calculation
    for(int i=0;i<alphabetSize;i++){
        if(maximumSpaceBearing < maxWidthArr[i]){
            maximumSpaceBearing = maxWidthArr[i];
```

```
        }
    }
    maximumSpaceBearing = maximumSpaceBearing/2;
    noOfStates = pow(maximumSpaceBearing,2);
}

int spaceMap[alphabetSize][glyphHeight];
int spaceVolume[alphabetSize];

void generateSpaceMap(){
    for(int i=0;i<alphabetSize-1;i++){
        for(int j=0;j<glyphHeight;j++){
            spaceMap[i][j] = glyphRightSpaceMap[i][j] + glyphLeftSpaceMap[i
                ↪ +1][j] + kernValueArr[i];
            spaceVolume[i]=spaceMap[i][j];
        }
    }
}

int rewardfunction(cv::Mat state, int glyphId ){
    int reward=0;
    int penalty=0;
    //cv::Mat currentState;
    generateSpaceMap();
    for(int i=0;i<alphabetSize;i++){
        for(int j=0;j<glyphHeight;j++){
                    // Base condition implementation
            if(spaceMap[i][j] > maximumSpaceBearing || spaceMap[i][j] <0 ){
                penalty++;
            }

        }
    }
    int mean=0;
    int variance=0;

    for(int i=0;i<alphabetSize;i++){
    mean += spaceVolume[i] / alphabetSize;
    }
    variance = abs(mean - spaceVolume[glyphId] );];
```

```
    penalty = variance/2;

    if(intialState == gernerateState()){
        penalty++;
    }

    reward = reward-penalty;
    return reward;
}
```

B.6 Q Learning

```
void qLearn(){
    // Set learning parameters
    float learningRate = .8;
    float gamma = .9;
    int numEpisodes = 2000;
    int s;
    int a;
    //create lists to contain total rewards and steps per episode
    std::list<double> rewardList;
    for(int i=0;i<numEpisodes;i++){
        s = rand() % noOfStates;
        glyphId = rand() % alphabetSize;
        a = rand() % 2;
        double rewardAll = 0;
        int j =0;
        //The Q-Table learning algorithm
        while(j<999){ // no of state transitions untill eventual break(
            ↪ finish exploration)
            j++;
            //Choose an action by greedily (with noise) picking from Q table
            a = get_max_Q(s) + (rand() % noOfStates;+1)*(1./(i+1)))
            //Get new state and reward from environment
            generateState(); // render current state
            action(glyphId, a);
            r = rewardfunction(stateTranslate(s,glyphId));
            //Update Q-Table with new knowledge
            Q[s][a] = Q[s][a] + lr*(r + y*get_max_Q(s1)) - Q[s][a])
```

```
            rewardAll += r
            s = s1
        }
        rewardList.append(rAll)
    }
}
```