Alphabet Glyph Generator

William Yoong, Jacob Lim, Malody Hoe

# 1 Introduction

You must use this template doc as your starting point. Read the whole thing before writing, since it will save you time. Don’t change margins, spacing, fonts, or styles. Everything is set up to accurately measure page count.

This is the introduction. The introduction is the place to motivate your article and explain what will be covered. Typically this is done in one or two paragraphs, but occasionally authors will stretch it out to three or four. The introduction is incredibly important and should be refined or rewritten once the article is done to make sure it’s the best part of your article. From previous experience, the introduction and conclusion are often the weakest parts, which is unfortunate since this can cause readers to immediately stop reading your article.

The tone of your article should be expert and friendly. Carefully construct relevant examples that bring out the genius and relevance of the article. Try to inspire the reader to go off and actually use it. The article should be aimed at the intermediate/expert level, but still easy to understand. Strive for clear and concise sentences that are easy to digest. More people will benefit if they understand it the first time it is read. Above all, always keep a heavy slant toward practical game development (as opposed to academic research).

Many games, especially Sci-Fi and Fantasy, have their own made-up languages with their corresponding alphabets. These alphabets usually go through multiple iterations before the designers can finally settle on the right one for the language. This means many man hours are taken up by the artists to produce said iterations. What if there was a way to quickly produce these alphabets, simply by keying in a few values? That is what Alphabet Glyph Generator (AGG) intends to do. For example, coming up with the Demon Alphabet in DOOM could have been much easier if artists had access to AGG to come up with a good base to start designing. This would save a lot of time in their workflow, and allow for more iteration. Si-Fi and Fantasy games take up a substantial chunk of games in the market and there’s still more to come. Having an AGG will help improve the game and speed up workflow.

# 2 Alphabets Design

Before getting into the algorithm of AGG, we should first have a basic understanding of alphabets and the patterns that they have. Luckily, there are a great deal of languages on Earth to draw examples from. First, let us take a look at the English alphabet. The English alphabet is a good simple example to start of with, as the alphabet mainly consists of no more than 2 strokes to produce (with the exception of some letters). The English alphabet mainly consists of lines and arcs (curves), as with many other languages. In English, the alphabet generally flows well between strokes. With the exception of some letters, the strokes usually connect to each other. The minimal number of strokes also helps with this. This is one pattern to keep in mind when making an AGG – continuity and flow.

[](https://www.thebeijinger.com/sites/default/files/thebeijinger/blog-images/313215/characters.png)

Figure 1 Examples of common Chinese characters.

Next, let’s take a look at a more complicated language, that uses a completely different design philosophy in creating their characters/alphabets. Referring to Figure 1, you can see a table of common Chinese characters. Chinese characters are described as logograms, and with good reason. The Chinese glyphs are said to be derived from the shapes of what the words represent.

[](http://www.ancientscripts.com/chinese.html)

Figure 2 List of evolutions to from drawings to modern Chinese.

In Figure 2, the first line, shows the evolution of the Chinese word for human. The word evolves from a simple drawing of a person to an exaggerated form to enhance the features of the object in question. Then, over the years, it has been simplified to what it is today. This is the design philosophy of the Chinese when it comes to (creating) their characters. So, another pattern to keep in mind is to draw reference from real life and using it as the basis (or seed) for the glyph.

# 3 The Algorithm of Alphabet Glyph Generator

The language we will be using for Alphabet Glyph Generator will be JavaScript. The reason for using JavaScript is because it has access to canvas element, which has tools for us to draw strokes onto the screen such as straight lines, Bezier curve, arcs, and many others.

Listing 1 Randomizing the number of parts to draw.

let parts = [];

for(let part = 0; part < maxParts; ++part) {

int diff = maxParts – minParts;

if(random.int(0, diff) < part - minParts + 1)

break;

parts.push(genPart());

}

So the first step we take is to generate the number of parts we want to draw between the minimum and maximum range. Each part consists of many different strokes and they will be drawn together at the end, using genPart() to create each parts. Most alphabets start off from top to bottom and left to right, thus we add a bias such that start points will often start from the top-left position.

Once we determine our first starting point, we randomize a number between the minimum and maximum stroke and set that as the number of strokes for the current part. For each stroke in a part, there is a chance to reuse a previous stroke if any or we generate the stroke per normal. When generating a stroke, there are biases that are factor in. One of them determine the odds of generating a straight line, the other determine the odds of generating an arc. So all in all, there are three cases, generating a straight line, an arc or a Bezier curve as shown in Listing 2.

Listing 2 Generating a stroke, determine by RNG

function genStroke(point) {

if(random.number(0, 1) < settings.jagginess)

{

let points = [ point ];

for(let i = 1; i < Line.requiredPoints; ++i)

points.push(point = genNextPoint(point));

return new Line(points);

}

else

{

if(random.number(0, 1) < settings.arcWeight)

{

let points = [ point ];

for(let i = 1; i < Arc.requiredPoints; ++i)

points.push(point = genNextPoint(point));

return new Arc(points);

}

else

{

let points = [ point ];

for(let i = 1; i < Bezier.requiredPoints; ++i)

points.push(point = genNextPoint(point));

return new Bezier(points);

}

}

}

At the end of each stroke, there is another random number that is generated to see if we want to add a finishing touch to the part. From the final point that marks the end of the stroke, we can either add a circle or a tick depending on another number that is randomly generated. That’s it; we are done with our very first stroke, now we continue generating the remaining strokes to complete our first part. Once the part is done, we see if there are any parts left to be generated, if there are no more parts to be generated, we are left with the final product that will be drawn on the screen that can be seen in Figure 3.

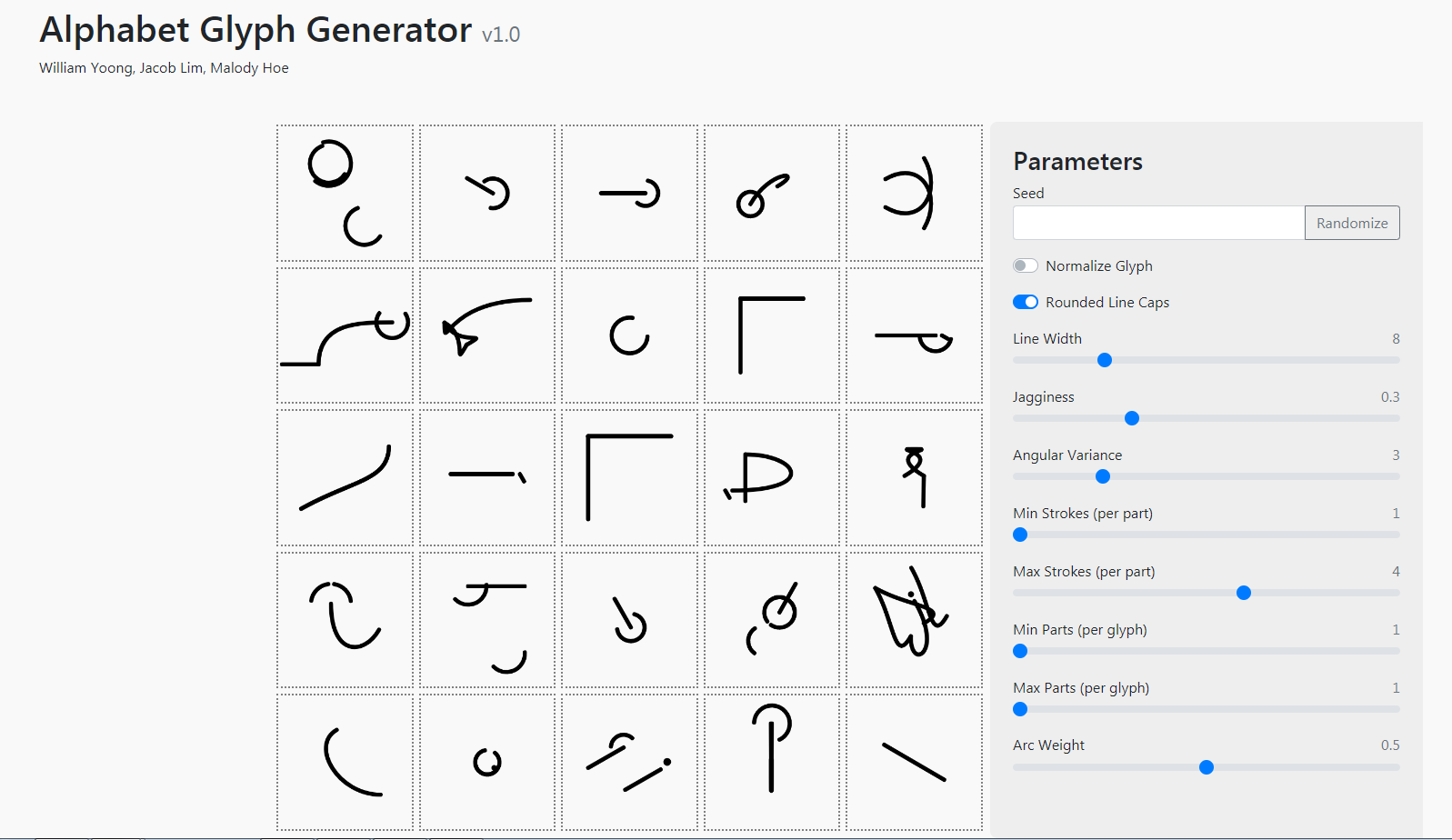


Figure 3 The final product

# 2 Headings and Styles

There are four levels of headings, but we recommend that you primarily stick with just the first two (readers start losing track of your hierarchy once they get to heading 3 and 4). Make sure you only use the “Styles” in Word’s ribbon bar (don’t change fonts or sizes). For example, this paragraph is written with the Normal style (Times New Roman 12 point). If you include references to code, like the class Route or the variable iCount, then make sure the particular code reference is in the Code style (Courier New 12 point).

When titling your headings, avoid overly cute or humorous titles. Yes, you are a funny writer, but we need to keep a level of professionalism. Subtle humor might be OK in limited amounts, but your section editor reserves the right to change or clean it up. It seems obvious, but make your titles helpful to understanding what follows.

# 4 Figures and Tables

## 4.1 Figures

(Note: The previous two consecutive headings without text between them is a problem. This is called a stacked heading and isn’t allowed. You’ll need to break these up with at least one sentence of normal text.)

Please embed figures directly into the Word file (but we MUST have the original figure files). Figures should be numbered starting at 1 and have a description that is a full and complete sentence (ending in a period). Refer to figures by name, as in the following example. Please refer to Figure 1.

Collision_Detection_and_Resolution_Figure02

Figure 1 Overlap testing is problematic for small, fast moving objects, like bullets.

Non-screenshot figures must be vector-based, such as using Word drawing capabilities (don’t use bitmaps unless they are screenshots). Any program that can produce vector-based art should work, as long as it can output a file in Adobe Illustrator (.ai), PDF, or EPS.

## 4.2 Tables

Tables differ from figures in that they have their description placed above the table. Tables should be numbered starting at 1 and the description should be a full sentence ending in a period. Please don’t make any fancy formatting within tables.

Table 1 Statistics gathered from past player moves.

|  |  |  |
| --- | --- | --- |
| Player Sequence | Occurrences | Frequency |
| Low Kick, Low Punch, Uppercut | 10 times | 50% |
| Low Kick, Low Punch, Low Punch | 7 times | 35% |
| Low Kick, Low Punch, Sideswipe | 3 times | 15% |

# 5 Code

If you are going to include code fragments, you’ll need to follow some simple guidelines:

1. Make sure that the code is using the Code style (Courier New 12 point).
2. Make sure there are NO TABS in the code (replace tabs with four spaces).
3. Make sure every line of code is 67 characters or less to avoid wrapping.
4. Class names, types, namespaces, attributes, methods, variable names, keywords, functions, modules, commands, properties, parameters, values, objects, events, XML/HTML tags, and similar elements inside the normal body of text should be listed in Courier New (12 Point) style.
5. Include the minimum amount of code required to get your point across. You can place additional code on the book’s website, so only the pertinent parts need to take up space in your article. The following is an example of some code.
6. Word will auto-capitalize the first letter of sentences, so watch out for this in your code.

## 5.1 Code Listings

You can have a couple lines of code right after a paragraph, but if you have too much, it should be a code listing, like Listing 1. The entire listing will be surrounded by a gray box in the final book.

Listing 1 Function to add one to an integer.

int AddOne(int a)

{

return a + 1;

}

# 6 Equations

If you want to highlight an equation, you can refer to it like Equation 1. Use the Equation Editor in the INSERT tab of the ribbon bar. You can place equations in-line in a paragraph, but it usually ends up making the paragraph spacing look weird, so avoid it if possible.

(1)

# 7 Editing

Your section editor and peers will help edit/comment on your article. When they do so, they will be using the “reviewing” tools in Word.

The amount of editing that section editors perform will vary based on your skill as a writer. If sections of your article are poor (poor English, confusing, unclear, etc.), those sections might be dropped or rewritten by yourself or the editors. If the entire article doesn’t meet the standards of the book, the decision to drop the article can be made by the section editor or series editor. This is rare, but it inevitably happens to one or two articles.

If your article requires heavy editing in places, section editors might help out, but you are free to adjust or rewrite these sections yourself. You’ll see the final version that will be printed so that you can approve or adjust any editing. In the past, most authors appreciate the help, if only to point out what was unclear so that they could rewrite it themselves. Others were grateful for having a fresh set of eyes understand the ideas that were trying to come out but needed a little help.

# 4 Conclusion

Your article should include a brief conclusion summarizing what the reader should have learned from the article. You can also motivate the reader to use the technique or give guidance for taking the ideas further.

Learning to see patterns in alphabets can range from easy to hard, but essential when making an alphabet glyph generator as it decides the type of glyphs it will produce. Some alphabets are simple, needing only a few simple strokes to construct, while other languages are more complicated, needing multiple strokes (usually disjointed) to construct. The more patterns one can derive from a range of languages, the more generic the AGG will become, hence, more versatile. Conversely, deriving patterns from a select group of languages will move the AGG towards a more specialized route. With this basic knowledge, you can start to consider what kind of language you are interested in making and start working on an AGG that fits your designs.

# 9 References

A list of references is not absolutely mandatory, but it’s highly recommended. Below is a list of guidelines for each type of reference. When the reference in the text, be sure it is inside of a sentence and supports the ideas presented [Lewis 91]. Don’t refer to a bracketed reference directly as a noun.

## 9.1 Authored book:

[Woods 06] Woods, D.D. and E. Hollnagel. 2006*. Joint cognitive systems*. Boca Raton: Taylor & Francis.

## 9.2 Chapter in multiauthored book:

[Wiens 83] Wiens, J. A. 1983. Avian community ecology: An iconoclastic view. In *Perspectives in ornithology*, ed. A. H. Brush, and G. A. Clark, 355–403. Cambridge: Cambridge Univ. Press.

## 9.3 Journals:

[Terborgh 74] Terborgh, J. 1974. Preservation of natural diversity. *BioScience* 24:715-22.

## 9.4 Electronic journal:

[Testa 00] Testa, B., and L. B. Kier. 2000. Emergence and dissolvence in the self-organisation of complex systems. *Entropy* 2, no. 1 (March): 1-25. http://www.mdpi.org/entropy/papers/e2010001.pdf.

## 9.5 Unpublished Documents:

[Schwartz 00] Schwartz, G. J. 2000. Multiwavelength analyses of classical carbon-oxygen novae. PhD diss., Arizona State Univ.

[O’Guinn 87] O’Guinn, T. C. 1987. Touching greatness. Paper presented at the annual meeting of the American Psychological Association, New York.

## 9.6 Online Documents:

[Adamic 99] Adamic, L.A., and B.A. Huberman. 1999. The nature of markets in the World Wide Web. Working paper, Xerox Palo Alto Research Center. http://www.parc.xerox.com/istl/groups/

iea/www/webmarkets.html (accessed March 12, 2001).

[U.S. Census Bureau 00] U.S. Census Bureau. 2000. Health insurance coverage status and type of coverage by sex, race, and Hispanic origin. Health Insurance Historical Table 1. http://www.census.gov/hhes/hlthins/historic/hihisttl.html.