Alphabet Glyph Generator

William Yoong, Jacob Lim, Malody Hoe

# 1 Introduction

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 expedite 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 off 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 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 curves, 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());

}

The first step is to generate the number of parts we want to draw between the minimum and maximum range. Each part consists of many different strokes which will be drawn together at the end, using genPart() to create each part. Most alphabets start off from top to bottom and left to right, thus we added 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 as per the algorithm. When generating a stroke, there are biases that are factored in. One of them determine the odds of generating a straight line, the other determine the odds of generating an arc. 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, determined 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.

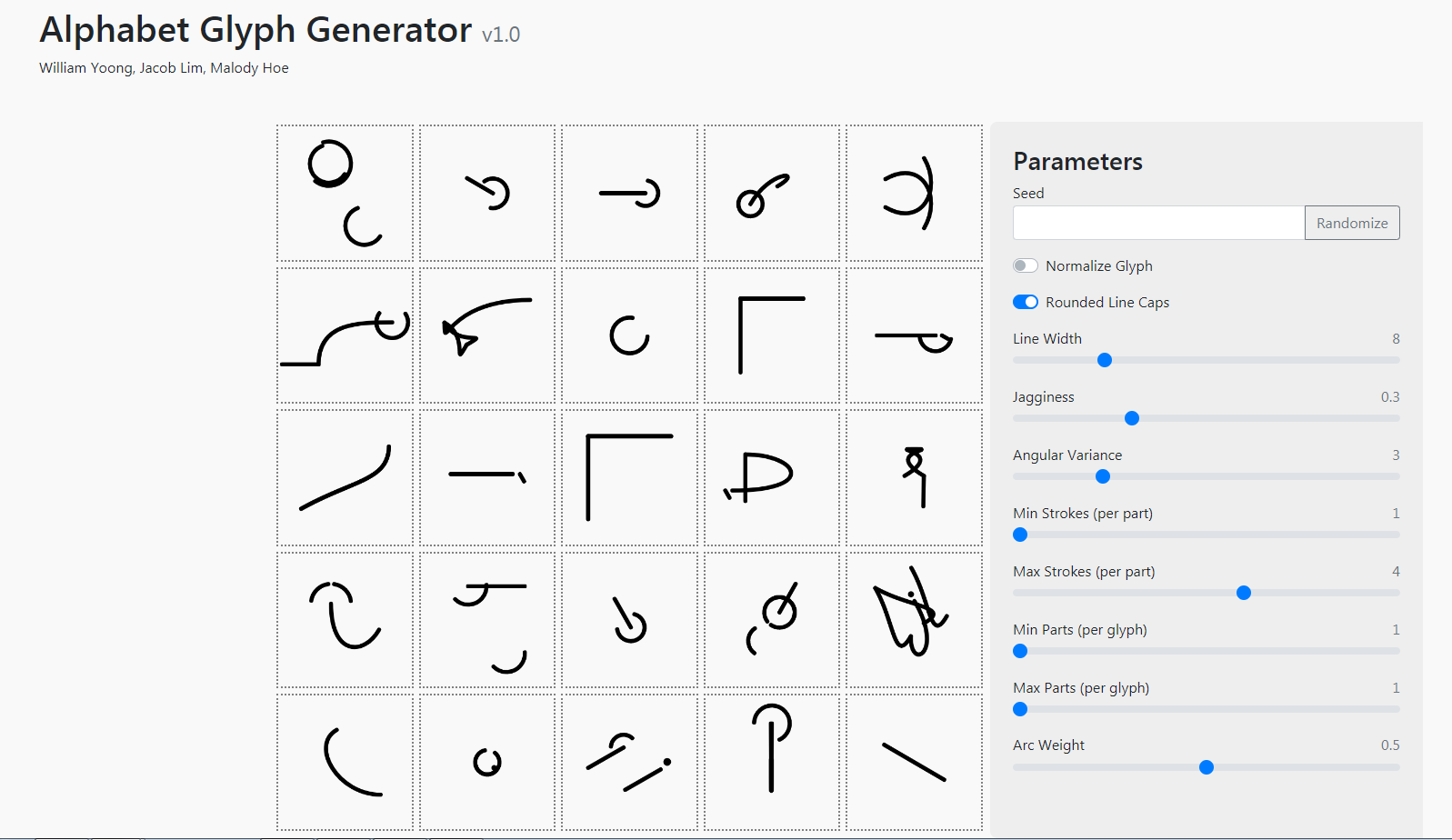


Figure 3 The final product.

# 4 Conclusion

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.