Connor Jones

AN ABSTRACT IMPLEMENTATION OF MEDIA STREAMING

1 INTRODUCTION

This project is an abstract implementation of a typical job that an embedded system would have to do in a real-time video streaming application. Video streaming is the process of downloading and displaying portions of a video without the need for the full video to be downloaded. The simulated task is taking in image data through an input device, converting that data into a usable format, then outputting the image to a device in a way that a person can view it. This will all have to be completed within a deadline in order to be ready to receive the next frame of the video. This gives a sense of what needs to be done to design and implement a media streaming application and a real-time system in general. The end goal for this project was to synchronize the transfer of frames and see what the maximum frame rate is for different resolutions using this method of video streaming.

2 BACKGROUND

Researchers began working to devise a solution for streamable media in the 1990s and early 2000s. They began by trying to devise new protocols that would be needed to implement client-server video streaming while maintaining video quality. This was done using a variety of different real-time specific transport protocols but the most prevalent was based on UDP and was called the Real-Time Transport Protocol or RTP [1]. This protocol allowed real-time systems to have better information and control over the timing of packets. This laid out the fundamentals for video streaming but is ultimately no longer utilized now due to issues over its widespread use and difficulty for users [2].

It became apparent that for video streaming to be accessible it would need to use a protocol that was already well established and available on every system. The clear choice was HTTP so that video streaming could be a normal part of using the Web. This method was developed in 2010 and was titled Dynamic Adaptive Streaming over HTTP or DASH [3]. This provided a basis of what was wanted for modern video streaming. This is the method that is used by major streaming services such as Netflix, Youtube, Hulu, and more [2].

The work in video streaming was not only in the methods of delivery to the user but also in the format and way the user's machine and server processes that information. The two major focuses of that research were scalability and reliability. Works described the best file format for streaming video. It is important for the file to be able to work with different levels of resolution, frame rate and count, and redundancies [4]. This is an important idea to draw inspiration from in this project as there will be multiple resolutions and frame rates that will be tested. It would be impractical to devise a new method of dealing with images for each frame rate and resolution.

Video stream research in reliability for client and server machines focused on scheduling how each packet would be output to the user. This is especially important during times where the server is near peak load and when the channel to the user is unstable and causes occasional packet loss. These scheduling algorithms revolve around the idea that not all packets must be received for the user to experience the video as intended, if the number of packets dropped is minimized then it is possible that the packets that are dropped can be strategically chosen in order to maximize the fluidity of the video stream. One solution uses a priority based earliest deadline first method to send only the packets that must be sent, or resent, during busy times [5]. Other methods attempt to bound the amount of dropped packets over a period of time during peak usage times. One method uses a custom scheduling algorithm named the Modified Proportional Share Scheduling Algorithm that prioritizes based on the period, computation time, number of tasks, and time between drops [6]. Another method uses the earliest deadline first scheduling method but applies a window and determines how many packets in that window are able to be dropped and uses that to determine the schedulability of jobs [7].

As streaming video has been researched many solutions have been found and implemented today. Modern research mainly focuses on the real-time analysis of video streams. This is mainly done through machine learning for a variety of different purposes such as quality assurance [8], traffic analysis and rule enforcement [9], and more. A smaller area of research that is more closely related to this project is the reliable streaming of high frame rate video in networks that are unreliable. These networks include mobile networks [10] and wireless networks [11]. This project differs as it is not trying to optimize reliability in an unstable environment but rather optimize framerate over a stable connection. This knowledge will be relevant if networks in the future become more stable or if you are streaming video from a local reliable server.

3 Methodology / Metrics

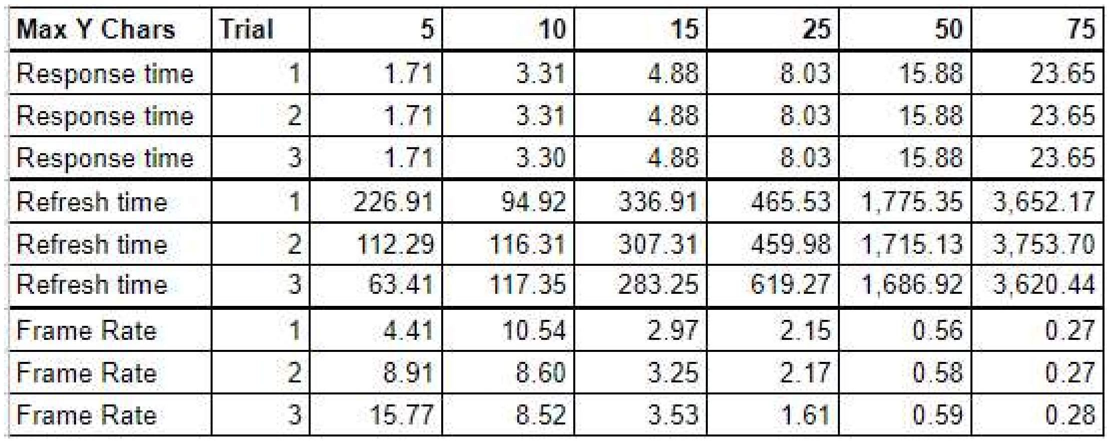
My semester project is an implementation project in which a microcontroller is responsible for taking in a set of values representing a grayscale image and will output ascii art of that image. This imitates what a media embedded real time system would possibly have to do: take in image data, convert it to a different format, and output it to a device, all within a reasonable time in order to receive the next image data. Since there are not a large number of tasks and response time is being minimized to achieve the highest frame rate, a fixed priority scheduler is used. This is because it is the option with the lowest overhead and the system is simple and repetitive enough that determining priorities is not overly difficult. UART is being used for both input and output because it is easy to implement and the lab already contains the materials necessary. All communication done via UART is with a baud rate of 115200. It also learns from the downfall of RTP by using a protocol that is well established and widely used. Unlike many of the current video streaming methods and those described above this implementation is only going to be sending one line of one frame per packet rather than a short bit of the video. This is because the goal of this project is to maximize the response time and therefore frame rate rather than reliability due to a sometimes slow or unstable connection. It is faster to process only a small amount at a time and output it than to take in a larger number of frames and output them over time.

This is implemented with the following method. First the python script takes in a GIF (Graphical Interchange Format) [12] that is split up into individual frames. Those frames are converted to grayscale pixel values via a python script and stored in a two dimensional array, one dimension for the frame and one dimension of grayscale values. The grayscale values represent a block of pixels that will be represented by a single ascii character. The amount of pixels determined by the max characters in the Y direction, which is a user defined variable. These settings are then sent to the microcontroller. The grayscale values for one line of characters in the X direction are then sent to the microcontroller as well. Whenever the microcontroller receives data from the python server it raises an interrupt flag to signal the values have arrived. The callback function then determines the task that will be added to the queue and that task's respective flag is raised. The settings task is simple and will save the settings for later use. The line received task converts the grayscale characters to ascii based on their intensity using the following ramp: " .:-=+*#%@". Those characters are then transmitted back to the connected PC and displayed. The line received task then checks if this is the last line of the frame and if it is it returns the cursor to the top of the image to reset for the next frame.

4 Results and Discussion

The entire process described above was timed using one of the on board timers to measure refresh time, frame rate, and the response time. Refresh time is the time that it takes to completely print a new frame to the screen. Frame rate is the inverse of refresh time, which is easier to conceptualize [13]. Response time is the time it takes the microcontroller to take in the data from the UART, convert it, then send it back to the server for display. Different resolutions have been tested to determine how image size affects total time from grayscale input to ascii. This information was used to determine the maximum frame rate possible at different resolutions. The testing results can be shown in the table below. An image of the last frame of each tested resolution can be viewed in the appendix.

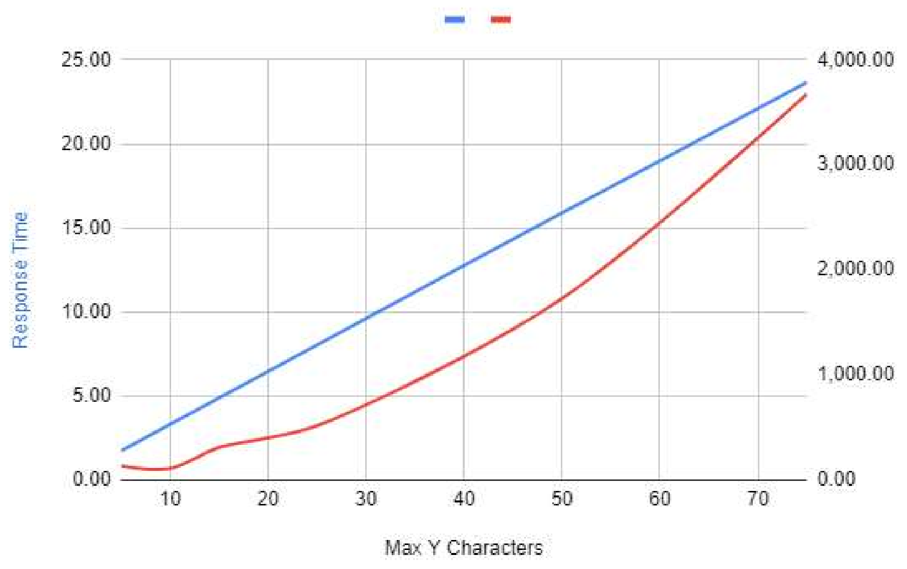
Table 1: Timing Data at a Variety of Resolutions



Max Y Chars	Trial	5	10	15	25	50	75
Response time	1	1.71	3.31	4.88	8.03	15.88	23.65
Response time	2	1.71	3.31	4.88	8.03	15.88	23.65
Response time	3	1.71	3.30	4.88	8.03	15.88	23.65
Refresh time	1	226.91	94.92	336.91	465.53	1,775.35	3,652.17
Refresh time	2	112.29	116.31	307.31	459.98	1,715.13	3,753.70
Refresh time	3	63.41	117.35	283.25	619.27	1,686.92	3,620.44
Frame Rate	1	4.41	10.54	2.97	2.15	0.56	0.27
Frame Rate	2	8.91	8.60	3.25	2.17	0.58	0.27
Frame Rate	3	15.77	8.52	3.53	1.61	0.59	0.28

Timing rates are lower than originally expected because this did not seem like a very incisive task for a microcontroller; however, I was not considering the speed of UART. The UART had a baud rate of 115200. That equates to about 0.1152 Mbps, that is a much slower transfer rate than what is typically used in a streaming application. For reference the average internet speed is 108.4 Mbps [14], which is about 1000 times faster than the UART. Another interesting result was the jitter present in the refresh time, and therefore framerate, but not in the response rate. This is due to the consistency of the microcontroller and the inconsistency of the python server that the microcontroller communicates with. The response time only accounts for the operation of the microcontroller and is therefore very consistent and repeatable. The frame rate accounts for the timing of the whole system and because python is a much higher level program running on a high level operating system it has less control over computer hardware and the timing is less consistent [15].One key feature in the results that may be difficult to observe in the table but is quite easy to see in the chart below is that the response time increases linearly with the max amount of Y characters but the frame rate increases exponentially. This is because response time only accounts for one line of the ascii picture which only increases by $n * r$ characters whenever you increase the number of Y characters by n with an aspect ratio of r. The refresh rate, however, increases by $r * (2kn + n^2)$ where r is still the aspect ratio and k is the previous max Y characters.

Image 1: Response and Refresh Time vs. Max Y Characters



5 Conclusion

This project was able to achieve the goals set out for it. It is able to fully convert and display a gif file in ascii characters using an abstracted streaming setup. There are, however, improvements that could be made to increase the performance and consistency. First, a higher transfer rate would increase the frame rate drastically. The transition between the server and client through UART is the bottleneck in this system. If that speed was closer to the speed of an average internet connection that would no longer be the case. A more consistent server would help reduce jitte, this could be achieved by running the server directly on another microcontroller or using a simpler operating system. If it is not possible to get a more consistent server it may help to reduce jitter by using a buffer in order to have data more readily available to convert to ascii and send as opposed to the current method of having to receive data at every conversion. While this project did not produce any ground breaking improvements to media streaming, it gave me an insight into media streaming and real time systems in general. It allowed me to work closer with microcontrollers and applied core concepts from the class such as: scheduling, through the microcontroller interrupts and flags, resource sharing, through managing the UART with a locking mechanism, and periodic and aperiodic tasks, through the various tasks that was the microcontroller was responsible for.

6 References

[1] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “Rfc3550: Rtp: A transport protocol for real-time applications,” 2003.

[2] B. Li, Z. Wang, J. Liu, and W. Zhu, “Two decades of internet video streaming: A retrospective view,” vol. 9, no. 1s, oct 2013. [Online]. Available: https://doi.org/10.1145/2505805

[3] T. Stockhammer, “Dynamic adaptive streaming over http –: Standards and design principles,” in Proceedings of the Second Annual ACM Conference on Multimedia Systems, ser. MMSys ’11. New York, NY, USA: Association for Computing Machinery, 2011, p. 133–144. [Online]. Available: https://doi.org/10.1145/1943552.1943572

[4] P. Amon, T. Rathgen, and D. Singer, “File format for scalable video coding,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 17, no. 9, pp. 1174–1185, 2007.

[5] K. Gao, W. Gao, S. He, P. Gao, and Y. Zhang, “Real-time scheduling on scalable media stream delivery,” in 2003 IEEE International Symposium on Circuits and Systems (ISCAS), vol. 2, 2003, pp. II–II.

[6] M. Yoo, B. Ahn, D. Lee, and H. Kim, “A new real-time scheduling algorithm for continuous media tasks,” in 2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (IEEE Cat. No.01CH37233), vol. 2, 2001, pp. 417–420 vol.2.

[7] R. West, Y. Zhang, K. Schwan, and C. Poellabauer, “Dynamic window-constrained scheduling of real-time streams in media servers,” IEEE Transactions on Computers, vol. 53, no. 6, pp. 744–759, 2004.1

[8] M. Seufert, P. Casas, N. Wehner, L. Gang, and K. Li, “Stream-based machine learning for real-time qoe analysis of encrypted video streaming traffic,” in 2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), 2019, pp. 76–81.

[9] A. S ̧enta ̧s, S. Kul, and A. Sayar, “Real-time traffic rules infringing determination over the video stream: Wrong way and clearway violation detection,” in 2019 International Artificial Intelligence and Data Processing Symposium (IDAP), 2019, pp. 1–4.

[10] J. Wu, B. Cheng, M. Wang, and J. Chen, “Delivering high-frame-rate video to mobile devices in heterogeneous wireless networks,” IEEE Transactions on Communications, vol. 64, no. 11, pp. 4800–4816, 2016.

[11] J. Wu, C. Yuen, M. Wang, J. Chen, and C. W. Chen, “Tcp-oriented raptor coding for high-frame-rate video transmission over wireless networks,” IEEE Journal on Selected Areas in Communications, vol. 34, no. 8, pp. 2231–2246, 2016.

[12] K. Iqbal, “Gif - image file format,” Sep 2019. [Online]. Available: https://docs.fileformat.com/image/gif/

[13] Intel, “What is refresh rate and why is it important?” [Online]. Available: https://www.intel.com/content/www/us/en/gaming/resources/highest-refresh-rate-gaming.html

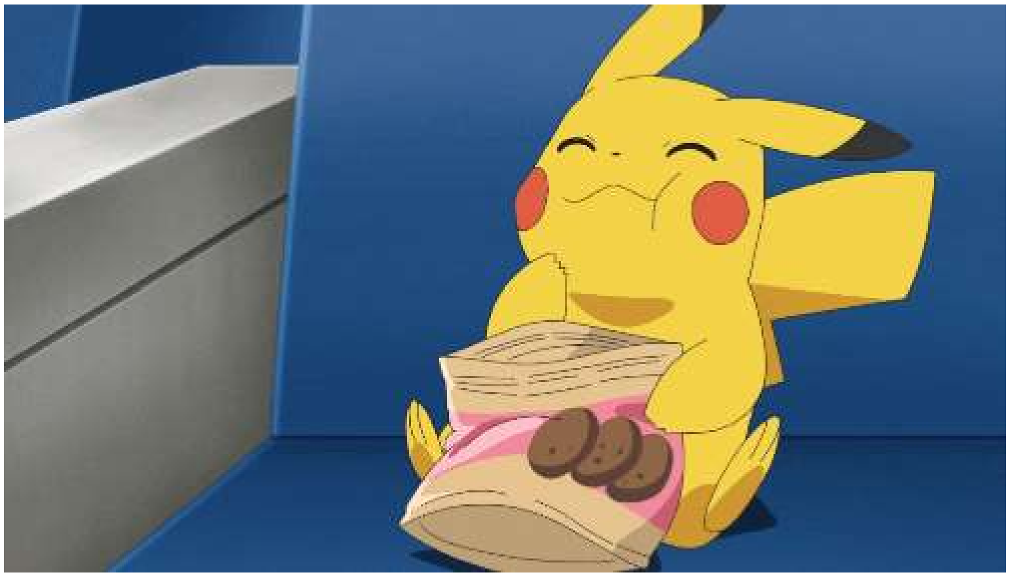
[14] “Average internet speed by state 2022.” [Online]. Available: https://worldpopulationreview.com/state-rankings/average-internet-speed-by-state

[15] Prasanna, “Advantages and disadvantages of high-level language: Pros and cons of high-level language, benefits and drawbacks,” Mar 2022. [Online]. Available: https://www.aplustopper.com/advantages-and-disadvantages-of-high-level-language/

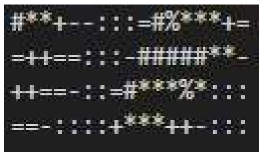
7 Appendix

The code used for this project is included with the project submission. Below are different resolutions of the final frame produced by the microcontroller.

Reference Frame:

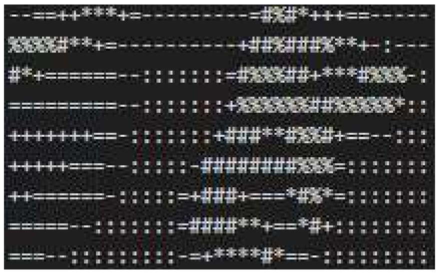


5 Max Y Characters:



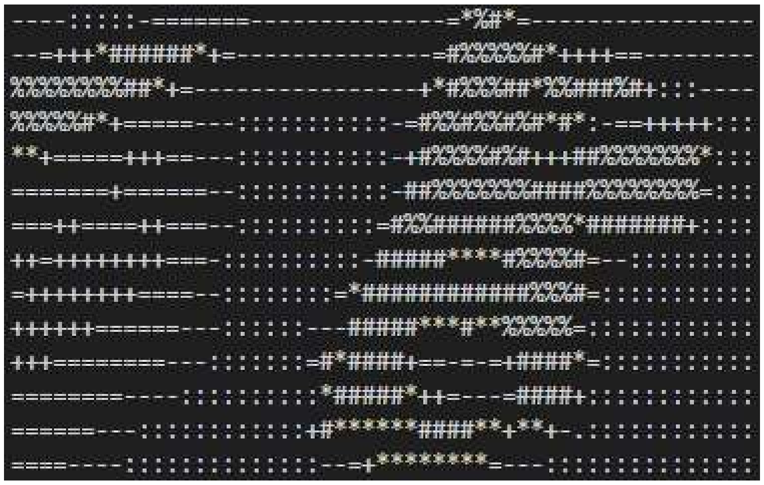
```
#**+--:::=#%***+=
=++==:::-#####**-
++==-::=#***%*:::
==-::::+***++-:::
```

10 Max Y Characters:



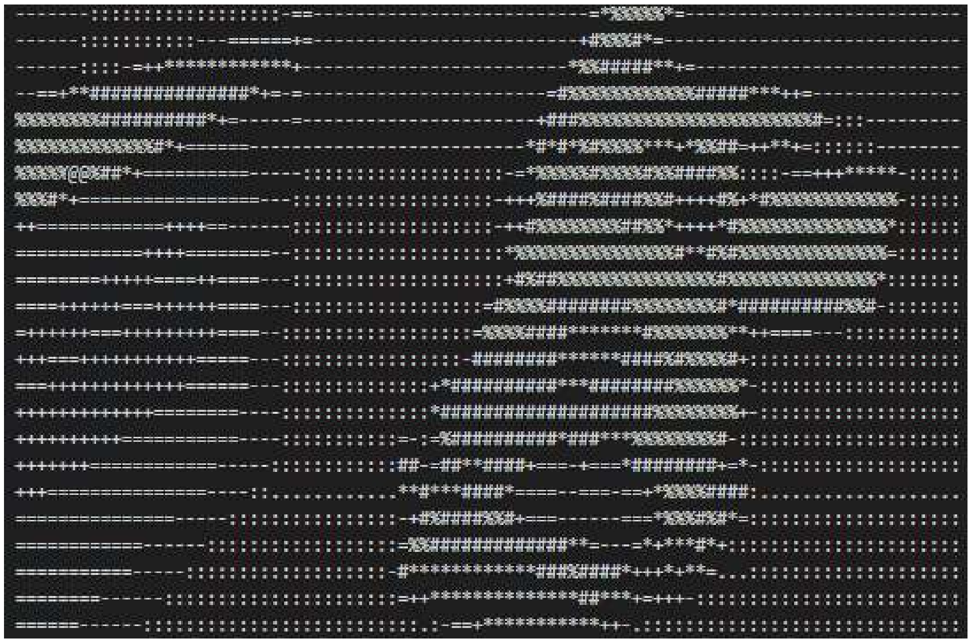
```
--==++***+=---------=#%#*+++==-----
%%%%#**+=----------+##%###%**+-:---
#*+======--:::::::=#%%%##+***#%%%-:
=========--:::::::+%%%%%%##%%%%%*::
+++++++==-:::::::+###**#%%#+==--:::
+++++===--:::::-#########%%%=::::::
++======-:::::=+###+===*#%*=:::::::
=====--:::::::=####**+==*#+::::::::
===--:::::::::-=+****#*==-:::::::::
```

15 Max Y Characters:



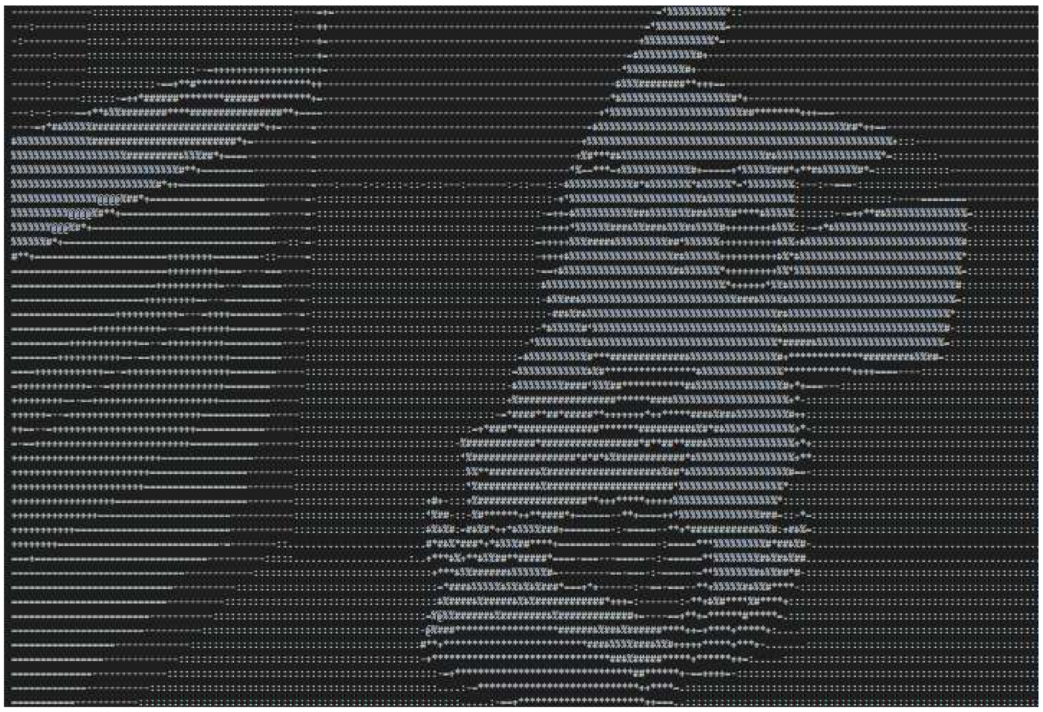
```
----:::::-========--------------=*%#*=----------------
--=+++*######*+=---------------=#%%%%%#*++++==--------
%%%%%%%%##*+=----------------+*#%%%##*%%###%#+:::----
%%%%%#*+=====---:::::::::::-=#%%#%%#%#*#*:-==+++++:::
**+=====+++==---:::::::::::-+#%%%%#%#+++##%%%%%%%%*:::
=======+======--:::::::::::-##%%%%%%%####%%%%%%%%%=:::
===++====++===--::::::::::=#%%#######%%%%*########+:::
++=++++++++===-::::::::::-#####****#%%%%#=--::::::::
=++++++++====--::::::::=*#############%%%#=::::::::::
++++++======---::::::---#####***#**%%%%%=::::::::::
+++=========---:::::::=#*#####+==-=-=+####*=::::::::::
========----::::::::::*######*++=---=####+:::::::::::
======---:::::::::::+#******####**+**+-.::::::::::::
====----::::::::::::--=+*********=---::::::::::::::
```

25 Max Y Characters:



```
-------::::::::::::::::::::-==-----------------------=*%%%%%*=------------------------
------::::::::::::---======+=-----------------------+#%%%#*=--------------------------
------::::-=++**************+-----------------------*%%######**+=-----------------------
--==+**##################*+=-=--------------------=#%%%%%%%%%%%%%%######***++=-------------
%%%%%%%%%###########*+=------=--------------------+###%%%%%%%%%%%%%%%%%%%%%%%%#=:::---------
%%%%%%%%%%%%%%#*+======-----------------------*#*#*%#%%%%%***+*%%##=++**+=::::::--------
%%%%%@@%##*+==========-----::::::::::::::::::::-=*%%%%%#%%%%%#%%%######%%::::-==+++*****-:::::
%%%#*+==================---::::::::::::::::::::-+++%####%####%%#++++#%+*#%%%%%%%%%%%%%%%-:::::
++============++++==------:::::::::::::::::::::-++#%%%%%%%%%##%%*+++++*#%%%%%%%%%%%%%%%%%*:::::
=============++++=========--::::::::::::::::::::::*%%%%%%%%%%%%%%%%%#**#%#%%%%%%%%%%%%%%%%%=:::::
=========+++++====++=====---::::::::::::::::::::::+#%#%%%%%%%%%%%%%%%%%%%#%%%%%%%%%%%%%%%%*::::::
====++++++===++++++++=====---::::::::::::::::::::-#%%%%#############%%%%%%%%%%#*############%%#-::::::
=++++++===+++++++++=====--::::::::::::::::::=%%%%####*******#%%%%%%%%**++====---:::::::::::
+++===++++++++++++======---:::::::::::::::::-#########******####%#%%%%%#+::::::::::::::::::
===++++++++++++++=======---::::::::::::::+*##########***###########%%%%%%%*-:::::::::::::::::::
+++++++++++++=========----::::::::::::::*##########################%%%%%%%%+-::::::::::::::::::
++++++++++===========----:::::::::::=-:=%###########*###***%%%%%%%%#-::::::::::::::::::
+++++++============-----::::::::::::##--##**####+===-+===*#########+=*-:::::::::::::::::
+++=================----::...........**#***####*====-===-==+*%%%%####:.................
================-----:::::::::::::::-+#%####%%#+===------===*%%%#%#*=:::::::::::::::::::
==============------::::::::::::::::=%%##############**=---=*+***#*+:::::::::::::::::::
===========-----:::::::::::::::::::-#***********###%####*+++*+**=..:::::::::::::::::::
========------::::::::::::::::::::::=++**************##***+=+++-::::::::::::::::::::::::
=======------:::::::::::::::::::::::..:-==+***********++-.::::::::::::::::::::::::::::
```

50 Max Y Characters:



75 Max Y Characters: