Convolutional neural networks. Can they help us solve pathplanning efficiently?

Jesús Villar Méndez

Final Degree Work
Bachelor's Degree in
Video Game Design and Development
Universitat Jaume I

July 1, 2021

Supervised by: Joaquín Torres Sospedra

To my parents, who, on their own, unique, way, have always supported me on this journey we call life.

To my sister, María, for always having an eye on me during this hard process of carrying with the subject of Mobile Device Applications while working on my external internship and moving this project forward, I don't always have a place to thank her and this is the perfect chance.

To my best friend Pablo, for sharing with me a little bit of his joy day after day and making this four years I have been studying far from home worthwhile.

To those who have accompanied me despite my ups and downs here in Castellón.

Acknowledgments

I would like to thank my Final Degree Work supervisor, Joaquín Torres Sospedra, for allowing me to work at my own pace and letting me know how to handle the delivery and presentation of this kind of documents, José Vicente Martí Avilés, coordinator of Final Degree Works, for lending me a helping hand while worried about deadlines and delivery tasks and Francisco Miguel Morales Sánchez, friend of mine, for lending me a bit of his time and computation power while implementing the neural network depicted on this project and using his Drive to allow me sending data set images when my computer couldn't.

Abstract

This document presents the project report of the Video Games Design and Development Degree Final project by Jesús Villar Méndez. It consists on an HTML5 and Javascript program that allows the user to solve randomly generated maps of islands using a path-planning algorithm to solve them and a pix2pix neural network that has been trained with the data set generated by the program. The program also allows file imports in order to train more specific cases and a download function. Besides that, the project also integrates some CSS style modifications that provides a better looking result and makes the user experience far more pleasant.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Contents

The problem that will be assessed on this project document is the viability of the use of convolutional neural networks on path planning problems. This chapter reflects what were the objectives presented in order to start the project as well as the initial mindset I was at when the project development started.

1.1 Work Motivation

Being this project a project related to neural networks the initial motivation was to, at last, after four years on this degree, get to understand at an user level how neural networks work. The idea that I had is that neural networks, in this case, convolutional ones, worked internally with layers of filters that somehow created different results or got to detect some specific traits of an image. It seemed natural to me to think that path planning problems were a specific topic that could be addressed with this kind of filters but I did not know to which stent. That was the birth of this project thesis and knowing that neural network training is costly maybe that cost would be worth if it allows the program to provide results with similar quality but better performance.

1.2 Objectives

In order to get a clear idea about how useful might or might not be the use of convolutional neural networks I have come up with a series of sub objectives that can help us get the necessary resources to put onto test this idea.

1. Develop an application that, starting from an image file previously obtained from a render, gets to solve the path planning problem going from a point A to B in a precise way, not necessarily at execution time.

2. Implement the basics of a convolutional neural network either using a library or writing down the code by oneself.

3. Train that neural network using the data obtained from the application created.

4. Put into practice the results and take conclusions about the do ability and final state of the technique.

1.3 Environment and Initial State

As I pointed earlier, my interest on neural networks gave this project a promising initial state, despite not knowing exactly how they worked it makes an easy and enjoyable task to learn more about the internal working of this technique. In relation to the data set, things were a little bit different, I had notions on how to program for HTML5, JavaScript and CSS but I had never delved that much so it will be interesting to see what I can learn.

1.4 Related subjects

Along the four years of this degree on Video game development and design a few subjects have already touched some topics related to this FDP, these subjects are:

1. VJ1217 - Design and Development of Web Games. Introducing HTML5 and JavaScript programming.

2. VJ1214 - Video Game Consoles and Devices. Introducing low-level programming.

3. VJ1234 - Advance Interaction Techniques. Introducing deeper concept of programming such as ML agents and genetic algorithms. Inspiring to say the least.

4. VJ1231 - Artificial Intelligence. Introducing neural networks for the first time.

5. VJ1212 - Graphics Communication. Introducing Blender and 3D rendering.

6. VJ1216 - 3D Design. Introducing texture use on 3D programs.

7. VJ1209 - 2D Design. Introducing drawing algorithms such as Bresenham's line algorithm.
8. VJ1221 - Computer Graphics. Introducing shader programming and texture programming.
9. VJ1224 - Software Engineering. Introducing project planning.

CHAPTER 2

Planning and resources evaluation

Contents

On this chapter I will expose the original planning of the project in relation to the objectives presented on chapter one, section 1.2 and later it will be explained how some elements of temporal planning got to change in order to deal with my external internship and other subject assessments.

2.1 Planning

The development of this project will be fragmented in three main blocks:

- Previous to training work.
  - Render generation. Using a camera that renders at an specified resolution (256px * 256px at least). Using a render will help me create the basic problems for the data set in an efficient way and will allow me to make more and more images with the least effort possible.
  - Path planning generator. On this block it will be developed a program that should solve the path finding problem in a way that should provide better results than A* can give. This path planning generator, though not strictly necessary, will give me the ability to create a data set with less effort than the one I would need if I got to make all the data set, problems and solutions, all by myself.

- Research and implementation of neural networks. On this block I will enter full on researching convolutional neural networks specifications and balance the pros and cons of creating my own neural network or using a library that can implement it for me. This will be the hardest part of the development process of this project but it will provide me the insight necessary to grasp the knowledge I will need to make conclusions. Besides that, in this block I will also train the neural network with the data set created and see the do ability of a small demo using the training results.

- Conclusions. This is the most important part of the project, I will get to write down all the conclusions obtained from the development of neural networks and give a verdict on why using convolutional neural networks is a good or bad idea when talking about path planning.

2.1.1 Expected results

Taking into account the huge amount of variables depicted on this planning it is plausible to expect from this research:

1. A data set generator.

2. A training data set.

3. A set of images generated by the neural network.

4. A clear idea or concept that will allow us to deduce the feasibility of using this technique.

2.2 Resource Evaluation

In order to develop the full project I will make use of these Technologies and tools:

1. MSI GT72VR 7RD Dominator (7th gen CPU, 16GB RAM, GTX 1060 GPU). My day to day use PC.

2. Visual Studio Code. The simplest and cleanest way to code, easy and comfortable thanks to the use of extensions it will help create the path finding program with ease.

3. HTML5 and JavaScript. After thinking a while about which programming language could I use to create a program with an interface I thought about VJ1217 - Design and Development of Web Games and the experience I gained with HTML and JavaScript using the canvas and after a more detailed understanding of the properties that a canvas has I think it will suit me. I also took a look at processing, a java-based language more focused on image treatment but, being the work

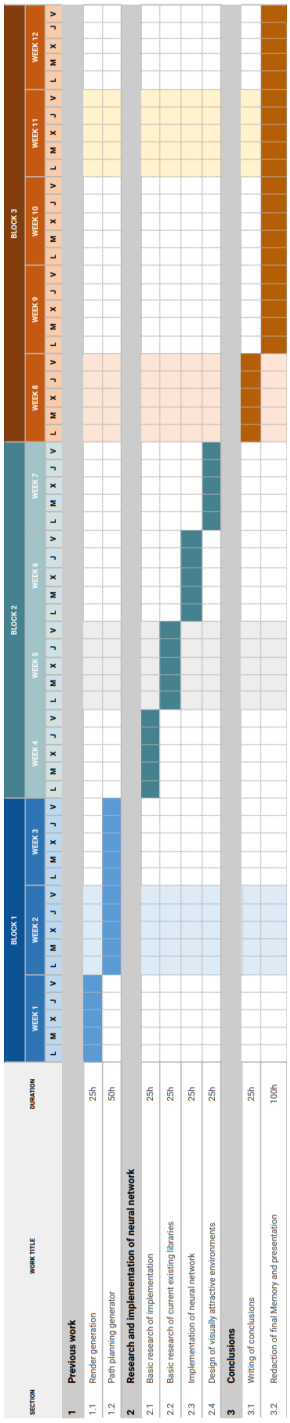


Figure 2.1: Gantt diagram

that I will do here focused on block 1 I considered learning another programming language just to get an easier walk on this part I thought it was not worth a shot, that being said, if at the end HTML5 and JavaScript turn out to be harder than expected I have my eye on processing.

4. Blender. Having used Blender for 3 of the four years on this degree has given me the confidence to know it will suit me well when developing the renders of the path planning problems. Having the knowledge I have on nodes and shaders I think it will be the perfect way to enhance productivity and obtain a deeper understanding of this tool I love so much.

5. Unity. As the game engine we have learned more profoundly through the degree it is the wisest choice I can ever make if I want to try out and test the path planning neural network.

6. Python, C#? Not knowing if I am going to develop the neural network from scratch or taking it from a library I do not know which language I will get to use. That being said, I know it is common for artificial intelligence scripts to be programmed on python, so it is worth taking a closer look on it when the time comes.

CHAPTER 3

Project analysis and design

Contents

This chapter presents the requirements analysis, design and architecture of the proposed work, as well as its interface design.

3.1 Requirement Analysis

To carry out a job, it is necessary to perform a preliminary analysis of its requirements. In this section it will be detailed the functional and non-functional requirements of the presented work.

3.1.1 Functional Requirements

The functional requirements we look forward to while developing this application are:

The proposed system should allow the use of images created on Blender, give the user the ability to place the initial and final point so the program can solve it and let you download the images. This functional requirements are shown at tables 3.1, 3.2, 3.3 and 3.4.

Input:	Image file
Output:	Image displayed on screen
The user can import its own images, following certain rules in order to train the neural network with more specific tasks and in order to get more concrete results.	

Table 3.1: Functional requirement «REQ1. Image manage»

Input:	User click on screen
Output:	Set of initial and end point
The user clicks on the screen and gets to place on the map the initial and end points that the path planning algorithm will solve in order to train the neural network.	

Table 3.2: Functional requirement «REQ2. Drawing function»

Input:	User clicks on "Download"
Output:	PNG image requested
The user clicks on the download button and gets in return the PNG of the map displayed on canvas.	

Table 3.3: Functional requirement «REQ3. Download function»

Input:	User clicks on "Solve"
Output:	Canvas display solution requested
The user clicks on the solve button and gets in return the path from initial point to end point displayed on canvas.	

Table 3.4: Functional requirement «REQ4. Solve function»

3.1.2 Non-functional Requirements

As for non-functional requirements it is, from a design perspective, important:

1. The system will be easy to use.
2. The interface will be clean and responsive.
3. Maps will consist on islands and they will be aesthetically pleasant.

3.2 System Design

In order to visualize how the program flow will work there is this activity diagram: (See figure 3.1)

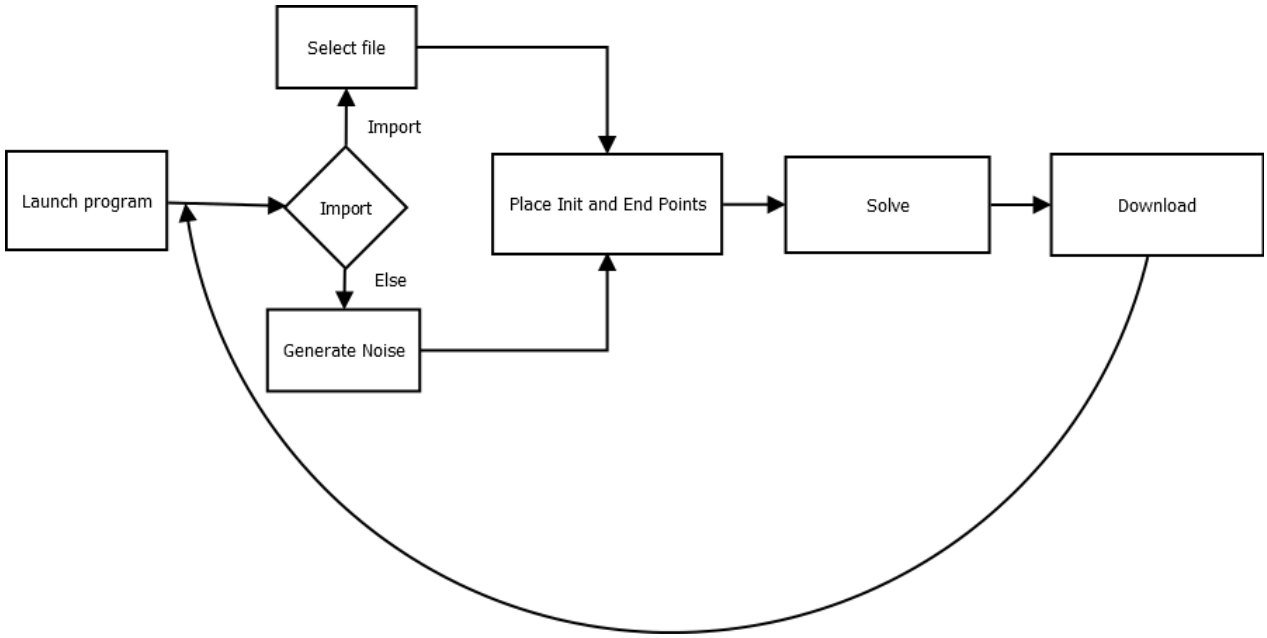


Figure 3.1: Activity Diagram

As it can be read on the figure 3.1 the user can choose between importing an image from his PC or randomly generate one using a noise generator function, then place the points on the interface, solve the path planning problem and then download the image of the path and the initial premise.

3.3 System Architecture

This program, being created on HTML5 and JavaScript, can be run on almost any system that can deal any web browser HTML5-compliant. Google Chrome or Firefox

should work just fine. Besides that it might be helpful to use an 5th generation or greater CPU in order to run the program smoothly.

3.4 Interface Design

The interface, as described on the non-functional requirements 3.1.2, has to be clean and responsive. In order to do so it will be implemented using CSS and provide the user a list of buttons that invoke the functionalities required 3.1.1. (See figure 3.2)

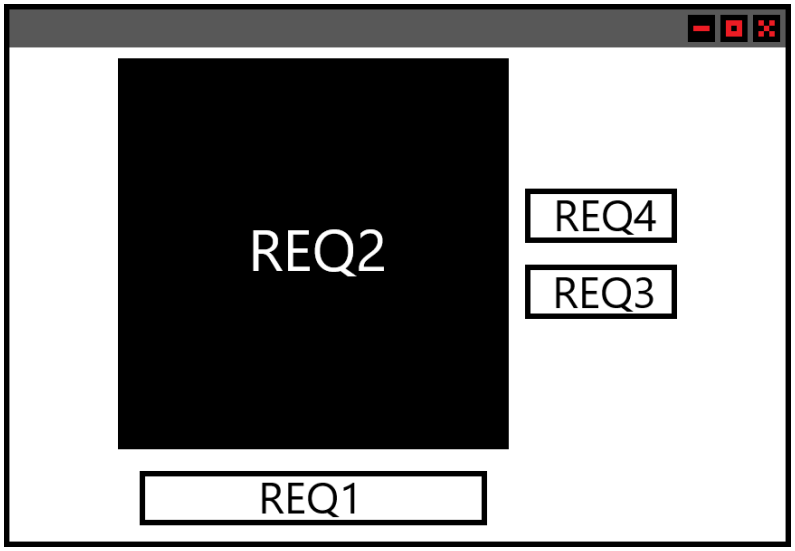


Figure 3.2: UI mock-up

Chapter 4

Work Development and Results

Contents

On this chapter it will be detailed how this project has been developed and its results.

4.1 Introduction

On this section, it will be introduced how the data set problem has been addressed, the ins and outs during its development and how the neural network got to be chosen and implemented.

4.2 Data set development

4.2.1 Data set description

The final data set presented is conformed by a set of 206 images, later modified to increase the information to 824 images via rotations and flips. Each of this images got to be transformed to JPEG, losing some quality but on essence they are based on a noise texture, layered in order to create islands, the input images have two red points signaling the place where the path must begin and end and the target images are just the same but with magenta strokes signaling the path the neural network should draw.

4.2.2 Blender rendering

Following the planning, the project development started and the first step was to open Blender and try out and generate some randomly generated islands. In order to do so, it has been implemented a blender material that, using the object texture coordinate and a mapping node that connect to the vector input of a Noise Texture node, generates a noise with 0.4 scale, 2.0 detail and 0.0 distortion which is deprived of any color by using a Color Ramp node.(See figure 4.1) Then this material is applied to a plane with a camera pointing at it with no anti aliasing and a resolution of 256x256 pixels in order to render it.

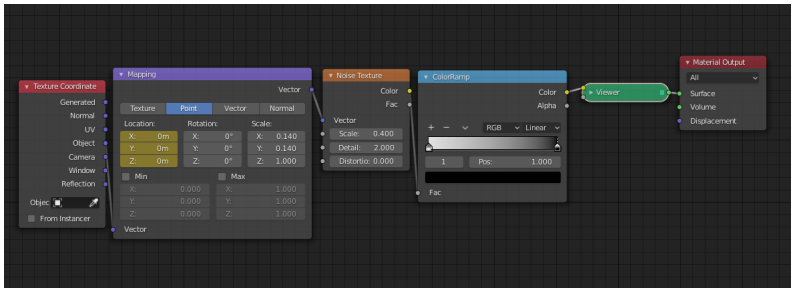


Figure 4.1: Noise material

Once the render is done. By using the image Compositer that Blender lends us and applying to it another Color Ramp Node, the values of black and white the noise has can be painted by layers changing the image into a plain colors islands' map or a topology map with more detail. (See figures 4.2, 4.3)

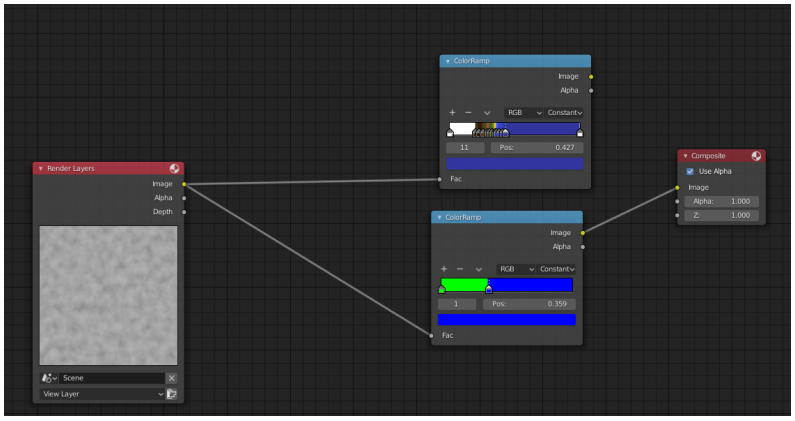
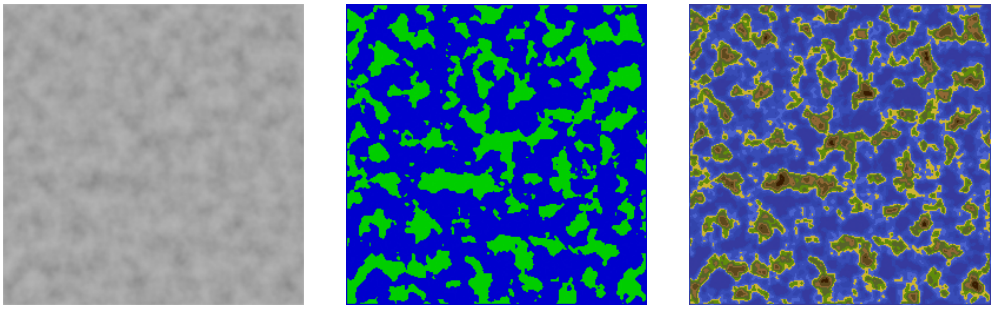


Figure 4.2: Compositer

This maps were at first very helpful during the first steps of development but they were not perfect. Despite adjusting the Compositer to draw the maps using pure RGB colors (Red [255,0,0], Green[0,255,0], Blue[0,0,255]) the resulting image fidelity was not



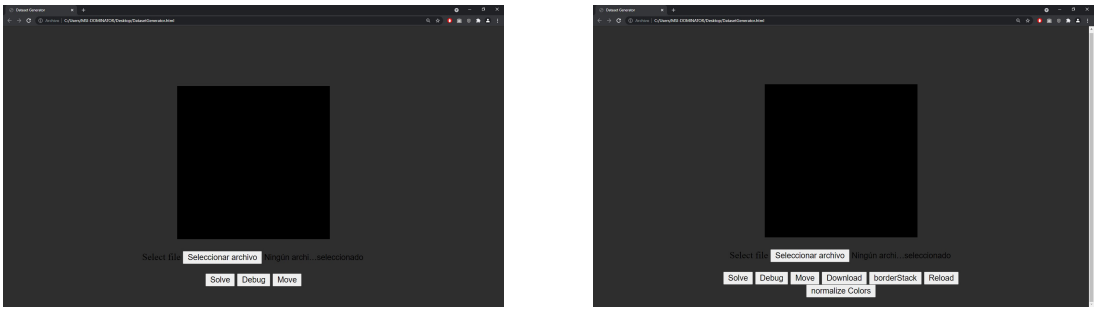
(a) Noise map (b) Plain colors map (c) Beautified map

Figure 4.3: Blender noise islands generation

correct, so, in the HTML program it was required a normalizing method to properly make the algorithm work.

4.2.3 HTML5 first steps

Carrying this work on, the time for using JavaScript was near. It has already been created a HTML5 basic interface that included a few buttons, a canvas[2], an input file section[10] and a hidden image which display was "None" that will storage the imported images before it was displayed into the canvas. (See figure 4.4)



(a) Initial HTML5 program state (b) Debug features

Figure 4.4: HTML5 initial display

This visual would eventually evolve into a more refined one. (See figures 4.11, 4.10)

4.2.4 Class Point

First of all it was an urgent need to implement a point class to help the drawing process way more intuitive[4]. This point class implemented

- Its own constructor based on x,y position on canvas.

- A getColor method which differentiates between, black, white, red, green, blue, magenta and yellow. Any color that is not into that list would be defined as undefined.

- A getColorRGBA method which returns an array with the RGBA values of the color on the point this method is called on.

- A setColor method that paints the point into a given RGBA color code.

- A nearTo method which returns a Boolean that confirms if the point where this method is called on is near to a certain color.

- A countNear method that counts how many points of certain color are near the point the method is called on.

- A tangentOfColor method that returns the points of certain color that are in touch with the point the method is called on.

- A pointsNearTo method that returns the points of certain color that are in nearby the point the method is called on.

- A directionTo method that returns an array with the direction in relation to the point the method is called on to a certain color.

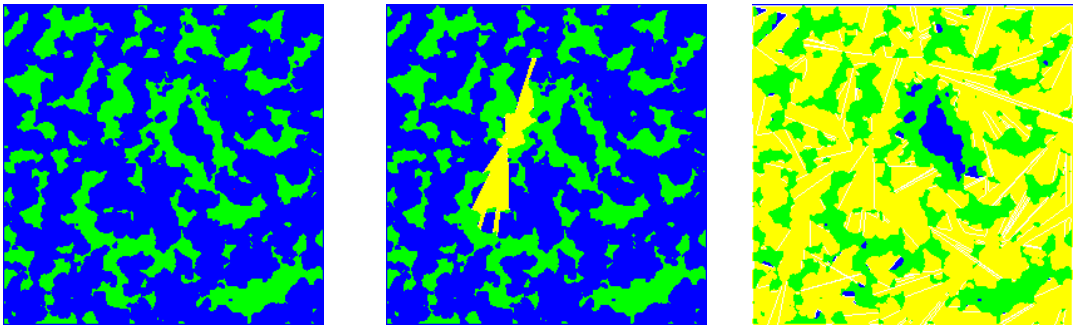
- A next method, used on loops to follow certain order.

- A last method, used on loops to follow certain order.

- A getDirectionVector method that returns the normalized direction vector based on its last position.

4.2.5 Path finding algorithm first approach

It was the time for implementing the path finding algorithm the neural network will have to decode and I thought about finding the shortest path between two points. A* was my first thought. It has been developed on subjects like VJ1231 Artificial Intelligence, however I started to think that a way point based algorithm will suit this topic best and with that in mind and taking into account that there is no shortest path than a straight line between two points a line generation algorithm must be implemented. And so it got done, Bresenham's line algorithm[18] got implemented and points of interest were looked for. After some tests and profuse thinking some tries were made. (See figure 4.5)

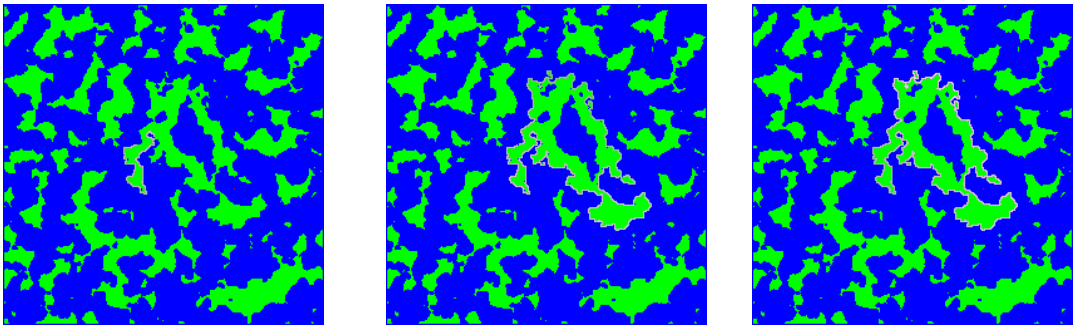


(a) Initial map state (b) Initial iteration (c) Final iteration

Figure 4.5: First approach to path finding algorithm

4.2.6 Path finding algorithm final approach

This initial steps were not quite promising but something was certain, the most important points where placed right into the island borders, in the limit where you can trace a line without crossing land and where you cannot do so. There had to be a way to find those points and here the next idea was met. If a straight line between initial point and final point was drawn the island where the line collides should get bordered and, on that borders, it must be found the points that delimit the frontier where a line can be drawn without colliding any land. Taking advantage on the fact that the map resolution is 256x256 pixels and the color depth that canvas use is of 255 on four channels of color, the color of the point can be use as a data structure with the methods next and last interpreting blue and alpha as x and y for next point coordinates and red and green as the x and y for last point coordinates. By doing so the results of the borders were unidirectional paths that ended where no more points can be drawn. (See figure 4.6)



(a) Initial approach to border island (b) Second approach to border island (c) Final approach to border island

Figure 4.6: Border island approaches

This method was interesting but it had a major trouble. There were loops that could not be solved. On certain points the border of the island generates a strait of one pixel

of width so if the border generator enters that loop (thing that is unlikely to happen without certain rules) it cannot keep on propagating and the border is not fully drawn. To solve the problem the border drawing algorithm must be improved. The rule finally followed by the algorithm is:

1. If the previous point is near a green point try to turn on that direction.
2. If the current point is near a green point try to turn on that direction.
3. If none of the previous possibilities are correct try instead turn to the opposite direction.
4. Try to go forward.
5. Try to go to the opposite direction of the first intent.

Following this instructions the algorithm will try to enter each and every nook. When the pointer cannot go any deeper the variable redrawneeded is set to true and the recursion goes back and the algorithm then follows its next desired direction completing the loop completely. (See figure 4.7)

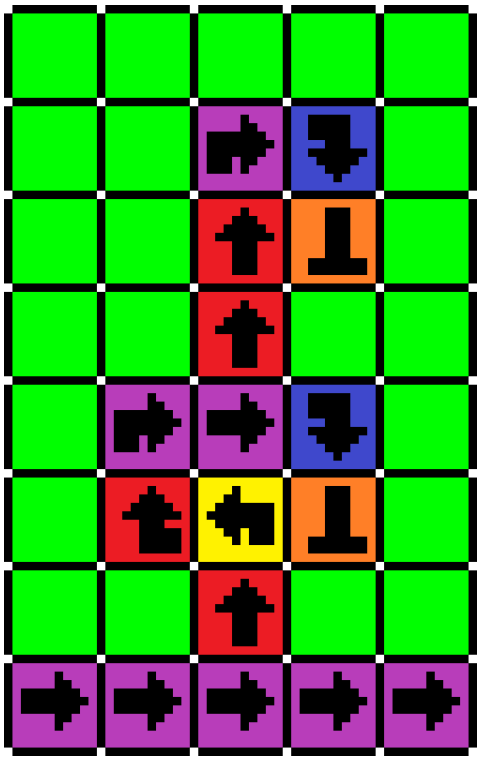


Figure 4.7: Border follow algorithm diagram

With the border drawn there must be created an algorithm that can follow it. Starting from the contact point, the algorithm follows to the next point determined by next()

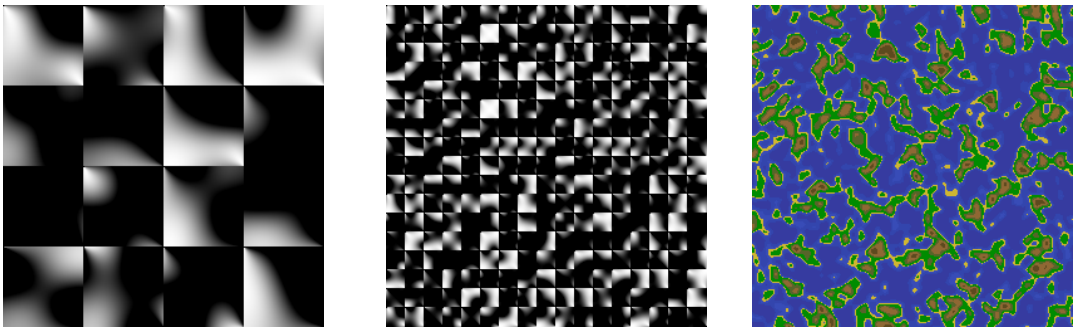
method but also tries to find other direction points that are perpendicular to its direction. If a point with that condition is met the next point is pushed into a stack and the perpendicular point is followed. Once there are no more points to follow, the stack pops the last point saved and so all points of the border are followed.

This method also implements the checks necessaries to see if you can draw a line from the point that draws the line towards the end.

4.2.7 Noise generation

While developing these features I had to take a break from the project if I wanted to pass VJ1229 Mobile Device Applications. On that break I got to the realisation that if noise were to be generated externally it would be so user-demanding to generate all images on blender and import them all one by one so why not implementing my own noise generator? The idea was great but that was not quite well planned, I researched through all noise generation tutorials I could find[13][5][11][7] but the expected results could not be achieved. (See figures 4.8a and 4.8b). I even tried to test some concepts about random[3] noise generation but soon I realised that it was consuming some precious time I had not at that point.

Against my will but knowing that this was the best thing It could be done with the amount of time remaining I searched the internet and found Noise.js[9], a library developed by Joseph Gentle, soon it got implemented on the project combining two simplex noises with random seeds the detailed looked for was found, the normalizeRenderColors function changed to suit correctly the new noise and it was working. (See figure 4.8)



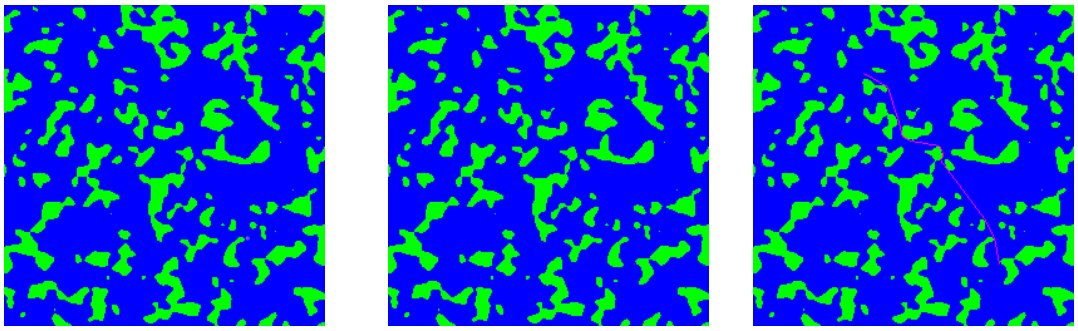
(a) Perlin noise attempt (b) Perlin noise attempt (c) Noise.js result

Figure 4.8: Noise generator approaches

4.2.8 Path finding problems

Unfortunately time was running out and I had to end the data set because too much time had already been spent on its development. Waypoint class was quickly developed and some methods were dedicated to create a tree that would have to grow towards the final point. using the border methods described before. That is where real trouble

began. FollowLoop and border methods had some small issues, border required, due to a specific case, that the only island in green would necessarily be the one who has to get bordered. if that was not the case loops could be made between islands and the algorithm would enter an infinite loop. Solved that issue, if islands were big enough, the recursion stack size got exceeded and the program crashed. That issue could not be solved in time if I wanted to train the data set so in the end I reduced the islands size and left the algorithm halfway the point I wanted it to get. It worked, but not consistently enough. The project needed to keep on going forward so the data set got to be generated with that algorithm, if the neural network was able to learn that it would be able to learn with a similar data set a little bit more carefully crafted. That being said the data set was created. It was composed at first by 103 images for the input and another 103 images for the output. (See figure 4.9)



(a) Normalized image (b) Input (c) Target result

Figure 4.9: Data set composition

4.3 Neural network development

The development of the neural network started at first looking for documentation and searching the best kind of neural network that could help solve this problem.

The first option that appeared after realising that building the neural network from scratch was not a viable option[1] was Brain.js[16][8], a JavaScript library that could allow the project to be fully supported on HTML5 and ease the implementation because it would not be necessary to deeply research other programming languages. However that was not actually the best approach to follow. With further research Google Colab appeared as a training solution. Not only it does not require any computation from the system using it but it also provide a virtual GPU that gets to do all the hard work and it would not be necessary for me to install Conda, numpy or TensorFlow. The tool was chosen but the approach was still not certain.

After a little bit more of research and looking through Two Minute Papers YouTube channel[17] it was there, Pix2Pix[14][15] [12], a Generative Adversarial Network intended to transform an image into another. The paper was available and the descriptions on how

Figure 4.10: Final program unsolved

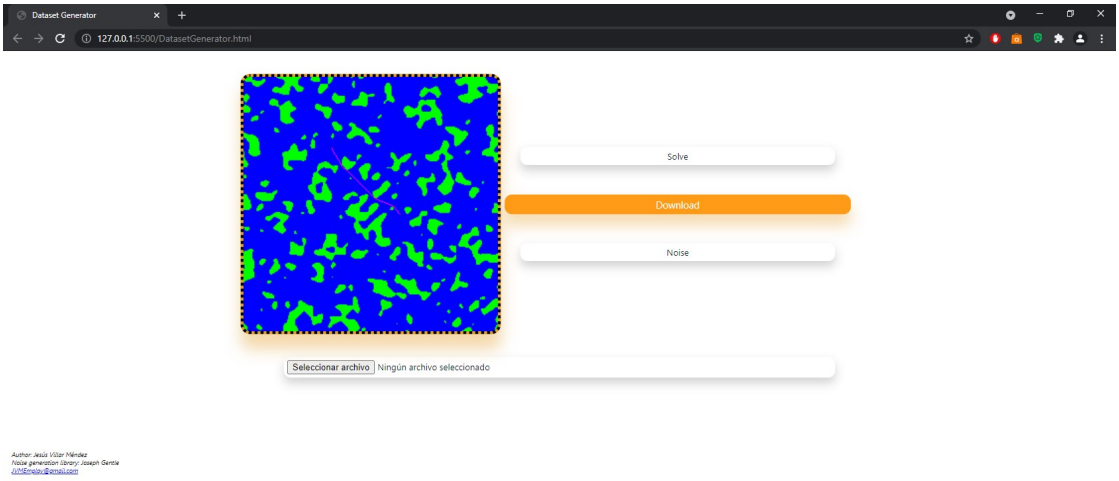


Figure 4.11: Final program solved

the neural network worked was clear. The architecture of this neural network consisted on a generator conformed by an encoder and decoder that, parting from an image, generates another with the data that got discriminated by the encoder and decoder. That generated image is then judged by a discriminator, a system that internally works similarly to the generator but instead of creating a new image it creates a diagram where it is marked how believable the result of the generator is and with that information the results is evaluated and the neural network back propagates this results so the learning process is made.

On his hand the Spanish youtuber Dot CSV made a video tutorial[6] on how to implement this paper[14]. Followed that tutorial and looking into the recommendations made by the paper author about the training data set size was some changes were made. The original data set was composed by 103 images. This images were made following some clear ideas in order to make this work. It was obvious that if each map had only one solution the neural network will not even recognise the starting point and ending point as an important part of the process of generation. With that in mind there were a few solved images from different points on the same map. Despite that the size of the data set was ten times smaller than the one recommended but looking into the random jitter implementation it was clear. If the images were rotated 4 times and mirrored another four times the size of the data set would increase eight times its size. The images were converted to JPEG to avoid troubles with the alpha channel and then the training began.

Google Colab offers 12 hours a day to the users to work with this tool for free. If you want to use better GPUs or extend that time the fee is of 9.99€ a month. This fact was not taking into account the first time the program was sent to train and the process stopped at epoch 73 leaving some inconclusive results. The training got to be repeated once again 12 hours later losing a huge amount of time and it got to epoch 217. And then, after months of work, the results appeared.

4.4 Design choices

During the development of the data set, some design features that were clear got to change. The two most important features that have changed drastically are:

1. Border Finding Algorithm. The border finding algorithm initially proposed looked for accessible areas that would eventually be travelled by using a search tree. However this implementation lacked concreteness, it was useful to know which places on the map should be travelled but not the exact point the path should be printed on in order to be the shortest path. After some updates on the algorithm it got to be disposed so the border drawing with points projection described at subsection 4.2.6 which ended up being more efficient and intuitive.

2. Noise Generation. On the objective section 1.2, while talking about this project functionalities it is specified that render generated images can be solved with the data set generation program. With that in mind the file selector was added to the

project but as time passes It was more and more evident that having the noise generator outside the project was a bad idea, it required so much effort from the user to generate, find, select, place and save the image so in order to make the user experience easier a noise button was added to the project, saving a lot of time in the data set generation but investing a little bit too much on implementing this generator.

4.5 Results

(See figure 4.12)

Taking into account the objectives presented in chapter 1, section 1.2, and the expected results in chapter 2, section 2.1, subsection 2.1.1 on one way or another they have all been met however the results can be improved.

1. Develop an application that, starting from an image file previously obtained from a render, gets to solve the path planning problem going from a point A to B in a precise way, not necessarily at execution time. The data set generator can be improved by implementing A* to the waypoints structure and there are a few bugs that should be taking into account to provide a better user experience. Whatsoever the program was not fully intended for user usage and its main objective was to facilitate the task of creating the data set, objective that has been met.

2. Implement the basics of a convolutional neural network either using a library or writing down the code by oneself. This task has been fully complete.

3. Train that neural network using the data obtained from the application created. This task has been fully complete.

4. Put into practice the results and take conclusions about the do ability and final state of the technique. This task will be completed on the conclusions subsection of this results section.

Even though the objectives have been achieved, the planning process was not followed as conscientiously as I would have like because of the development of other simultaneous projects ending up extending the data set creation process and compressing the time of neural network implementation and memory craft.

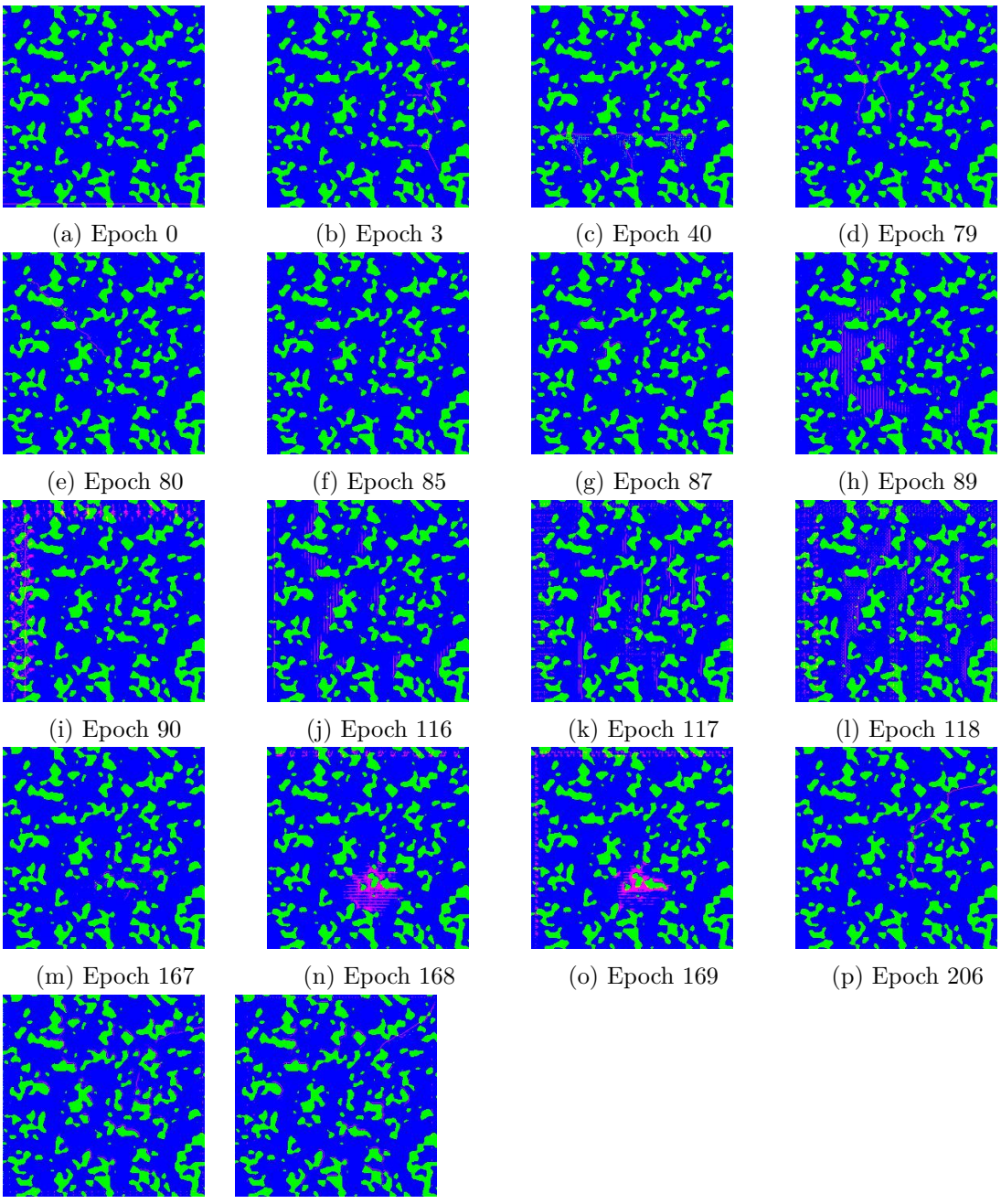
4.5.1 Neural network results analysis

The results the neural network has given allow us to get into conclusions.

First of all, the neural networks started its learning by drawing purple strokes at random and during the learning process the neural network starts recognising island borders efficiently. Basic concepts such as, paths must be drawn on water are efficiently learnt but it does not recognise any relation between the path it has to generate and the red points placed on the map.

The hypothesis I weigh on why does it not work properly seems to point to the size of the initial and ending points which. After being filtered through the generator encoder the information that places the red points is partially lost and so the neural network can learn how to draw paths but cannot successfully paint the right path.

This problem seems to me a design problem so the results are inconclusive. If the hypothesis results to be true, convolutional neural networks can solve path planning problems. That being said, this document cannot prove that idea.



(a) Epoch 0 (b) Epoch 3 (c) Epoch 40 (d) Epoch 79

(e) Epoch 80 (f) Epoch 85 (g) Epoch 87 (h) Epoch 89

(i) Epoch 90 (j) Epoch 116 (k) Epoch 117 (l) Epoch 118

(m) Epoch 167 (n) Epoch 168 (o) Epoch 169 (p) Epoch 206

(q) Epoch 209 (r) Epoch 214

Figure 4.12: Final results collage

Chapter 5

Conclusions and Future Work

Contents

In this chapter, the conclusions of the work, as well as its future extensions are shown.

5.1 Conclusions

To wrap up this FDP there are a few ideas that are stuck on my head. It has been four years since I started studying this career and I have put huge amounts of work and love on projects that have been immensely by other students, the pressure of doing something remarkable without dying at the attempt was critical during the development of this project. Mobile device applications, legal aspects of video games, this project and the external internships at the same time have crushed my soul a little bit, and while I am writing this I, at last, feel some relieve, it is all coming together.

Neural networks have got my attention from time to time, I had a feeling that this project would show impressive results but I feel a little disappointed on how the results could not be conclusive. With this neural network "fiasco" at least I have had an opportunity to improve my ability with JavaScript, HTML5 and CSS.

Despite all the suffering, I have enjoyed this project a lot.

5.2 Future work

Though enjoying this project development, my head needs some rest. I know myself and it is probable that it will come the day when I look back at these days and think "I can fix that", so chances are I will probably continue working on this project. My hypothesis are there on the table, waiting to be proven, it is just a matter of time, the way is to improve the path planning algorithm, augment the size of the red dots and learn more about how neural networks internally work.

Bibliography

[1] 3Blue1Brown. Neural networks playlist. https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1$_6$7000$Dx_ZCJB - 3pi\&ab_channel = 3Blue1Brown$. Accessed : 2021 - 04 - 28.

[2] anonym. Canvas antialias. https://www.xspdf.com/resolution/50518085.html. Accessed: 2021-02-21.

[3] anonym. Math.random(). https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global$_Objects/Math/random. Accessed : 2019 - 04 - 26.

[4] anonym. Pixel manipulation with canvas. https://developer.mozilla.org/es/docs/Web/API/Canvas_API/Tutorial/Pixel$_manipulation_with_canvas. Accessed : 2019 - 02 - 28.

[5] Solving Code. Noise map generator (perlin noise) using javascript and canvas. https://www.youtube.com/watch?v=AZRSkm9UD6o&ab$_channel = SolvingCode. Accessed : 2021 - 04 - 5.

[6] Dot CSV. Generando flores realistas con ia - pix2pix | ia notebook 5. https://www.youtube.com/watch?v=YsrMGcgfETY&ab$_channel = DotCSV. Accessed : 2021 - 06 - 11.

[7] Hugo Elias. Perlin noise. https://web.archive.org/web/20080724063449/http://freespace.virgin.net/hugo.elias/models/m$_perlin.htm. Accessed : 2021 - 04 - 10.

[8] FreeCodeCamp.org. Neural networks with javascript - full course using brain.js. https://www.youtube.com/watch?v=6E6XecoTRVo&ab$_channel = freeCodeCamp.org. Accessed : 2021 - 05 - 4.

[9] Joseph Gentle. Noise.js. https://github.com/josephg/noisejs. Accessed: 2021-04-7.

[10] Thomas Steiner Kayce Basques, Pete LePage. Read files in javascript. https://web.dev/read-files/. Accessed: 2021-02-19.

[11] LaNsHoR. Ruido perlin. https://www.lanshor.com/ruido-perlin/. Accessed: 2021-04-10.

[12] MoPaMo. ml5-library. https://github.com/ml5js/ml5-library. Accessed: 2021-06-10.

[13] Jen Lowe Patricio Gonzalez Vivo. The book of shaders. https://thebookofshaders.com/. Accessed: 2021-03-7.

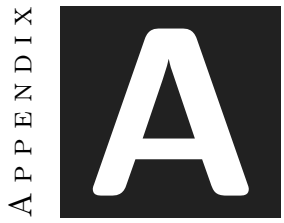
[14] Tinghui Zhou Alexei A. Efros Phillip Isola, Jun-Yan Zhu. Image-to-image translation with conditional adversarial nets. https://phillipi.github.io/pix2pix/. Accessed: 2021-06-11.

[15] Yining Shi. Pix2pix. https://learn.ml5js.org//reference/pix2pix. Accessed: 2021-06-10.

[16] Ringa Tech. Tu primera red neuronal - inteligencia artificial. https://www.youtube.com/watch?v=UNFFLJPW7KQ&ab$_channel = RingaTech. Accessed : 2021 - 05 - 4.$

[17] TwoMinutePapers. Ai makes stunning photos from your drawings (pix2pix) | two minute papers 133. https://www.youtube.com/watch?v=u7kQ5lNfUfg&ab$_channel = TwoMinutePapers. Accessed : 2021 - 06 - 11.$

[18] Wikipedia. Algoritmo de bresenham. https://es.wikipedia.org/wiki/Algoritmo_de_Bresenham. Accessed: 2021-03-8.



Other considerations

This appendix is included to comment some aspects not considered in the rest of the template.

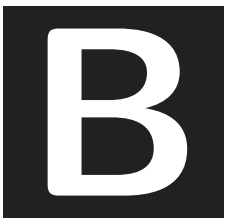
A.1 Bibliography

All elements placed on the Bibliography were key to the evolution of this project, however this project does not fully rely on bibliography and a lot of work behind the data set generator path finding algorithm is original.

A.2 Neural network accuracy

The displayability of the neural network accuracy was not presented on this document because of some troubles related to google Colab working.

Appendix



Source code

HTML5 Code

```
<!DOCTYPE html>
<html>
    <head>
        <title>Dataset Generator</title>
        <style>
            body {
                background: rgb(255, 255, 255);
            }

            .button {
                background: #ffffff;
                color:rgb(14, 27, 39);
                padding: 5px;
                width: 100%;
                outline: none;
                border: none;
                border-radius: 10px;
                box-shadow: 0px 8px 15px rgba(23, 27, 21, 0.2);
                transition: all 0.3s ease 0s;
                font-family:-apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto,
                Oxygen, Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
            }

            .button:hover {
                background-color: #ff9b17;
                box-shadow: 0px 15px 20px rgba(229, 159, 46, 0.4);
                color: #fff;
                transform: scale(110%);
                font-family:-apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto,
                Oxygen, Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
            }

            .foot {
                padding: 5px;
                padding-top: 100px;
                background: #ffffff;
            }

            #footText {
                font-size: 0.6em;
                font-style: italic;
                font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto,
                Oxygen, Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
            }

            #main {
                margin-top: 10px;
                text-align: center;
            }
            #canvas {
                background-color: rgb(0, 0, 0);
                border-style:dotted;
```

```
53                 border-radius: 10px;
54                 border-color: rgb(229, 159, 46);
55                 box-shadow: 0px 15px 15px rgba(229, 159, 46, 0.4);
56                 transform: scale(150%);
57                 height: 256px;
58                 width: 256px;
59             }
60             #body {
61                 margin: auto;
62                 margin-top: 100px;
63                 width: 50%;
64                 text-align: center;
65                 justify-content: space-evenly;
66                 display: flex;
67             }
68 
69             #buttons {
70                 margin: initial;
71                 padding-left:95px;
72                 width: 100%;
73                 text-align: center;
74                 justify-content: space-evenly;
75                 display: flex;
76                 flex-direction: column;
77                 align-items: stretch;
78             }
79             #selector{
80                 margin: auto;
81                 margin-top: 100px;
82                 width: 50%;
83                 text-align: center;
84                 justify-content: space-evenly;
85                 display: flex;
86             }
87         </style>
88     </head>
89     <body>
90         <div id="body">
91             <canvas id="canvas"></canvas>
92 
93             <p><img id="preview"></p>
94 
95             <p id = "buttons">
96                 <button class = button onclick = "solveWaypoint();">Solve</button>
97                 <button class = button id="debug1" onclick = "download();">Download</button>
98                 <button class = button id="debug5" onclick = "CreateRandomNoise()">Noise</button>
99             </p>
100         </div>
101 
102         <script src='perlin.js'></script>
103         <script src='datasetGenerator.js'></script>
104 
105         <div id = "selector"><input class = "button" type="file" onchange="previewImage(this);"></div>
106         <footer class="foot">
```

```
            <p id = "footText">Author: Jesús Villar Méndez<br>
                Noise generation library: Joseph Gentle <br>
                <a href="mailto:JVMEmploy@gmail.com">JVMEmploy@gmail.com</a></p>
        </footer>
    </body>
</html>
```

JavaScript Code

```
class Point {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
    position() {
        return this.y * 4 * canvas.width + this.x * 4;
    }
    getColor() {
        var result = 'U';
        if (data[this.position()] == 255) {
            if (data[this.position() + 1] == 255) {
                if (data[this.position() + 2] == 255) {
                    result = 'W';
                }
                else if (data[this.position() + 2] == 0) {
                    result = 'Y';
                }
            }
            else if (data[this.position() + 1] == 0) {
                if (data[this.position() + 2] == 255) {
                    result = 'M'
                }
                else if (data[this.position() + 2] == 0) {
                    result = 'R'
                }
            }
        }
        else if (data[this.position()] == 0) {
            if (data[this.position() + 1] == 255) {
                if (data[this.position() + 2] == 0) {
                    result = 'G';
                }
            }
            else if (data[this.position() + 1] == 0) {
                if (data[this.position() + 2] == 255) {
                    result = 'B'
                }
                else if (data[this.position() + 2] == 0) {
                    result = 'K'
                }
            }
        }
```

```
44          return result;
45      }
46      setColor(r, g, b, a) {
47          data[this.position()] = r;
48          data[this.position() + 1] = g;
49          data[this.position() + 2] = b;
50          data[this.position() + 3] = a;
51      }
52      getColorRGBA() {
53          var r = data[this.position()];
54          var g = data[this.position() + 1];
55          var b = data[this.position() + 2];
56          var a = data[this.position() + 3];
57          return new Array(r, g, b, a);
58      }
59      nearTo(color) {
60          var aux1 = new Point(this.x, this.y + 1);
61          var aux2 = new Point(this.x + 1, this.y);
62          var aux3 = new Point(this.x, this.y - 1);
63          var aux4 = new Point(this.x - 1, this.y);
64          var aux5 = new Point(this.x + 1, this.y + 1);
65          var aux6 = new Point(this.x - 1, this.y + 1);
66          var aux7 = new Point(this.x + 1, this.y - 1);
67          var aux8 = new Point(this.x - 1, this.y - 1);
68          return aux1.getColor() == color || aux2.getColor() == color ||
69              aux3.getColor() == color || aux4.getColor() == color ||
70              aux5.getColor() == color || aux6.getColor() == color ||
71              aux7.getColor() == color || aux8.getColor() == color;
72      }
73      countNear(color) {
74          var aux1 = new Point(this.x, this.y + 1);
75          var aux2 = new Point(this.x + 1, this.y);
76          var aux3 = new Point(this.x, this.y - 1);
77          var aux4 = new Point(this.x - 1, this.y);
78          var count = 0;
79          if (aux1.getColor() == color) {
80              count++;
81          }
82          if (aux2.getColor() == color) {
83              count++;
84          }
85          if (aux3.getColor() == color) {
86              count++;
87          }
88          if (aux4.getColor() == color) {
89              count++;
90          }
91          return count;
92      }
93      tangentOfColor(color) {
94          var down = new Point(this.x, this.y + 1);
95          var right = new Point(this.x + 1, this.y);
96          var up = new Point(this.x, this.y - 1);
97          var left = new Point(this.x - 1, this.y);
```

```
98
99          var elements = new Array(0);
100
101         if (up.getColor() == color) {
102             elements.push(up);
103         }
104         if (down.getColor() == color) {
105             elements.push(down);
106         }
107         if (right.getColor() == color) {
108             elements.push(right);
109         }
110         if (left.getColor() == color) {
111             elements.push(left);
112         }
113         return elements;
114     }
115     pointsNearTo(color) {
116         var aux1 = new Point(this.x, this.y + 1);
117         var aux2 = new Point(this.x + 1, this.y);
118         var aux3 = new Point(this.x, this.y - 1);
119         var aux4 = new Point(this.x - 1, this.y);
120         var aux5 = new Point(this.x + 1, this.y + 1);
121         var aux6 = new Point(this.x - 1, this.y + 1);
122         var aux7 = new Point(this.x + 1, this.y - 1);
123         var aux8 = new Point(this.x - 1, this.y - 1);
124
125         var elements = new Array(0);
126
127         if (aux1.getColor() == color) {
128             elements.push(aux1);
129         }
130         if (aux2.getColor() == color) {
131             elements.push(aux2);
132         }
133         if (aux3.getColor() == color) {
134             elements.push(aux3);
135         }
136         if (aux4.getColor() == color) {
137             elements.push(aux4);
138         }
139         if (aux5.getColor() == color) {
140             elements.push(aux5);
141         }
142         if (aux6.getColor() == color) {
143             elements.push(aux6);
144         }
145         if (aux7.getColor() == color) {
146             elements.push(aux7);
147         }
148         if (aux8.getColor() == color) {
149             elements.push(aux8);
150         }
151         return elements;
```

```
152     }
153     directionTo(color) {
154         var down = new Point(this.x, this.y + 1);
155         var right = new Point(this.x + 1, this.y);
156         var up = new Point(this.x, this.y - 1);
157         var left = new Point(this.x - 1, this.y);
158 
159         var directedToGreen = new Array(0);
160 
161         if (up.getColor() == color) {
162             directedToGreen.push('U');
163         }
164         if (down.getColor() == color) {
165             directedToGreen.push('D');
166         }
167         if (right.getColor() == color) {
168             directedToGreen.push('R');
169         }
170         if (left.getColor() == color) {
171             directedToGreen.push('L');
172         }
173         return directedToGreen;
174     }
175     next() {
176         var coors = this.getColorRGBA()
177         return new Point(coors[2], coors[3]);
178     }
179     last() {
180         var coors = this.getColorRGBA()
181         return new Point(coors[0], coors[1]);
182     }
183 
184     getDirectionVector() {
185         var coors = this.getColorRGBA()
186         return new Point(coors[2] - this.x, coors[3] - this.y);
187     }
188 }
189 
190 class wayPoint {
191     constructor(x, y, parent) {
192         this.x = x;
193         this.y = y;
194         this.parent = parent
195         this.d = 0;
196         if (parent != null) {
197             this.d = Math.sqrt((parent.x - this.x) ^ 2 + (parent.y - this.y) ^ 2)
198         }
199         this.childs = new Array(0);
200     }
201     addChild(x, y) {
202         var push = true;
203         this.childs.forEach(element => {
204             if (element.x == x && element.y == y) {
205                 if (element.d == this.d) {
```

```
206                     push = false;
207                 }
208             }
209         });
210         if (this.parent != null && this.parent.x == x && this.parent.y == y) {
211             push = false;
212         }
213         if (push) {
214             this.childs.push(new wayPoint(x, y, this));
215         }
216     }
217     renderUp() {
218         var nextNode = this.parent;
219         var thisPoint = new Point(this.x, this.y);
220         if (nextNode != null) {
221             var parentPoint = new Point(nextNode.x, nextNode.y);
222         }
223         while (nextNode != null) {
224             DrawColorLine(thisPoint, parentPoint);
225             thisPoint.setColor(255, 0, 0, 255);
226             parentPoint.setColor(255, 0, 0, 255);
227             thisPoint = new Point(nextNode.x, nextNode.y);
228             if (nextNode.parent != null) {
229                 parentPoint = new Point(nextNode.parent.x, nextNode.parent.y);
230                 nextNode = nextNode.parent;
231             }
232             else {
233                 nextNode = null;
234             }
235         }
236         generalInit.setColor(255, 0, 0, 255);
237         end.setColor(255, 0, 0, 255);
238         reload();
239     }
240 }
241 
242 var findEnd = true;
243 var canvas = document.getElementById("canvas");
244 var context = canvas.getContext("2d");
245 var myImageData;
246 var data;
247 var init = new Point(null, null);
248 var generalInit = new Point(null, null);
249 var end = new Point(null, null);
250 var redrawneeded = false;
251 var solved = false;
252 var contact;
253 var borderStack = new Array();
254 var pointsToSet = 2;
255 var waypointTree = new wayPoint(0, 0, null);
256 var noiseSeed1 = Math.random();
257 var noiseSeed2 = Math.random();
258 var normalized = false;
259 context.imageSmoothingEnabled = false;
```

```
document.getElementById("preview").style.display = "none";
canvas.height = 256;
canvas.width = 256;
context.drawImage(new Image(), 0, 0, canvas.width, canvas.height);
myImageData = context.getImageData(0, 0, canvas.width, canvas.height);
data = myImageData.data;
canvas.addEventListener('click', checkColor, false);


function previewImage(input) {
    var reader = new FileReader();
    reader.onload = function (e) {
        document.getElementById("preview").setAttribute("src", e.target.result);
    };
    reader.readAsDataURL(input.files[0]);
    setTimeout(() => { setImage(); }, 100);
}
function setImage() {
    console.log("Wiwi")
    var image = document.getElementById("preview");
    if (Math.min(image.height, image.width) > 256) {
        canvas.height = 256;
        canvas.width = 256;
    }
    else {
        canvas.height = image.height;
        canvas.width = image.width;
    }

    pointsToSet = 2;
    context.drawImage(image, 0, 0, canvas.width, canvas.height);
    myImageData = context.getImageData(0, 0, canvas.width, canvas.height);
    data = myImageData.data;
    reload();
}

function download() {
    downloadCanvasAsImage("target");
    changeColors('R', 0, 0, 255, 255);
    changeColors('M', 0, 0, 255, 255);
    generalInit.setColor(255, 0, 0, 255);
    end.setColor(255, 0, 0, 255);
    reload();
    downloadCanvasAsImage("init");
}

function downloadCanvasAsImage(title) {
    let downloadLink = document.createElement('a');
    downloadLink.setAttribute('download', title + ".png");
    canvas.toBlob(function (blob) {
        let url = URL.createObjectURL(blob);
        downloadLink.setAttribute('href', url);
        downloadLink.click();
    });
```

```
314 }
315
316 function solveWaypoint() {
317     waypointTree = new wayPoint(generalInit.x, generalInit.y, null);
318     normalizeRenderColors();
319     findEnd = true;
320     growTree(waypointTree);
321 }
322
323 function growTree(waypoint) {
324     var borderInit = solve()
325     if (borderInit != null) {
326         followLoop(borderInit, waypoint, true)
327         clearLoops();
328         if (waypoint.childs.length > 0) {
329             waypoint.childs.forEach(child => {
330                 init = new Point(child.x, child.y);
331                 if (!canDrawLine(init, end) && findEnd) {
332                     growTree(child);
333                 }
334                 else if (findEnd) {
335                     child.addChild(end.x, end.y);
336                     child.childs.forEach(child => {
337                         if (child.x == end.x && child.y == end.y) {
338                             child.renderUp();
339                             findEnd = false;
340                         }
341                     });
342                 }
343             });
344         }
345     }
346 }
347 function drawLine(pointA, pointB) {
348     if (canDrawLine(pointA, pointB)) {
349         DrawColorLine(pointA, pointB);
350         solved = true;
351         return null;
352     }
353     var x = pointA.x;
354     var y = pointA.y;
355     var dY = pointB.y - pointA.y;
356     var dX = pointB.x - pointA.x;
357     var IncYi;
358     var IncXi;
359     var IncYr;
360     var IncXr
361     if (dY >= 0) { IncYi = 1; }
362     else { dY = -dY; IncYi = -1; }
363     if (dX >= 0) { IncXi = 1; }
364     else { dX = -dX; IncXi = -1; }
365     if (dX >= dY) { IncYr = 0; IncXr = IncXi; }
366     else { IncXr = 0; IncYr = IncYi; var aux = dX; dX = dY; dY = aux; }
367     var avR = 2 * dY; var av = avR - dX; var avI = av - dX;
```

```
368
369     var currentPos = new Point(null, null);
370     var auxPos = new Point(null, null);
371     contact = new Point(null, null);
372     while (!(x == pointB.x && y == pointB.y)) {
373         currentPos.x = x;
374         currentPos.y = y;
375         if (currentPos.getColor() == 'R'
376         && !(currentPos.x == init.x && currentPos.y == init.y)) {
377             break;
378         }
379         auxPos.x = currentPos.x + IncXi;
380         auxPos.y = currentPos.y + IncYi;
381         if (av >= 0 && auxPos.getColor() == 'G') {
382             contact.x = x;
383             contact.y = y;
384             changeColors('G', 255, 255, 0, 255);
385             colorInTouchIsland(contact, 'Y', 0, 255, 0, 255);
386             borderIsland(contact.x, contact.y, pointA.x, pointA.y);
387             changeColors('Y', 0, 255, 0, 255);
388             return contact;
389         }
390         else {
391             auxPos.x = currentPos.x + IncXr;
392             auxPos.y = currentPos.y + IncYr;
393             if (auxPos.getColor() == 'G') {
394                 contact.x = x;
395                 contact.y = y;
396                 changeColors('G', 255, 255, 0, 255);
397                 colorInTouchIsland(contact, 'Y', 0, 255, 0, 255);
398                 borderIsland(contact.x, contact.y, pointA.x, pointA.y);
399                 changeColors('Y', 0, 255, 0, 255);
400                 return contact;
401             }
402         }
403         if (av >= 0) { x = x + IncXi; y = y + IncYi; av = av + avI; }
404         else { x = x + IncXr; y = y + IncYr; av = av + avR; }
405     }
406     solved = true;
407     return null;
408 }
409
410 function clearLoops() {
411     changeColors('U', 0, 0, 255, 255);
412 }
413
414 function clearInitEnd() {
415     init = null;
416     generalInit = null;
417     end = null;
418     pointsToSet = 2;
419     reload();
420 }
421
```

```
422
423 function solve() {
424     solved = false;
425     if (!normalized) {
426         normalizeRenderColors();
427     }
428     return drawLine(init, end);
429 }
430
431 function changeColors(color, r, g, b, a) {
432     var changingPoint = new Point(0, 0);
433     for (var x = 0; x < canvas.width; x++) {
434         for (var y = 0; y < canvas.height; y++) {
435             changingPoint.x = x;
436             changingPoint.y = y;
437             if (changingPoint.getColor() == color) {
438                 changingPoint.setColor(r, g, b, a);
439             }
440         }
441     }
442 }
443
444 function colorInTouchIsland(point, islandColor, r, g, b, a) {
445     var p = point.pointsNearTo(islandColor);
446     if (p.length > 0) {
447         for (var i = 0; i < p.length; i++) {
448             p[i].setColor(r, g, b, a);
449             colorInTouchIsland(p[i], 'Y', 0, 255, 0, 255);
450         }
451     }
452 }
453
454 function normalizeRenderColors() {
455     if (!normalized) {
456         normalized = true;
457         var auxPoint = new Point(null, null);
458         for (x = 0; x < canvas.width; x++) {
459             for (y = 0; y < canvas.height; y++) {
460                 auxPoint.x = x;
461                 auxPoint.y = y;
462                 if (auxPoint.getColorRGBA()[0] == 54 &&
463                     auxPoint.getColorRGBA()[1] == 59 &&
464                     auxPoint.getColorRGBA()[2] == 160
465                 ) {
466                     auxPoint.setColor(0, 0, 255, 255);
467                 }
468                 else if (auxPoint.getColorRGBA()[0] == 51 &&
469                     auxPoint.getColorRGBA()[1] == 69 &&
470                     auxPoint.getColorRGBA()[2] == 173
471                 ) {
472                     auxPoint.setColor(0, 0, 255, 255);
473                 }
474                 else if (auxPoint.getColorRGBA()[0] == 56 &&
475                     auxPoint.getColorRGBA()[1] == 76 &&
```

```
476                     auxPoint.getColorRGBA()[2] == 183
477                 ) {
478                     auxPoint.setColor(0, 0, 255, 255);
479                 }
480                 else if (auxPoint.getColorRGBA()[0] == 255 &&
481                     auxPoint.getColorRGBA()[1] == 0 &&
482                     auxPoint.getColorRGBA()[2] == 0) {
483                     auxPoint.setColor(255, 0, 0, 255);
484                 }
485                 else if (auxPoint.getColorRGBA()[0] == 207 &&
486                     auxPoint.getColorRGBA()[1] == 182 &&
487                     auxPoint.getColorRGBA()[2] == 49) {
488                     auxPoint.setColor(0, 255, 0, 255);
489                 }
490                 else if (auxPoint.getColorRGBA()[0] == 0 &&
491                     auxPoint.getColorRGBA()[1] == 140 &&
492                     auxPoint.getColorRGBA()[2] == 0) {
493                     auxPoint.setColor(0, 255, 0, 255);
494                 }
495                 else if (auxPoint.getColorRGBA()[0] == 87 &&
496                     auxPoint.getColorRGBA()[1] == 100 &&
497                     auxPoint.getColorRGBA()[2] == 42) {
498                     auxPoint.setColor(0, 255, 0, 255);
499                 }
500                 else if (auxPoint.getColorRGBA()[0] == 143 &&
501                     auxPoint.getColorRGBA()[1] == 104 &&
502                     auxPoint.getColorRGBA()[2] == 52) {
503                     auxPoint.setColor(0, 255, 0, 255);
504                 }
505                 else if (auxPoint.getColorRGBA()[0] == 93 &&
506                     auxPoint.getColorRGBA()[1] == 74 &&
507                     auxPoint.getColorRGBA()[2] == 30) {
508                     auxPoint.setColor(0, 255, 0, 255);
509                 }
510             }
511         }
512         reload();
513     }
514     else {
515         normalized = false;
516         CreateNoise();
517     }
518 
519 }
520 
521 
522 function canDrawLine(pointA, pointB) {
523     var x = pointA.x;
524     var y = pointA.y;
525     var dY = pointB.y - pointA.y;
526     var dX = pointB.x - pointA.x;
527     var IncYi;
528     var IncXi;
529     var IncYr;
```

```
530     var IncXr
531     if (dY >= 0) { IncYi = 1; }
532     else { dY = -dY; IncYi = -1; }
533     if (dX >= 0) { IncXi = 1; }
534     else { dX = -dX; IncXi = -1; }
535     if (dX >= dY) { IncYr = 0; IncXr = IncXi; }
536     else { IncXr = 0; IncYr = IncYi; var aux = dX; dX = dY; dY = aux; }
537     var avR = 2 * dY; var av = avR - dX; var avI = av - dX;
538 
539     var currentPos = new Point(null, null);
540     var auxPos = new Point(null, null);
541     contact = new Point(null, null);
542     var drawable = true;
543     while (!(x == pointB.x && y == pointB.y)) {
544         currentPos.x = x;
545         currentPos.y = y;
546         if (av >= 0) {
547             auxPos.x = currentPos.x + IncXi;
548             auxPos.y = currentPos.y + IncYi;
549         }
550         else {
551             auxPos.x = currentPos.x + IncXr;
552             auxPos.y = currentPos.y + IncYr;
553         }
554         if (auxPos.getColor() == 'G') {
555             drawable = false;
556             break;
557         }
558         if (av >= 0) { x = x + IncXi; y = y + IncYi; av = av + avI; }
559         else { x = x + IncXr; y = y + IncYr; av = av + avR; }
560     }
561     return drawable;
562 }
563 
564 function DrawColorLine(pointA, pointB) {
565     var x = pointA.x;
566     var y = pointA.y;
567     var dY = pointB.y - pointA.y;
568     var dX = pointB.x - pointA.x;
569     var IncYi;
570     var IncXi;
571     var IncYr;
572     var IncXr
573     if (dY >= 0) { IncYi = 1; }
574     else { dY = -dY; IncYi = -1; }
575     if (dX >= 0) { IncXi = 1; }
576     else { dX = -dX; IncXi = -1; }
577     if (dX >= dY) { IncYr = 0; IncXr = IncXi; }
578     else { IncXr = 0; IncYr = IncYi; var aux = dX; dX = dY; dY = aux; }
579     var avR = 2 * dY; var av = avR - dX; var avI = av - dX;
580 
581     var currentPos = new Point(null, null);
582 
583     contact = new Point(null, null);
```

```
584     while (!(x == pointB.x && y == pointB.y)) {
585         currentPos.x = x;
586         currentPos.y = y;
587 
588         currentPos.setColor(255, 0, 255, 255);
589         if (av >= 0) { x = x + IncXi; y = y + IncYi; av = av + avI; }
590         else { x = x + IncXr; y = y + IncYr; av = av + avR; }
591         reload();
592     }
593 }
594 
595 function followLoop(point, waypoint, visibility) {
596     var aux = point.next();
597 
598     var vector = point.getDirectionVector();
599     var tangentPoints = point.tangentOfColor('U');
600     if (tangentPoints.length > 2) {
601         for (var i = 0; i < tangentPoints.length; i++) {
602             if (arePerpendicular(point, tangentPoints[i])) {
603                 if (!(tangentPoints[i].x == point.next().x
604                 && tangentPoints[i].y == point.next().y)
605                     && (tangentPoints[i].last().x == point.x
606                     && tangentPoints[i].last().y == point.y)
607                 ) {
608                     if (!(point.next().x == point.x && point.next().y == point.y)) {
609                         borderStack.push(point.next());
610                     }
611                     aux = tangentPoints[i];
612                 }
613             }
614         }
615     }
616     //Normal treatment
617     var wayPointPoint = new Point(waypoint.x, waypoint.y)
618     if (!(point.next().x == point.x && point.next().y == point.y)) {
619         if (canDrawLine(point, wayPointPoint) && !visibility) {
620             visibility = true;
621             if (waypoint.parent != null && waypoint.parent.parent != null) {
622                 if (!canDrawLine(point,
623                     new Point(waypoint.parent.parent.x, waypoint.parent.parent.y))) {
624                     waypoint.addChild(point.x, point.y);
625                 }
626             }
627             else {
628                 waypoint.addChild(point.x, point.y);
629             }
630         }
631         else if (visibility && !canDrawLine(point, wayPointPoint)) {
632             visibility = false;
633             if (waypoint.parent != null && waypoint.parent.parent != null) {
634                 if (!canDrawLine(point.last(),
635                 new Point(waypoint.parent.parent.x, waypoint.parent.parent.y))) {
636                     waypoint.addChild(point.last().x, point.last().y);
637                 }
```

```
638             }
639             else {
640                 waypoint.addChild(point.last().x, point.last().y);
641             }
642         }
643         followLoop(aux, waypoint, visibility);
644     }
645 
646     if (point.next().x == point.x && point.next().y == point.y) {
647         if (borderStack.length > 0) {
648             if (canDrawLine(point, wayPointPoint) && visibility == false) {
649                 visibility = true;
650                 if (waypoint.parent != null && waypoint.parent.parent != null) {
651                     if (!canDrawLine(point,
652                         new Point(waypoint.parent.parent.x, waypoint.parent.parent.y))) {
653                         waypoint.addChild(point.x, point.y);
654                     }
655                 }
656                 else {
657                     waypoint.addChild(point.x, point.y);
658                 }
659             }
660             else if (visibility && !canDrawLine(point, wayPointPoint)) {
661                 visibility = false;
662                 if (waypoint.parent != null && waypoint.parent.parent != null) {
663                     if (!canDrawLine(point.last(),
664                     new Point(waypoint.parent.parent.x, waypoint.parent.parent.y))) {
665                         waypoint.addChild(point.last().x, point.last().y);
666                     }
667                 }
668                 else {
669                     waypoint.addChild(point.last().x, point.last().y);
670                 }
671             }
672             aux = borderStack.pop();
673             followLoop(aux, waypoint, visibility);
674         }
675         else {
676             return;
677         }
678     }
679 }
680 
681 function arePerpendicular(pointA, pointB) {
682     vectorA = pointA.getDirectionVector();
683     vectorB = pointA.getDirectionVector();
684     vectorA = normalizeVector(vectorA);
685     vectorB = normalizeVector(vectorB);
686 
687     return (Math.abs(vectorA.x) == Math.abs(vectorB.x)
688     || Math.abs(vectorB.x) == Math.abs(vectorB.y));
689 }
690 
691 function normalizeVector(vector) {
```

```
692     if (Math.abs(vector.x) > Math.abs(vector.y)) {
693         vector.y = 0;
694         vector.x = (vector.x != 0) ? Math.abs(vector.x) / vector.x : 0;
695     }
696     else {
697         vector.x = 0;
698         vector.y = (vector.y != 0) ? Math.abs(vector.y) / vector.y : 0;
699     }
700     return vector;
701 }
702 function getMousePos(canvas, evt) {
703     var rect = canvas.getBoundingClientRect();
704     var MousePos = new Point(Math.round((evt.clientX - rect.left) / (rect.right - rect.left) * canvas.width),
705     Math.round((evt.clientY - rect.top) / (rect.bottom - rect.top) * canvas.height))
706     return MousePos;
707 }
708 function checkColor(evt) {
709     var pos = getMousePos(canvas, evt)
710     if (pointsToSet == 0) {
711         console.log('================')
712         console.log('x, y ', pos.x, pos.y);
713         console.log('================')
714         console.log('LastX: ', data[pos.position()]);
715         console.log('LastY: ', data[pos.position() + 1]);
716         console.log('----------------')
717         console.log('NextX: ', data[pos.position() + 2]);
718         console.log('NextY: ', data[pos.position() + 3]);
719         console.log('================')
720         console.log(pos.getColor());
721     }
722     else {
723         pos.setColor(255, 0, 0, 255);
724         pointsToSet--;
725         if (pointsToSet == 1) { init = pos; generalInit = pos }
726         else { end = pos; }
727         reload();
728     }
729 }
730 function borderIsland(x, y, prevx, prevy) {
731     if (x < canvas.width && y < canvas.height && x > -1 && y > -1 && !solved) {
732         var point = new Point(x, y)
733         if ((point.getColor() == 'B' || point.getColor() == 'R') && point.nearTo('G')) {
734             var up = new Point(x, y - 1);
735             var down = new Point(x, y + 1);
736             var right = new Point(x + 1, y);
737             var left = new Point(x - 1, y);
738             var vector = new Point((x - prevx), (y - prevy));
739             //NormalizeVector
740             vector = normalizeVector(vector);
741             //PreviousPoint
742             var previousPoint = new Point(prevx, prevy);
743 
744             if (vector.x == 0) {
745                 if (vector.y == 1) {
```

```
                    vector = 'D'
                }
                else {
                    vector = 'U'
                }
            }
            else if (vector.x == 1) {
                vector = 'R';
            }
            else { vector = 'L' }

            let order = new Array();
            switch (vector) {
                case 'U':
                    if (previousPoint.directionTo('G').includes('R')) {
                        order.push(right);
                    }
                    else if (point.directionTo('G').includes('R')) {
                        order.push(right);
                    }
                    else { order.push(left) }
                    order.push(up);
                    if (order.includes(left)) {
                        order.push(right);
                    }
                    else {
                        order.push(left)
                    }
                    order.push(down);
                    break;
                case 'D':
                    if (previousPoint.directionTo('G').includes('L')) {
                        order.push(left);
                    }
                    else if (point.directionTo('G').includes('L')) {
                        order.push(left);
                    }
                    else { order.push(right); }
                    order.push(down);
                    if (order.includes(left)) {
                        order.push(right);
                    }
                    else { order.push(left); }
                    order.push(up);
                    break;
                case 'L':
                    if (previousPoint.directionTo('G').includes('U')) {
                        order.push(up);
                    }
                    else if (point.directionTo('G').includes('U')) {
                        order.push(up);
                    }
                    else { order.push(down); }
                    order.push(left);
```

```
800                     if (order.includes(up)) {
801                         order.push(down);
802                     }
803                     else { order.push(up); }
804                     order.push(right);
805                     break;
806                 case 'R':
807                     if (previousPoint.directionTo('G').includes('U')) {
808                         order.push(up);
809                     }
810                     else if (point.directionTo('G').includes('U')) {
811                         order.push(up);
812                     }
813                     else { order.push(down); }
814                     order.push(right);
815                     if (order.includes(up)) {
816                         order.push(down);
817                     }
818                     else { order.push(up); }
819                     order.push(left);
820                     break;
821             }
822             if ((order[0].getColor() == 'B'
823             || order[0].getColor() == 'R')
824             && order[0].nearTo('G')) {
825                 point.setColor(prevx, prevy, order[0].x, order[0].y);
826                 redrawneeded = false;
827                 borderIsland(order[0].x, order[0].y, x, y);
828             }
829             else if ((order[1].getColor() == 'B'
830             || order[1].getColor() == 'R')
831             && order[1].nearTo('G')) {
832                 point.setColor(prevx, prevy, order[1].x, order[1].y);
833                 redrawneeded = false;
834                 borderIsland(order[1].x, order[1].y, x, y);
835             }
836             else if ((order[2].getColor() == 'B'
837             || order[2].getColor() == 'R')
838             && order[2].nearTo('G')) {
839                 point.setColor(prevx, prevy, order[2].x, order[2].y);
840                 redrawneeded = false;
841                 borderIsland(order[2].x, order[2].y, x, y);
842             }
843             else {
844                 if ((right.x == contact.x && right.y == contact.y)
845                 || (left.x == contact.x && left.y == contact.y)
846                 || (up.x == contact.x && up.y == contact.y)
847                 || (down.x == contact.x && down.y == contact.y)) {
848                     solved = true;
849                 }
850                 else {
851                     redrawneeded = true;
852                 }
853                 point.setColor(prevx, prevy, point.x, point.y);
```

```
854                 return;
855             }
856 
857             if (redrawneeded && (order[0].getColor() == 'B'
858             || order[0].getColor() == 'R')
859             && order[0].nearTo('G')) {
860                 point.setColor(prevx, prevy, order[0].x, order[0].y);
861                 redrawneeded = false;
862                 borderIsland(order[0].x, order[0].y, x, y);
863             }
864             else if (redrawneeded && (order[1].getColor() == 'B'
865                 || order[1].getColor() == 'R')
866                 && order[1].nearTo('G')) {
867                 point.setColor(prevx, prevy, order[1].x, order[1].y);
868                 redrawneeded = false;
869                 borderIsland(order[1].x, order[1].y, x, y);
870             }
871             else if (redrawneeded && (order[2].getColor() == 'B'
872             || order[2].getColor() == 'R')
873             && order[2].nearTo('G')) {
874                 point.setColor(prevx, prevy, order[2].x, order[2].y);
875                 redrawneeded = false;
876                 borderIsland(order[2].x, order[2].y, x, y);
877             }
878         }
879     }
880 }
881 function reload() { context.putImageData(myImageData, 0, 0); }
882 
883 function CreateRandomNoise() {
884     canvas.height = 256;
885     canvas.width = 256;
886     noiseSeed1 = Math.random();
887     noiseSeed2 = Math.random();
888     pointsToSet = 2;
889     CreateNoise();
890 }
891 function CreateNoise() {
892     normalized = false;
893     var height = 100;
894     height = 115;
895     var auxPoint = new Point(0, 0)
896     for (var x = 0; x < canvas.width; x++) {
897         for (var y = 0; y < canvas.height; y++) {
898             auxPoint.x = x;
899             auxPoint.y = y;
900             scale = 4.25;
901             zoom = 175;
902             noise.seed(noiseSeed1);
903             var value = noise.simplex2(x * scale / zoom, y * scale / zoom);
904             noise.seed(noiseSeed2);
905             var value2 = noise.simplex2(x * scale * 2.5 / zoom, y * scale * 2.5 / zoom);
906             value = (((value + value2) / 2) + 1) * 128;
907 
```

```
908             if (value > 115 + height) { // High mountains
909                 auxPoint.setColor(93, 74, 30, 255)
910             }
911             else if (value > 95 + height) { // Mountains
912                 auxPoint.setColor(143, 104, 52, 255)
913             }
914             else if (value > 80 + height) { //Hillside
915                 auxPoint.setColor(87, 100, 42, 255)
916             }
917             else if (value > 55 + height) { //Grass
918                 auxPoint.setColor(0, 140, 0, 255)
919             }
920             else if (value > 45 + height) { // Sand
921                 auxPoint.setColor(207, 182, 49, 255)
922             }
923             else if (value > 40 + height) { //Shallow Water
924                 auxPoint.setColor(56, 76, 183, 255)
925             }
926             else if (value > 30 + height) { //Water
927                 auxPoint.setColor(56, 76, 183, 255)
928             }
929             else if (value > 10 + height) { //Deep Water
930                 auxPoint.setColor(51, 69, 173, 255)
931             }
932             else { //Very deep water
933                 auxPoint.setColor(54, 59, 160, 255)
934             }
935         }
936     }
937     reload();
938 }
```