Section A – Project background
To be completed by the candidate and returned to the teacher for approval before the project is started

Project title: Julia Set Generator

Implementation language and/or proposed packages to be used: Pascal

Name of end user and key role (ie job title) in relation to the project:

Section B – Summary of marks
To be completed by the teacher

Marks must be awarded in accordance with the instructions and criteria in the specification.

Section	Maximum mark	Mark awarded
1 Analysis	9	8
2 Documented design	12	11
3 Technical solution	42	35
4 Testing	8	8
5 Evaluation	4	3
Total mark	75	65

Details of additional assistance given
Record here details of any assistance given to this candidate which is beyond that given to the class as a whole and beyond that described in the specification (*continue on a separate sheet if necessary*).

Click here to enter text.

None

Concluding comments

Click here to enter text.

Please see project log.

To see how we comply with the Data Protection Act 1998 please see our Privacy Statement at aqa.org.uk/privacy

AQA
Realising potential

2017 Project log

A-level Computer Science (7517)
Computing Practical Project (7517/C)

Please attach a copy of this form securely to the front your candidate's work.

Centre number **Centre name**

Section one - the project
To be completed by the candidate and returned to the teacher for approval before the project is started

Project title Julia Set Generator

Project type ☐ problem ☒ investigation

Outline description
The projects main objective was to allow the user to display Julia sets by allowing them to add in their own constants which are then subbed into the defined function. The project was also an investigation into how these points are assigned their colour by allowing the user to generate a Z path which shows the user whether a point is inside or outside the set. There are also some other features allowing the user to produce an animation of a changing Julia set, changing the function and changing the colour of the set.

To be completed by the teacher:
From the given description the project is at a standard required for A-level Yes/No

Section two – project assessment
To be completed by the teacher

Analysis			
Level	**Criteria**	**Mark**	**Comments/evidence**
3	Fully or nearly fully scoped analysis of a real problem, presented in a way that a third party can understand. Requirements fully documented in a set of measurable and appropriate specific objectives, covering all required functionality of the solution or areas of investigation. Requirements arrived at by considering, through dialogue, the needs of the intended users of the system, or recipients of the outcomes for investigative projects. Problem sufficiently well modelled to be of use in subsequent stages.	7-9	Fully scoped. Hand drawn notes from end user and candidate's discussion of these ar particularly useful. Objectives could be broken down further, but very detailed. Could have more detail about end users' planned use for the software in how it would be used to teach a class.
2	Well scoped analysis (but with some omissions that are not serious enough to undermine later design) of a real problem. Most, but not all, requirements documented in a set of, in the main, measurable and appropriate specific objectives that cover most of the required functionality of a solution or areas of investigation. Requirements arrived at, in the main, by considering, through dialogue, the needs of the intended users of the system, or recipients of the outcomes for investigative projects. Problem sufficiently well modelled to be of use in subsequent stages.	4-6	
1	Partly scoped analysis of a problem. Requirements partly documented in a set of specific objectives, not all of which are measurable or appropriate for developing a solution. The required functionality or areas of investigation are only partly addressed. Some attempt to consider, through dialogue, the needs of the intended users of the system, or recipients of the outcomes for investigative projects. Problem partly modelled and of some use in subsequent stages.	1-3	
	No evidence presented	0	**Mark awarded: 8**

Documented design			
Level	**Criteria**	**Mark**	**Comments/evidence**
4	Fully or nearly fully articulated design for a real problem, that describes how all or almost all of the key aspects of the solution/investigation are to be structured/are structured.	10-12	Nearly fully articulated. Details on commentary sheet.
3	Adequately articulated design for a real problem that describes how most of the key aspects of the solution/investigation are to be structured/are structured.	7-9	
2	Partially articulated design for a real problem that describes how some aspects of the solution/investigation are to be structured/are structured.	4-6	
1	Inadequate articulation of the design of the solution so that it is difficult to obtain a picture of how the solution/investigation is to be structured/is structured without resorting to looking directly at the programmed solution.	1-3	
	No evidence presented	0	**Mark awarded:** 11

Technical solution – completeness			
Level	**Criteria**	**Mark**	**Comments/evidence**
3	A system that meets almost all of the requirements of a solution/an investigation (ignoring any requirements that go beyond the demands of A-level).	11-15	Fits level 3 description, but some of the smaller requests made by the end user at analysis stage were not fully done (these were less complex, but impacted slightly on ability to teach & with the tool made).
2	A system that achieves many of the requirements but not all. The marks at the top end of the band are for systems that include some of the most important requirements.	6-10	
1	A system that tackles some aspects of the problem or investigation.	1-5	
	No evidence presented	0	**Mark awarded:** 12

NOTES:
Completeness is not only about how well a solution meets the objectives set by the student but also what an expected technical solution might perform for this particular project.

Technical solution – techniques used			
Level	**Criteria**	**Mark**	**Comments/evidence**
3	The techniques used are appropriate and demonstrate a level of technical skill equivalent to those listed in Group A in **Table 1**. Program(s) demonstrate(s) that the skill required for this level has been applied sufficiently to demonstrate proficiency.	19-27	Mainly Group A/B - could have shown ability with OOP structure, but this was not attempted - maths very complex though. Good modular structure. Coding style was mainly good, but some identifiers were very unclear (A, B, C etc.).
2	The techniques used are appropriate and demonstrate a level of technical skill equivalent to those listed in Group B in **Table 1**. Program(s) demonstrate(s) that the skill required for this level has been applied sufficiently to demonstrate proficiency.	10-18	
1	The techniques used demonstrate a level of technical skill equivalent to those listed in Group C in **Table 1**. Program(s) demonstrate(s) that the skill required for this level has been applied sufficiently to demonstrate proficiency.	1-9	
	No evidence presented	0	**Mark awarded:** 23

NOTES:

The mark to be awarded, within the level, should be decided upon using these factors:

(1) The extent to which the criteria for the level have been achieved
(2) The quality of the coding style that the student has demonstrated
(3) The effectiveness of the solution.

It would be beneficial for these to also be referred to in the comments/evidence section.
Table 1 referred to is on pages 95-96 of the specification (version 1.4 December 2016)
Continue on a separate sheet if necessary

Testing			
Level	**Criteria**	**Mark**	**Comments/evidence**
4	Clear evidence, in the form of carefully selected representative samples, that thorough testing has been carried out. This demonstrates the robustness of the complete or nearly complete solution/thoroughness of investigation and that the requirements of the solution/investigation have been achieved.	7-8	Nearly complete testing - further issues identified by end user in evaluation, but this was probably down to needing to communicate better on limitations/requirements. Robust program. The system with the current requirements as they stand is fully tested.
3	Extensive testing has been carried out, but the evidence presented in the form of representative samples does not make clear that all of the core requirements of the solution/investigation have been achieved. This may be due to some key aspects not being tested or because the evidence is not always presented clearly.	5-6	
2	Evidence in the form of representative samples of moderately extensive testing, but falling short of demonstrating that the requirements of the solution/investigation have been achieved and the solution is robust/investigation thorough. The evidence presented is explained.	3-4	
1	A small number of tests have been carried out, which demonstrate that some parts of the solution work/some outcomes of the investigation are achieved. The evidence presented may not be entirely clear.	1-2	
	No evidence presented	0	**Mark awarded:** 8

Evaluation			
Level	**Criteria**	**Mark**	**Comments/evidence**
4	Full consideration given to how well the outcome meets all of its requirements. How the outcome could be improved if the problem was revisited is discussed and given detailed consideration. Independent feedback obtained of a useful and realistic nature, evaluated and discussed in a meaningful way.	4	Band 3 - please see commentary sheet.
3	Full or nearly full consideration given to how well the outcome meets all of its requirements. How the outcome could be improved if the problem was revisited is discussed but consideration given is limited. Independent feedback obtained of a useful and realistic nature but is not evaluated and discussed in a meaningful way, if at all.	3	
2	The outcome is discussed but not all aspects are fully addressed either by omission or because some of the requirements have not been met and those requirements not met have been ignored in the evaluation. No independent feedback obtained or if obtained is not sufficiently useful or realistic to be evaluated in a meaningfully way even if attempted.	2	
1	Some of the outcomes are assessed but only in a superficial way. No independent feedback obtained or if obtained is so basic as to be not worthy of evaluation.	1	
	No evidence presented	0	**Mark awarded:** 3

Total mark 65/75
Concluding comments: An advanced, robust project that demonstrated the candidtae's ability to program for an advanced mathematical topic despite not studying AS/A-level further maths.
Signed:

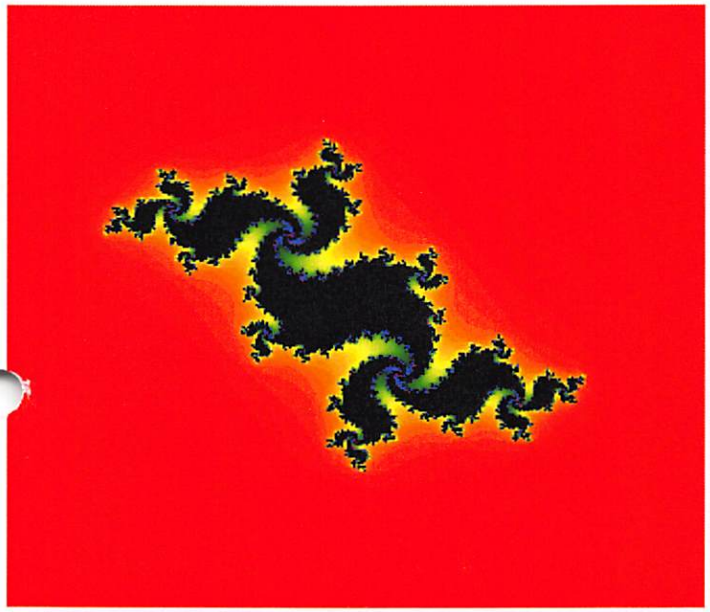
To see how we comply with the Data Protection Act 1998 please see our Privacy Statement at aqa.org.uk/privacy

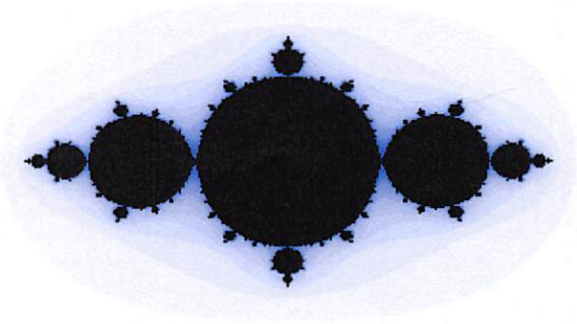
NEA Commentary Sheet

CENTRE No.				
CANDIDATE No.				
Programming Language	Pascal			
Title / Type of System	Julia Set Generator			
Analysis	**Comments**	**Page**	**Mark**	**Max**
Problem identification	A program to aid teaching how Julia sets are produced.			
Research methodology	Background maths: p2. Evidence for interaction with end user over objectives given p7 onwards.			
User(s) identification/requirements	Quite general. Surprisingly little detail about the main end user who has requested this software. This project was previously attempted by a previous CS student with quite limited success - the hope was that a stronger candidate this year would produce useable software for AS Further Maths extension workshop. Student obviously not given access to previous candidates' work. This student has gone beyond the end user's requested scope in order to explore the topic themselves and show a display of the sets produced.			
Background to the problem	Maths explained briefly.			
Data and processes	ADD on p5.			
DFDs	Current L0&L1 p8, proposed L0&1 p9.			
Num objs (What/How/When or SMART)	Lacking some of the early detail - focussed on advanced features, hence less than completely SMART at first. Candidate goes into more detail on p6-8.			
Proposed solution	Discussed.			
Models (E-R, Objects, Networks)	No OOP model produced.			
			8	9
Design				
Overall system design	Detailed flowchart p12. Shame IPSO missing.			
Design Data Dictionary	P23.			
Record/Database structure				
File sizes/Data Volumes				
Modular structure	Given p14.			
Required built in modules and structures				
Algs for data transformation	Pseudocode - p15-22. Very detailed and explained well.			
User interface (if appropriate)	Given with justification p28. Could really benefit from HCI designs showing Julia Set fractal output. But also crucially missing the diagram showing the iterative steps taken by each coordinate point to teach how the colour of each pixel is determined - this is needed to fully satisfy the end user even though it is much less complex! This is later discussed and coded - see p50 in technical solution.			
Storage media & format				
Security & integrity of data	Discussed.			
System security	Discussed.			
Validation incl errors				
			11	12
Technical Soln				
Completeness of Soln				
Objectives/requirements met	Mainly, although still some aspects the end user was not completely satisfied with.			
			12	15
Techniques used				
Group A algorithms used	Very advanced mathematics including powers of complex numbers. Z path now covered (notably missing detail in the design section). Description of other features summarised on p29.			
Group B algorithms used	Could have attempted OOP style, but did not.			
Group C algorithms used				
Self Documentation/Annotation (Basic)	Yes.			
Coding Style: Good characteristics	Well structured. Identifiers not always sensible.			
Coding Style: Excellent characteristics	Some.			
Overview guide (incl. setup)	P89 onwards.			
			23	27
System Testing				
Introduction & overview	Well considered test plan and overview.			
Test plan linked to objectives	No.			
Representative sample of test data	Yes.			
Expected and actual results annotated	Yes.			
Robustness/are there missing tests	Failed tests: 4.12, 4.22, 5.3, but code edits included after and re-tested. Other issues identified by end user in evaluation.			
			8	8
Appraisal				
Independent feedback	Yes			
Feedback evaluated	Yes			
Objectives evaluated	Objectives made more SMART at this point			
Further development?	Very brief. Doesn't address outstanding end user requirements.			
			3	4
Total			**65**	**75**

N.B. Some of these sections will NOT be appropriate for certain types of project, especially non-Data Processing ones. Not all of them may need to be completed. In all cases, you MUST refer to the Full Assessment Criteria if using this grid.

Agreed





Julia Set Generator

Contents

Analysis	Pages 1 - 11
Design	Pages 12 - 28
Technical solution	Pages 29 - 95
Testing	Pages 96 - 129
Evaluation	Pages 130 - 133

Analysis

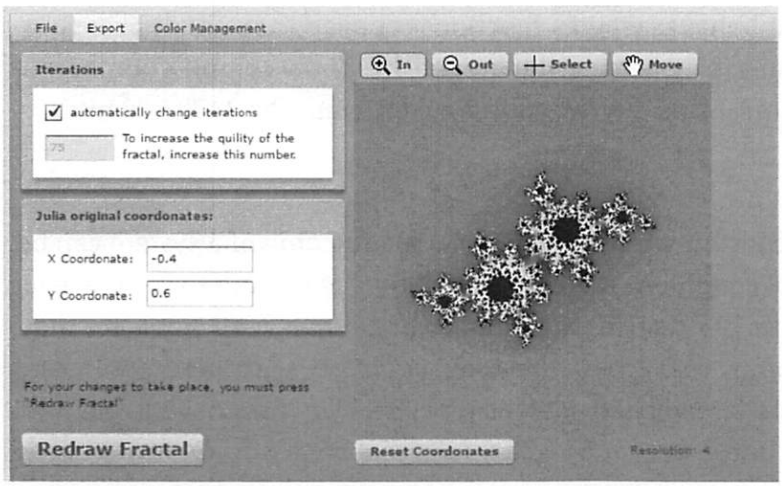
Background to identification

I have been doing maths for almost all of my life and have been fairly interested in most of the mathematical concepts. However I have seen very little of this maths applied which causes anything of great significance. One concept I have found interesting however, is the mysterious behaviour of complex numbers and the beautifully generated patterns that they are capable of producing. Complex Numbers seem to behave in a really chaotic manner when combined with a specific function, with the end result being these beautiful never ending fractals. One set of fractals in particular I want to investigate are Julia sets, produced specifically by functions like $F(Z) = Z^2 + C$, because this type of function Is capable of producing vast amounts of these fractals, all with their own unique features. Overall I would like to produce a program which is capable of generating these Julia sets, allowing the user to input their own values which would in turn alter the Julia set generated. I would also like to investigate into creating a system that not only generates the set but also shows how these Julia sets are generated and what gives each point inside it their unique colour.

Description of the current system

The current system used in most programs which generate Julia sets, are very common. The general system starts off with the user first inputting the complex number of their choosing, which acts as the c value, in other words the constant. The program then runs the calculations on the Complex number provided by the user and in turn generates the set. Some systems have extra features which give the user options like changing the colour scheme of the Julia set. What these programs on the other hand fail to do, is give the user a basic idea of how these Julia sets are generated, like for example what gives a point its specific colour. This is what I hope to include in my program which will hopefully put it above other Julia set generators.

Here is an example of a Julia set which uses the current system as you can see it allows the user the choose the constant, number of iterations and manage the colours but it does nothing to inform the user on how the Julia set is actually generated.



This Julia set generator came from http://www.easyfractalgenerator.com/julia-set-generator

How the calculation works

A Julia set is an image generated from a function like F (Z) = Z^2 + c where c remains constant throughout the entire procedure. The value of Z and C on the other hand are not always real numbers, the complex plane consists of both real numbers (X-axis) and imaginary numbers (Y-Axis), which act as coordinates causing them to form complex numbers, with the first number being the real part and the second number being the imaginary part. For Example, if you had the complex number 1 + 0.25i, 1 is the x coordinate and 0.25i is the y coordinate. This complex number is then squared and the Constant C is then added onto the squared complex Number and this is repeated several times. The reason this process is repeated several times is, because the program will only plot the point depending on whether it converges on a specific point or spirals in a loop, if this happens the point is in the Julia set and will be plotted black. If it does not spiral or converge on a point and instead escapes towards infinity, the complex number is not in the Julia set, and will be assigned a colour other than black depending on the number of iterations it takes to escape the radius of 2.This entire process will then be repeated for every single complex number inside the radius of 2, which will eventually produce the Julia set.

Source of research https://www.youtube.com/watch?v=2AZYZ-L8m9Q

Identification of the prospective users

Julia sets in general have few uses to the general public but for those involved in mathematical subjects and even art have vast amounts of uses for Julia sets and fractals.

Complex Numbers

If a person involved in maths has no previous knowledge of complex numbers and the complex plain, Julia sets could be a great way to demonstrate how complex numbers work in general, giving a general understanding of how to do calculations with them. For example a student could run several calculations on the equation $F(Z) = Z^2 + C$ with the same c value. They can then plot these points on a graph displaying the complex plain, which will eventually display how this point behaves when a function is applied to it. This makes Julia set generators great teaching tools for teachers who are teaching how complex numbers works to high level maths students. https://www.youtube.com/watch?v=2AZYZ-L8m9Q

Chaos Theory

Chaos theory is the theoretical concept which explains that an outcome of a system can be significantly changed if the initial conditions are altered slightly. If you were to set the value of C to 0 + 0i then you would get a basic circle with a radius of 1. However if you were the change this complex number slightly to 0.25 + 0i then you would get a completely different Julia set compared to the one you already had. Instead of a circle you now have a Julia set which resembles the shape of a butterfly. Another way Julia sets are associated with chaos

theory is the chaotic behaviour which occurs when the Z points inside the complex plain are applied to a function. The points behave very chaotically, going from each newly calculated point to the other, until it either escapes towards infinity, converges towards a specific point or even spiral in an infinite loop. This represents chaos theory because of the chaotic behaviour the points have when a function is applied to them.

Art

Julia sets and fractals in general can be also be used in some forms of art called fractal art, A form of digital art which is created by calculating and generating these fractals and then displaying them as still images. Julia sets can be used in this way by selecting a value of c which in turn generates the Julia set with all its unique characteristics and features.

Identification of user needs and acceptable limitations

The program overall has quite a few user needs which are absolutely essential to be provided to the user for the best possible experience. All of these needs are essential to all users whether or not they are using it for mathematical or artistic purposes.

Generated Julia Sets

My program must take 2 things into account which are essential for generating the Julia sets, performance and quality.

The quality of the generated Julia sets is very important because this is actually the base of the program. If the generated image is distorted and pixelated then it will not appeal the user because the Julia set produced will not be able to display its unique features and characteristics which make it different to other Julia sets.

Performance overall depends on two things such as the user's computer specifications and the quality of the generated Julia set produced. I'm expecting due to the huge amount of points being plotted a large portion of the computers CPU will be focused on producing the Julia set. So the image produced needs to be as efficient as possible so not so much of the CPU is used by the program. The efficiency of the program can be improved by limiting the amount of iterations the Julia set can handle and produce or better still give the user the option to change the amount of iterations they want, based on how powerful their computer is, this is however at the cost of quality.

There are also some other obvious and essential user needs. The user must be allowed to enter the complex number of their choosing which will represent the constant. This will be followed by a validation system which prevents any invalid inputs such as strings and character data types from being type in.

Data sources, destinations

Inputs

The first noteworthy input would definitely be the constant which is absolutely essential for the entire program to work. This value can be split into two different types of data which are known as the imaginary number and the real number, which is entered into the system by the user themselves. This is done by providing a box to type into for both the imaginary and real number.

The z-coordinates are also an input which is essential for the calculation of the Julia set. This value comes from inside the program via a component called TColorMapSeries which can be used to go through every single coordinate in the complex plain and temporarily store it in variable we will call the ZValue.

We also need to specify the maximum number of iterations the program is meant to handle. This data is entered automatically into the system by being assigned a value of 100. However this value can also be changed by the user, so the system is able to receive this data from the user if the user chooses to.

Outputs

The outputs involved in my program are mainly used to generate the actual Julia set. A Variable is required to store all of the coordinates and also the colour which is assigned to that point based on the number of iterations it has taken. This piece of data will then be used to output the coloured points on the screen as an interactive image due to the fact the user needs to be given the opportunity to zoom in or move around the fractal.

Further down you will see the objectives my end user wants me to include and one of them is to produce the path of a user-defined Z-coordinate. A variable is required to store the user defined Z value and the newly calculated Z values and display these points as a graph. I also need to display these new found coordinates next to the graph making the newly found points more clearly to the user.

Data volumes

Each Z-Coordinate used in the program is operated on multiple times producing many new Z-Coordinates until It leaves the complex plain or spirals in a loop or converges on a point. This means the Iterations value is going to be incrementing by 1 drastically because of the amount of times a function is used on a point. The amount of times the iterations is incremented by, depends on the Z-Coordinate and the constant.

There is also a large amount of Z-Coordinates required to be stored because the program needs to use every single Coordinate inside the radius of 2.

In the objective section my end user would like me to produce an animation which shows the user how Julia sets change with a changing constant. This means the constant is going to be changing every interval until the animation stops.

I do not expect my programs size to be very large because it will have no need for databases so I think it should be at least under 50MB. I however expect it to take up a lot of the computers CPU due to the large number of points which need to be operated on in order to produce each set.

Analysis data dictionary

Data (Input / Output)	Use
Imaginary Number (Input)	This is the imaginary part of the constant for the function$F(Z) = Z^2 + C$. If an invalid input is provided by the user then this value should be automatically assigned the value 0 so the program doesn't crash.
Real Number (Input)	This is the Real part of the constant for the function $F(Z) = Z^2 + C$. If an invalid input is provided by the user then this value should be automatically assigned the value 0 so the program doesn't crash.
Z Coordinates (Input)	This piece of data represents the Z value of a function like$F(Z) = Z^2 + C$. This data will then be operated on by this function and will carry on to do so from 1 to many iterations until the Z value leaves the radius of 2. This Function is Applied to every Single Z value within the complex plain.
Iterations	This value will be used to find out the amount of times the function is used on the Z value. My program will then use this data to determine whether this point is in the Julia set or not. If not, the number of iterations determines the colour of the point.
Maximum Iterations(Input)	This value will be used to define the maximum amount of iterations the calculation will go to before its stops if Z has not escaped the complex plain. This will also determine the overall quality and depth of the image. If an invalid input is provided by the user then this value should be automatically assigned a reasonable value so the program doesn't crash.
Julia Set(Output)	All of the coordinates in the complex plain need to be stored somehow so they can be used to generate an image of the Julia set. This value also needs to store the amount of iterations each point goes through and hence the colour of each individual point because this is what gives the generated Julia set its own unique features.

Object analysis diagrams- inheritance, aggregation

Numbered general and specific objectives of the project

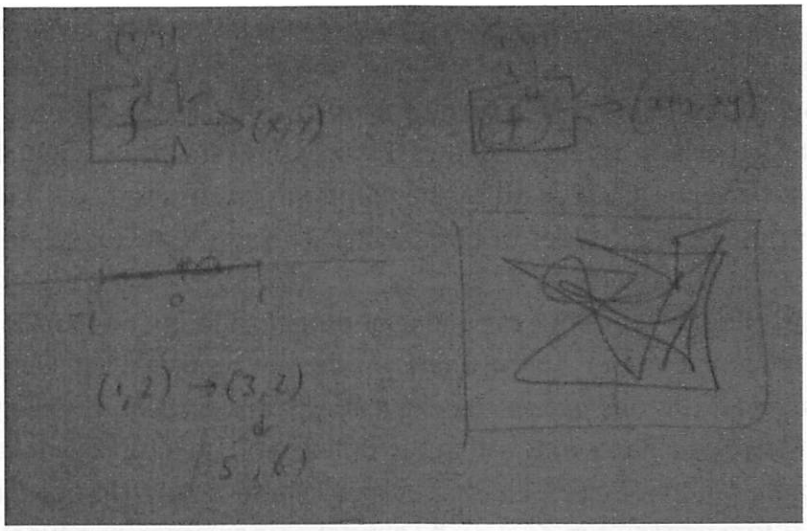
This section of my analysis will include objectives I have thought of myself and have discussed thoroughly with my end user as well as her also giving me her own objectives which she wants the program to achieve so it becomes more useful specifically towards her.

1. The first objective I personally would like to achieve would be to give the ability to the user to change the colour scheme of the Julia set they produce. This is not essentially an important feature but it does add more creativity and interactivity to the program causing the user to become more engaged with it.
2. My program must give the user the ability to choose their own complex number for the constant. This means giving them the freedom to choose whatever number they desire including decimal, negative and positive numbers.
3. My program must have a solid validation system which prevents anything invalid from being used within the program. This involves instances where strings and characters are being entered instead of integers and real numbers, and also when values are left blank, meaning the value is going to have to be set to something by default.
4. My Program must also allow the user to zoom in on the generated image. This is essential to the program because each generated Julia set needs to show the user its depth displaying how the Julia set journeys towards infinity.

Objectives my end user has discussed with me

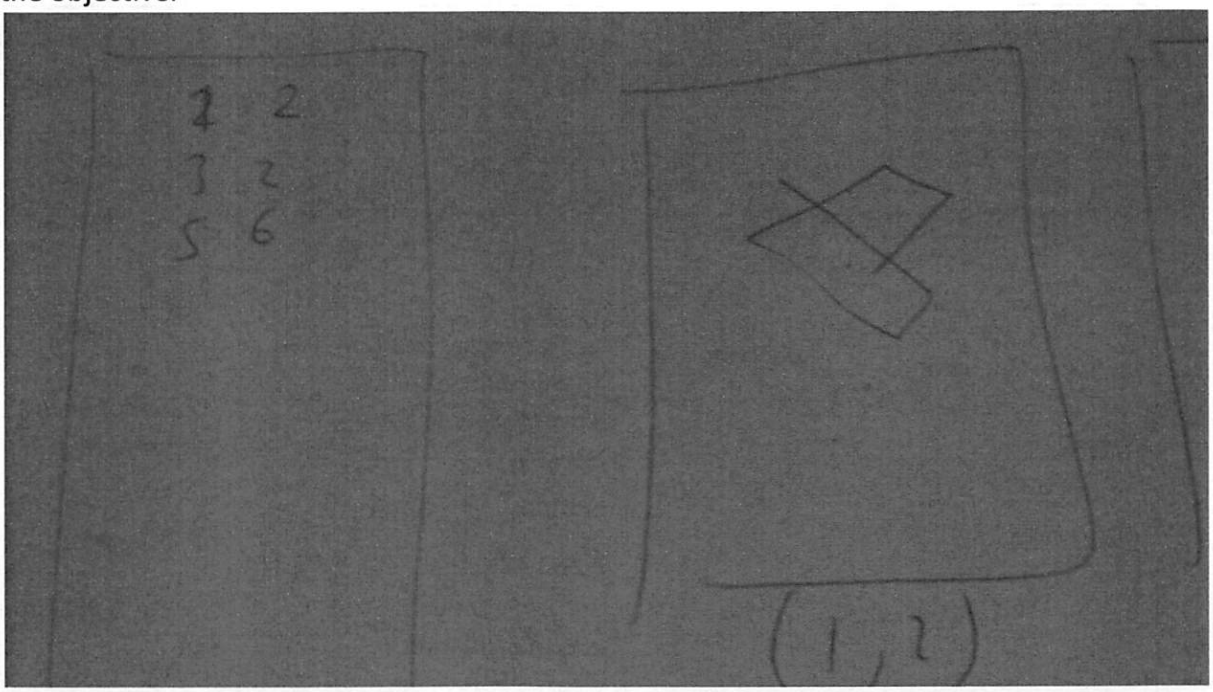
My end user, profession is an A-level maths teacher who liked the overall idea of my project but due to her profession, she wanted it to have more focus on the educational sides of Julia sets and how they are formed from the functions used in the calculations. When I consulted her about my project, it did not take place as an interview but more like a discussion where she would share some of the objectives she would like to see done in my project and discuss ways on how the overall problem could be solved.

1. The First Objective she was fairly eager to see in my project was a feature which would show the path a specific Z value takes to escape infinity, Converge on a particular point or even spiral in a never ending loop on a graph created when a function is applied to it with an already defined constant. This will also involve the user picking the specific Z value that they want to see the path of. The reason Mrs wanted me to include this in my program is because she wants her students to understand what gives a Julia set its features, in particularly the specific colour of the points which are determined by the number of iterations it take for a point to escape the complex plain.
   Here is a rough Idea and design of what my end user wants.



This rough sketch was drawn by my end user during the discussion. As you can see, this sketch shows the Z coordinates being determined by the user and then being applied to the function which in this case is labelled F. The diagram above displays a very chaotic graph which has been drawn to give an example of what the chaotic path of the Z value could be (Note this will only be done for the function $F(Z) = Z^3 + C$due to the program mainly being focused around this function.

2. As a little extension to the first objective my end user would like my program to display a list of all the points which are on the path. The reason my end user wanted this in was to make the points more clearly to the user. Below is evidence to support the objective.



The box at the side is used to represent the points which are present in the Z coordinates path.

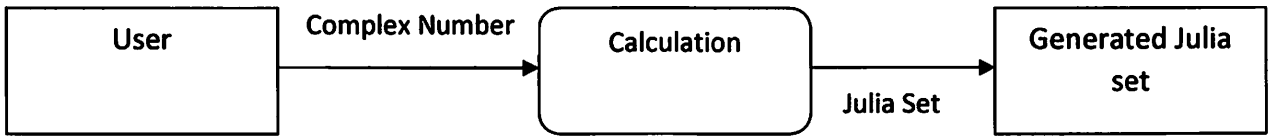
3. The third objective for my project is a feature which shows an animation of the variations of Julia sets when the imaginary or real number is changed by say 0.1 per unit time. I presented this idea to my end user saying it would show a good example of chaotic behaviour because it shows the user how drastically the Julia set can change if the values entered are slightly different. I consulted about whether this was a good idea or not to which she agreed it was worth including.

4. My fourth and final objective my end user proposed to me was to allow the user to change the function used on the points inside the complex plain. I am a little sceptical about this one because I believe it will present a bit of a challenge and whether I will be able to do it within the time limit. I do intend to do it but right now I think of it as more of an extension because I am focused right now on one particular function$F(Z) = Z^2 + C$. If I am not able to program a function which allows the user to change the function, I will definitely include already programmed functions such as $F(Z) = Z^3 + C$ to show the user how the Julia set changes with an increasing power.

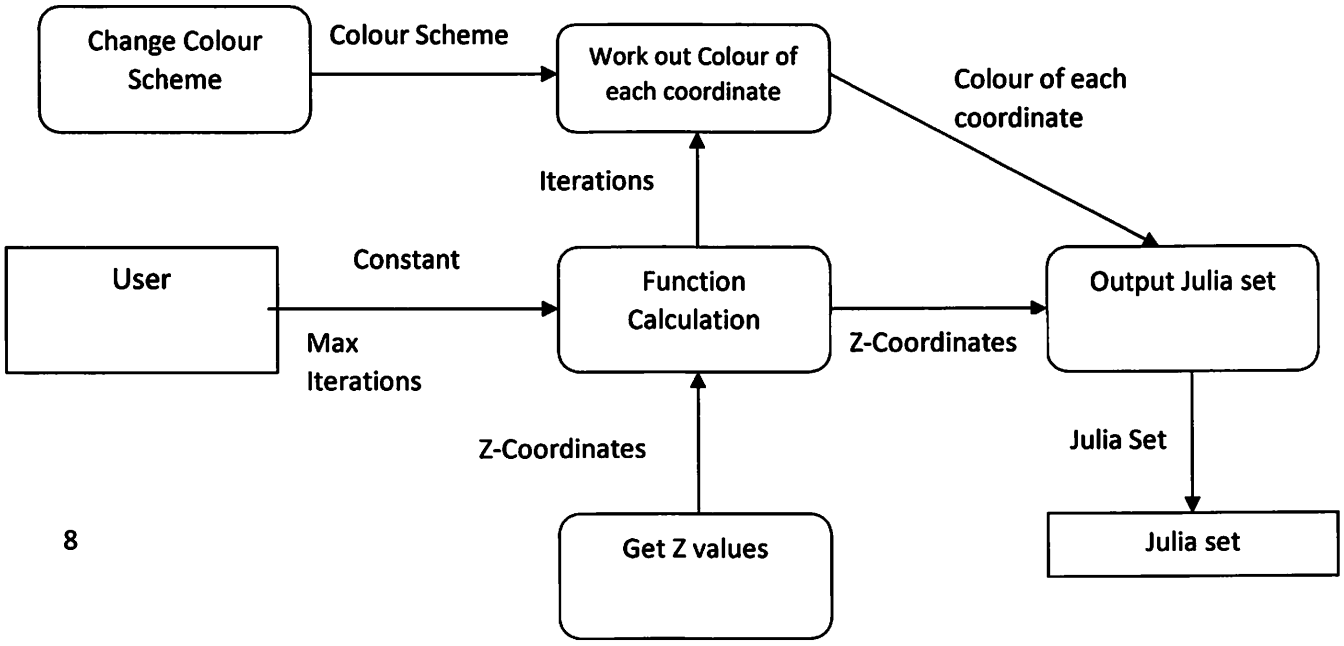
Data Flow Diagrams

This section will contain level 0 and level 1 data flow diagrams of both the current and my own proposed system which I intend to use.

Current System Level 0



Current System Level 1



Realistic appraisal of the feasibility of potential solutions

Non-Computerised solutions

- It is possible for someone to work out the number of iterations of all the majority of the Z values inside the radius of 2. Despite the fact this is possible, it would take a considerable amount of time for the user to actually perform the function multiple times on every single Z value making this solution very inefficient.

Of the shelf software solutions

- There are free online Julia set generators which allow the user to generate Julia sets and also change the colour scheme of each point. These online pieces of software on the other hand very rarely offer the user the choice to view the Z path of a user defined Z coordinate explaining what gives its properties and colour meaning the user may have to do it manually. An example of an off the shelf Julia set is from http://www.easyfractalgenerator.com/julia-set-generator.

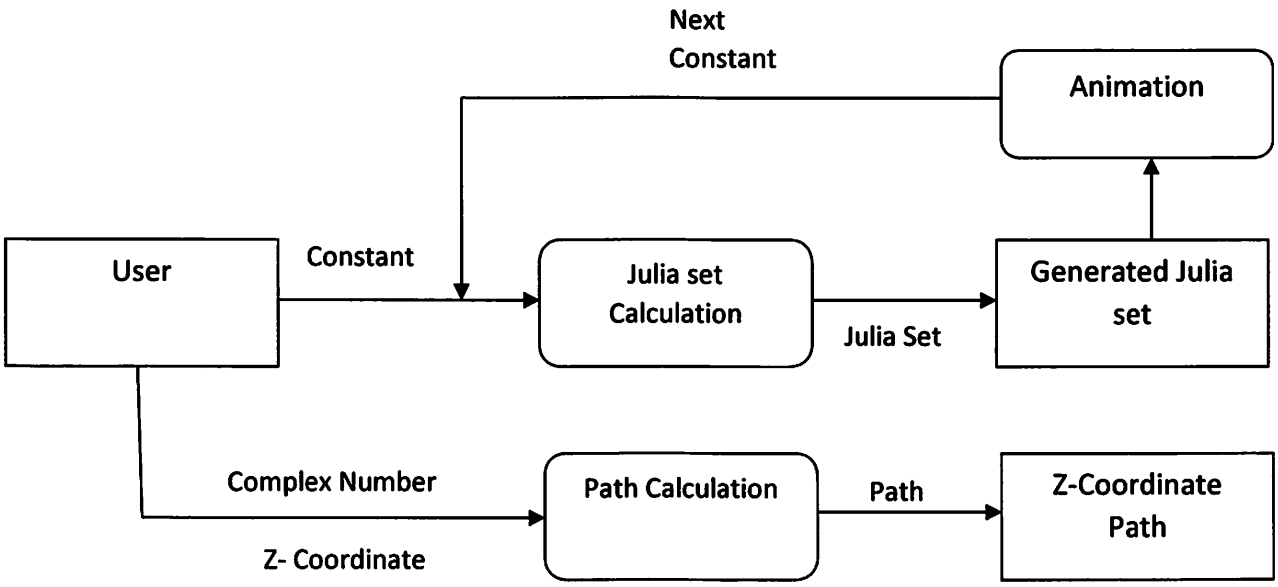
Bespoke software solutions

- Bespoke software is essentially what I aim my overall solution to become. Bespoke software is basically software which is designed for a user's specific needs, so in my case, my end user would like my program to produce a Z path for a user defined constant and Z value, so the user could receive a better understanding about how point has been operated on by the function and why it has been assigned a certain colour as well as the generation of the Julia set. The user also requires the program to produce an animation of a changing Julia set with a changing constant. All of these extras my end user has asked me to include, overall show how many user specific objectives my program needs meaning the final program will need to be a form of bespoke software.

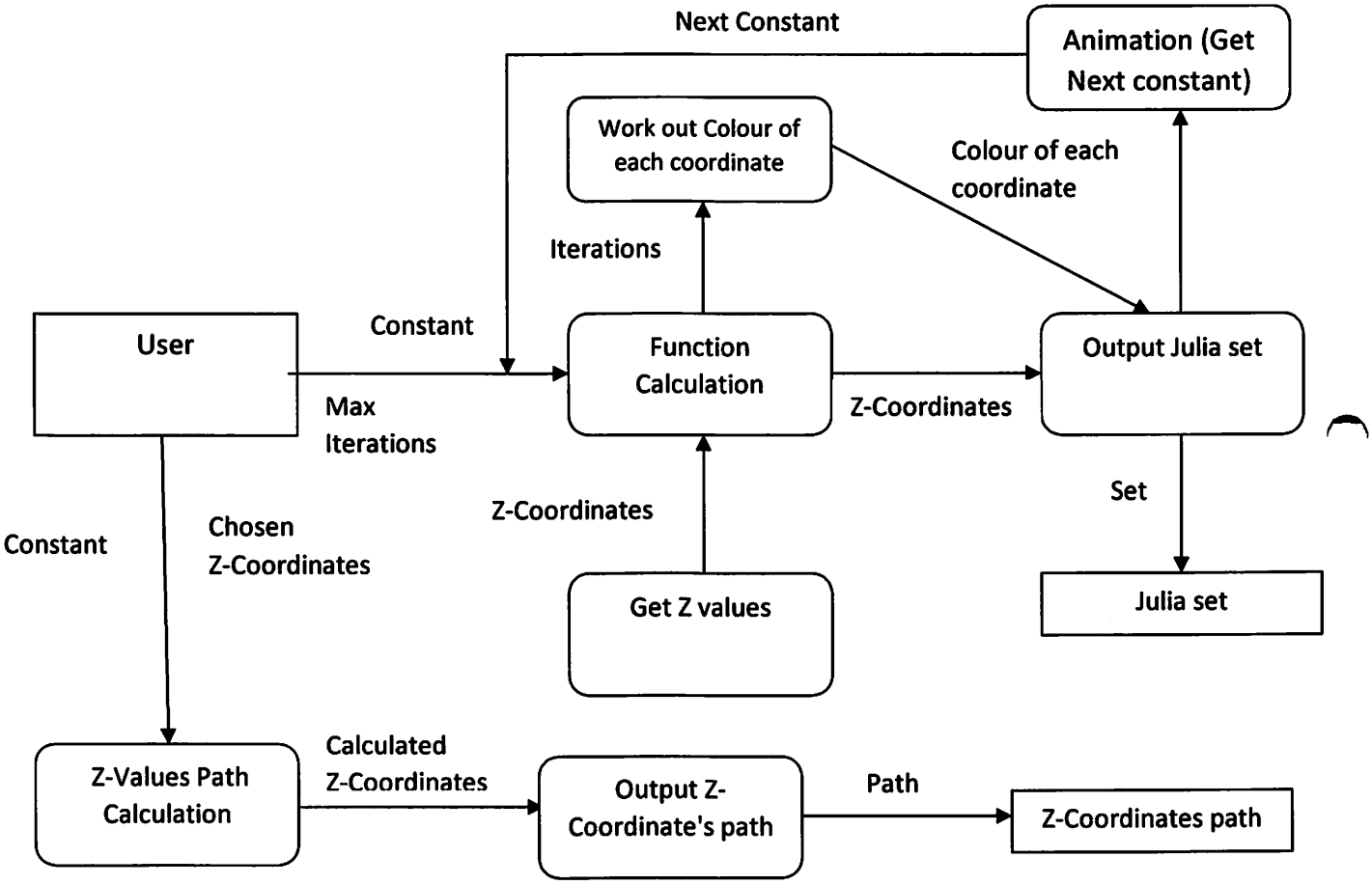
Justification of chosen solution

A piece of software I am already familiar with is Lazarus, which is a compiler designed for pascal and Delphi which also contains many different features for many different things. I knew immediately it needed a feature which is able to run graphs otherwise my program would not work. I came across a feature called TColorMapSeries which I believed could produce the fractals that I was looking to produce. After Researching the Component, from this source http://wiki.freepascal.org/TAChart_Tutorial:_ColorMapSeries,_Zooming it gave an example of a use of this component in the form of producing the Mandelbrot set. As the Mandelbrot set is a fractal its self I was now fully aware that this component is capable of producing fractals and more importantly Julia sets. After looking into TColorMapSeries more thoroughly, I found out that this component has three Parameters in the form of colour and the x and y coordinates. This is perfect because these 3 properties are exactly what I need to display these fractals. Lazarus also has other useful components in which I can use for other purposes like for example using a feature called TEdit for entering input data such as the constant and the maximum iterations the program is allowed to run. It is also very easy to debug because the Pascal and Delphi have very modular programming structures, so it is much easier to identify and solve the problem. The only problem I can see with using this sort of software is that the language it uses, Pascal is a very old language so the actual performance of the program may be overtaken by the performances of software which use more modern languages such as C# or Java.

Proposed System Level 0

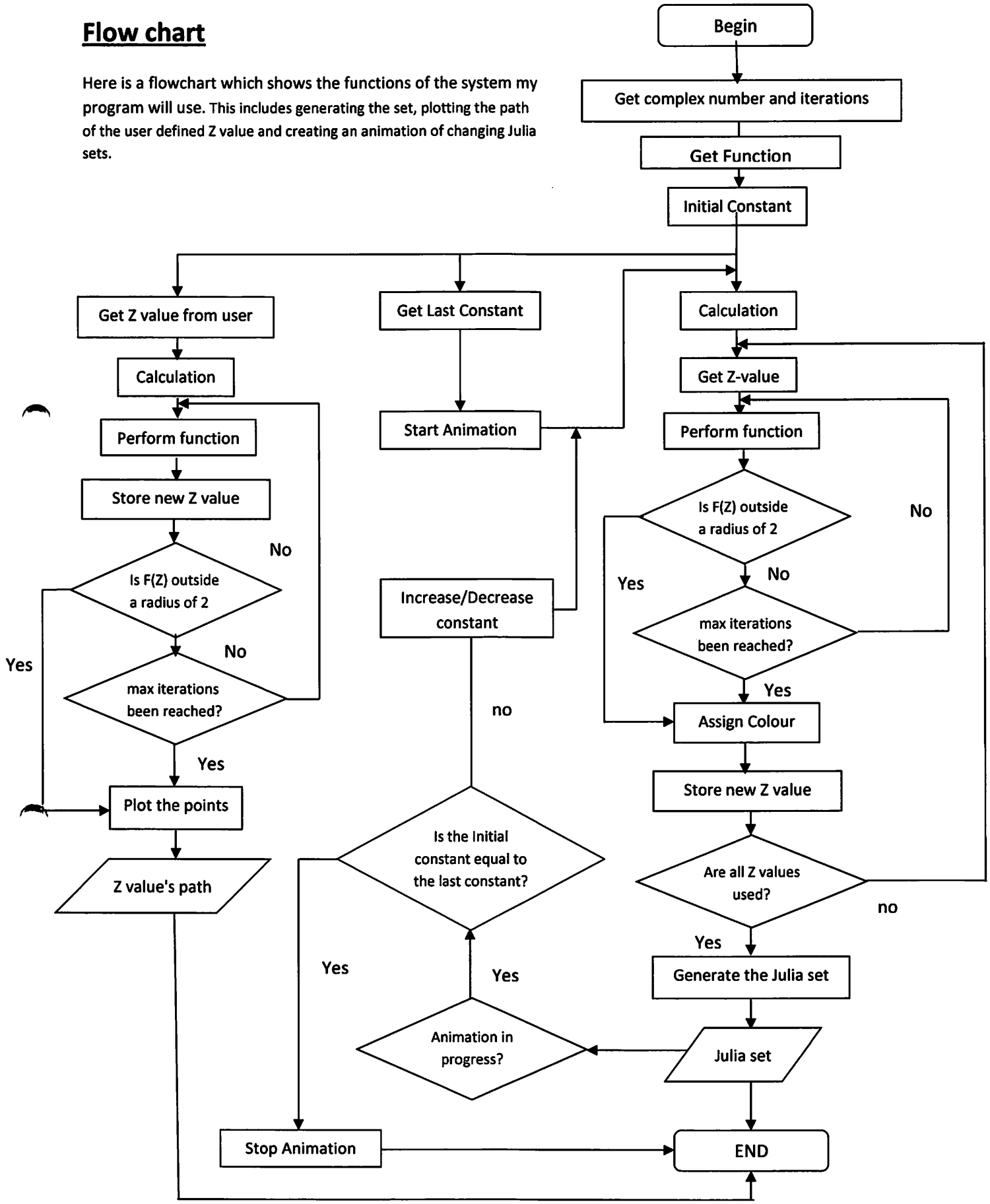


Proposed System Level 1



Flow chart

Here is a flowchart which shows the functions of the system my program will use. This includes generating the set, plotting the path of the user defined Z value and creating an animation of changing Julia sets.



Design

As you have ciphered from my analysis, my main project objective is to produce a program which calculates and generates Julia sets with user defined constants as well as other features which give the user an idea of how these sets behave and also how they are formed, by allowing them to produce animations of changing Julia sets and also to be able to produce paths of individual Z values. This section focuses on all of the necessary steps I need to take in order to take my project to the next step, by actually creating the code and the interface for it. The steps in this section, involve diagrams such as flow charts and structure charts which are used to give me a clear idea of how the final program will function and how the data will travel within it. This section also includes a rough prototype of the user interface, potential data that may be required for the program to function, devices needed to run and store the program and most importantly a draft of the code so I can get an idea of how to program the functions involved with generation of these features.

Flow chart description

There are 3 main sections to this flow chart, the generation of the Julia set, the generation of the used defined Z value's path and the animation which shows the variation of Julia sets with a changing constant.

Julia Set Generation

The first section of the flow chart is the part which shows how the main Julia set is generated. This section starts off by obtaining the initial constant from the user, which is then substituted into the user's chosen function as the value C or in other words the constant. The program will then obtain a value of Z and will sub it into the function. The function is then ran over multiple iterations until it has either escaped the radius of 2 or has reached its maximum number of iterations. The program will then assign a colour to that point based on the number of iterations the function has taken. If the number of iterations is equal to the maximum number of iterations then we will assign this point the colour black as it is inside the Julia set, If not it is set different colour. This Z value is then stored and the program then determines whether all of the points in the radius of 2 have been used, if not the program will repeat the previous 5 steps, except with a new Z value. This is repeated until all of the points in the radius of 2 have been used. Once all the Z values have been used the program is now able to generate set on screen, showing all of its unique characteristics.

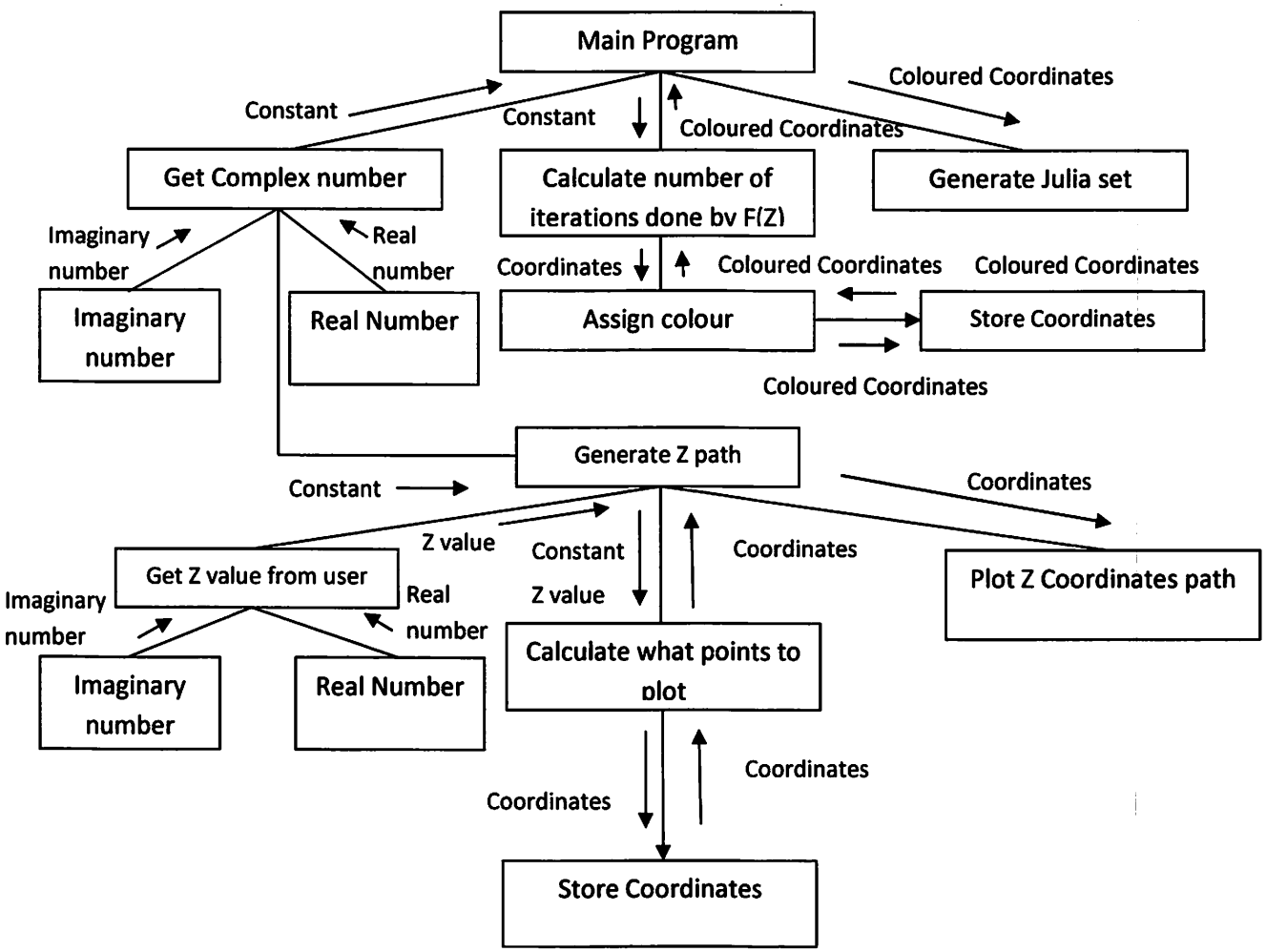
Z Path Generation

The Z path generation feature starts of exactly the same as the Julia set generation as the program first obtains the constant from the user. The program then obtains the user defined Z value from the user and then subs it into the function. NOTE, this feature only works with the function$F(Z) = Z^2 + C$. Once the Z value has been operated on by the function, the newly found Z value is stored. The function then continues to operate on the newly calculated Z values until the maximum number of iterations has been reached. Once all of the points have been collected the Z path can now be generated and displayed on screen.

Animation

Like the other 2 sections this part starts off by obtaining the constant given by the user. However unlike the Z path generation, the animation works for all three built in functions, so the program then needs to get the function chosen by the user. The program will then generate the Julia set for the initial constant given by the user for a certain amount of time. The program then checks to see if the constant is equal to the last constant, if not the program either increases or decreases the constant based on whether it is larger or smaller. It will then generate the set for all of the new constants in turn for a brief amount of time until the constant is equal to the last constant meaning the animation has now stopped displaying the Julia set of the last constant.

Structure Chart



Algorithms using structured English or pseudo code

This section of the design will show some of the algorithms I have decided to use and base the final code on. Each of these algorithms are written in pseudo code which I will transform into my chosen language Delphi.

Function / procedure description	Code and notes

Algorithm for generating the Julia set. NOTE There will be three of these(One for each function);	Procedure Chart1ColorMapSeries1Calculate(cost AX, AY: double; out AZ: double)
	// This function is specific to Lazarus. What it does is go through every single coordinate on the graph and then allows the following code to perform operations on them.
	var
	Iterations: Integer// This is used to keep track of the amount of times the function is applied on the constant
	ZCoordinate: TDoublePoint//This is used to temporarily store each of the Z values on the complex plain
	Begin
	ZCoordinate = DoublePoint(AX, AY)// This temporarily stores each coordinate in the variable,ZCoordinates
	JuliaSetMaths (Iterations, ZCoordinates)// This is the function which performs the maths on the coordinate (Description below)
	If Iterations = IterationsMax + 1 then // If the number of times the function is used is equal to the maximum amount of iterations then
	AZ = -1 // The colour of that point is assigned a black colour
	Else
	AZ = Iterations / IterationsMax// If the number of iterations is below the maximum then a colour is generated based on the amount of iterations the Z coordinates have. AZ is what stores the colour of a point.
	End;

This function is used to determine the whether a point is within the Julia set or not for the funtion $F(Z) = Z^2 + C.$	Procedure JuliaSetMaths (out Iterations: integer; z : TDoublePoint)// This function has 2 parameters. One being the iterations parameter which is used to record the number of times the function is used on the constant and z coordinate. The other parameter is the Z variable which is used to store the current Z coordinate which will be used in the function
	begin
	Iterations = 0// The number of iterations must always start at 0 every time this function is called as the function has not been used yet.
	repeat
	Z = DoublePoint(sqr(z.X) - sqr(z.Y), 2 * z.X * z.Y)// this is the function performing on the Z value and the constant
	Z = DoublePoint(RealNumber3 ,ImaginaryNumber2)+ Z
	Iterations = Iterations + 1;// The function has been used so iterations is incremented by 1
	until (Iterations >IterationsMax) or (sqr(z.X) + sqr(z.Y) > Limit)// If the Result of the function is outside the radius of 2 or the number of iterations has reached its maximum limit then the repeat loop ends.
	end;

Function / procedure description	Code and notes

This function is used to determine the whether a point is within the Julia set or not for the function$F(Z) = Z^3 + C$.	procedure Function2(out Iterations: integer; z: TDoublePoint);
	begin
	Iterations := 0;
	repeat
	Z := Doublepoint((z.X * z.X * z.X) - 3 * (z.Y * z.Y * z.X), -(z.Y * z.Y * z.Y) + 3 * (z.X * z.X * z.Y))
	Z := DoublePoint(RealNumber3, ImaginaryNumber2) + Z
	Iterations = Iterations + 1
	until (Iterations > IterationsMax) or (sqr(z.X) + sqr(z.Y) > Limit)
	end;

This function is used to determine the whether a point is within the Julia set or not for the function$F(Z) = Z^4 + C$.	procedure Function3(out Iterations: integer; z: TDoublePoint);
	begin
	Iterations := 0;
	repeat
	Z := Doublepoint((z.X * z.X * z.X * z.X) - 6 * (z.Y * z.Y * z.X * Z.X) +
	(z.Y * z.Y * z.Y * z.Y), -4 * (z.Y * z.Y * z.Y * Z.X) + 4 * (z.X * z.X * z.Y * z.X));
	Z := DoublePoint(RealNumber3, ImaginaryNumber2) + Z;
	Iterations := Iterations + 1;
	until (Iterations > IterationsMax) or (sqr(z.X) + sqr(z.Y) > Limit);
	end;

Function / procedure description	Code and notes

This function is used to obtain the constant Entered into the program by the user and convert it into a real number. It is also used to make the selected series generate the Julia set.	procedure GenerateSet(Sender: TObject)
	begin
	if ZSquared = True then // If the user wants to use the function $F(Z) = Z^2 + C$ then the following statements occur
	begin
	Chart1ColorMapSeries1.Active = False // All of the graphs used for displaying all the Julia sets with different functions are set to false.
	Chart1ColorMapSeries2.Active = False
	Chart1ColorMapSeries3.Active = False
	if Edit5.Text = '' then//If the user has not entered a value for the number of iterations
	Edit5.Text = '100'// It is set by default to 100;
	RealNumber = StrToFloat(Edit1.Text)//This is used to convert the real number entered by the user as a string into a real data type
	ImaginaryNumber = StrToFloat(Edit2.Text)//This is used to convert the Imaginary number entered by the user as a string into a real data type
	IterationsMax = StrToInt(Edit5.Text) //This is used to convert the maximum number of iterations entered by the user as a string into a real data type
	Chart1ColorMapSeries1.Active = True// Once the button used to generate the set is clicked the set will be generated so value needs to be changed to false in order for it to generate.
	if RadioButton2.Checked = True then // If the user wants to use the function $F(Z) = Z^3 + C$ then the following statements occur
	begin
	Chart1ColorMapSeries1.Active = False
	Chart1ColorMapSeries2.Active = False
	Chart1ColorMapSeries3.Active = False
	if Edit5.Text = '' then begin
	Edit5.Text = '100'
	endIf
	Chart1ColorMapSeries2.Active = True//The only difference this section has to the above section is that the series for $F(Z) = Z^3 + C$ is set to true
	RealNumber = StrToFloat(Edit1.Text)
	ImaginaryNumber = StrToFloat(Edit2.Text)
	IterationsMax = StrToInt(Edit5.Text)
	end;
	if RadioButton3.Checked = True then// If the user wants to use the function $F(Z) = Z^4 + C$ then the following statements occur
	begin
	Chart1ColorMapSeries1.Active = False
	Chart1ColorMapSeries2.Active = False
	Chart1ColorMapSeries3.Active = False
	if Edit5.Text = '' then
	Edit5.Text = '100'
	Chart1ColorMapSeries3.Active = True //The only difference this section has to the above sections is that the series for $F(Z) = Z^4 + C$ is set to true
	RealNumber = StrToFloat(Edit1.Text)
	ImaginaryNumber2 = StrToFloat(Edit2.Text)
	IterationsMax = StrToInt(Edit5.Text)
	end
	end
	end

Function / procedure description	Code and notes

If the user clicks on the box used to enter the values of the imaginary and real number and enters a string or does not enter a value at all, that value will be set to zero. This procedure is used multiple times for every text box which obtains data with integer and real data types.	procedure ValidateReal(Sender: TObject)
	begin
	if Edit1.Text = '' then begin //If the user enters nothing the value will be Zero
	Edit1.Text = '0'
	end;
	M := leftstr(Edit1.text,1)[1] //M is assigned the first character in the text box
	C := Ord(M)//M is transformed into its ordinal value
	Length1 = Length(Edit1.text) // This works out the length of the string
	If ((C) > 59) or ((C) < 45) or (length1 > 1) then begin // The ordinal value for -9 to 9 is 41 to 59 so any other characters typed in will be replaced by the value 0;
	Edit1.Text = '0'
	end;

This procedure is used to decrease the value of the real number, Maximum number of iterations and the imaginary number by a certain amount by clicking a button. This type of procedure will need to be replicated in the code several times for each of these variables.	procedure DecreaseReal(Sender: TObject)
	begin
	Chart1ColorMapSeries1.Active = False//The graph stops generating the set for a moment
	RealNumber = StrToFloat(Edit1.Text) - 0.05// The appropriate variable (In this case the real number) is decreased by 0.05
	Edit1.Text = FloatToStr(RealNumber3)// The real number in the box where you enter the value is updated
	Chart1ColorMapSeries1.Active = True//The graph is then regenerated again with a new set
	end;

This procedure is used to increase the value of the real number, Maximum number of iterations and the imaginary number by a certain amount by clicking a button. This type of procedure will need to be replicated in the code several times for each of these variables.	procedure IncreaseReal(Sender: TObject)
	begin
	Chart1ColorMapSeries1.Active = False//The graph stops generating the set
	RealNumber = StrToFloat(Edit1.Text) + 0.05// The appropriate variable (In this case the real number) is increased by 0.05
	Edit1.Text := FloatToStr(RealNumber)// The real number in the box where you enter the value is updated
	Chart1ColorMapSeries1.Active = True //The graph is then regenerated
	end;

This procedure is used to calculate all of the new Z-coordinates formed when both the user defined constant and original z-coordinate are entered in to the function	procedure GetZCoordinates(ConstantR, ConstantI) var Count : Integer// What iteration the Z value is calculated on begin begin Z1[Count - 1] = ZRealNumber[Count - 1] Z2[Count - 1]:= ZImaginaryNumber[Count - 1] ZRealNumber[Count] = ((ZRealNumber[Count - 1] * ZRealNumber[Count - 1]) - (ZImaginaryNumber[Count - 1] * ZImaginary[Count - 1])+ RealNumber) ZImaginaryNumber[Count] = ((2 * ZReal[Count - 1] * ZImaginaryNumber[Count - 1]) + ImaginaryNumber) end end

Function / procedure description	Code and notes

This procedure uses the data collected in the previous procedure to actually form the user defined Z-Coordinates path.	procedure ProduceZPath(Sender: TObject);
	var
	RealNumber, ImaginaryNumber : real
	begin
	ZReal[1] = StrToFloat(Edit3.Text) //First Real part of the user defined z coordinate
	ZImaginary[1] = StrToFloat(Edit4.Text) //First Imaginary part of the user defined z coordinate
	RealNumber= StrToFloat(Edit1.Text)// Real Number for constant
	ImaginaryNumber = StrToFloat(Edit2.Text)//Imaginary Number for constant
	GetZPath(ConstantR, ConstantI)
	For Count = 1 to 10 do begin//Used to go through all of the Z points(Count Might be increased in the final version);
	Chart2LineSeries1 .addxy(ZReal[Count],ZImaginary[Count])//Used to make sure the last point in the Z-Path does not link up with the first point
	end
	For Count = 10 to 1 do begin //Used to prevent the final point from linking up with the last.
	Chart2LineSeries1 .addxy(ZReal[Count],ZImaginary[Count]) //Used to plot the newly found Z Coordinates
	end
	end

Function / procedure description	Code and notes

This procedure is used to generate an animation of a changing Julia set by changing the constant by a certain amount at regular intervals.	procedure TEdit1.Timer1Timer(Sender: TObject);
	var
	Rate: real // Used to determine how much the constant increases by each time
	begin
	if ZSquared.Checked = True then //Used to determine which function will be used in the animation(In this case its Z-Squared + C)
	begin
	Rate := Strtofloat(Edit8.Text); // Obtaining Rate from the user
	Chart1ColorMapSeries1.Active := False; //Each series for each function is set to false
	Chart1ColorMapSeries2.Active := False;
	Chart1ColorMapSeries3.Active := False;
	if RealNumber3 < LastReal then
	RealNumber3 := StrToFloat(Edit1.Text) + Rate; //This determines whether to Increase the real part of the constant if it is larger than what the user wants the real number to get to, eg : he might want to get the real number 0.3 to 0.7
	Edit1.Text = FloatToStr(RealNumber3) //Real Number displayed in the text box on screen is updated
	if ImaginaryNumber2 < lastImaginary then
	ImaginaryNumber2 = StrToFloat(Edit2.Text) + Rate //This determines whether to Increase the imaginary part of the constant if it is larger than what the user wants the imaginary number to get to, eg : he might want to get the imaginary number 0.3i to 0.7i
	Edit2.Text = FloatToStr(ImaginaryNumber2)//Imaginary Number displayed in the text box on screen is updated
	if RealNumber3 > LastReal then
	RealNumber3 = StrToFloat(Edit1.Text) - Rate//This determines whether to decrease the real part of the constant if it is larger than what the user wants the real number to get to, eg : he might want to get the real number 0.3 to -0.6
	Edit1.Text = FloatToStr(RealNumber3)//Real Number displayed in the text box on screen is updated
	if ImaginaryNumber2 > lastImaginary then
	ImaginaryNumber2 = StrToFloat(Edit2.Text) - Rate //This determines whether to Increase the imaginary part of the constant if it is larger than what the user wants the imaginary number to get to, eg : he might want to get the imaginary number 0.3i to -0.7i
	Edit2.Text := FloatToStr(ImaginaryNumber2) //Imaginary Number displayed in the text box on screen is updated
	Chart1ColorMapSeries1.Active = True//Julia set is generated for a small amount of time before changing again
	if (RealNumber3 = LastReal) and (ImaginaryNumber2 = LastImaginary) then
	Timer1.Enabled = False //If the imaginary number and real number are equal to the last real number decided by the user the timer is set to false meaning the animation has now stopped;
	end
	if ZCubed.Checked = True then // The next two almost identical sections are exactly the same to the first but use different functions;
	begin
	Rate := Strtofloat(Edit8.Text);
	Chart1ColorMapSeries1.Active = False;

Design

```
    Chart1ColorMapSeries2.Active = False;
    Chart1ColorMapSeries3.Active = False;
    if RealNumber3 < LastReal then
      RealNumber3 = StrToFloat(Edit1.Text) + Rate
    Edit1.Text = FloatToStr(RealNumber3)
    if ImaginaryNumber2 < lastImaginary then
      ImaginaryNumber2 = StrToFloat(Edit2.Text) + Rate
    Edit2.Text = FloatToStr(ImaginaryNumber2)
    if RealNumber3 > LastReal then
      RealNumber3 = StrToFloat(Edit1.Text) - Rate
    Edit1.Text = FloatToStr(RealNumber3)
    if ImaginaryNumber2 > lastImaginary then
      ImaginaryNumber2 = StrToFloat(Edit2.Text) - Rate
    Edit2.Text = FloatToStr(ImaginaryNumber2)
    Chart1ColorMapSeries2.Active = True
    if (RealNumber3 = LastReal) and (ImaginaryNumber2 = lastImaginary) then
      Timer1.Enabled = False
  end;
  if RadioButton3.Checked = True then
  begin
    Rate := Strtofloat(Edit8.Text);
    Chart1ColorMapSeries1.Active = False
    Chart1ColorMapSeries2.Active := False
    Chart1ColorMapSeries3.Active = False
    if RealNumber3 < LastReal then
      RealNumber3 = StrToFloat(Edit1.Text) + Rate
    Edit1.Text := FloatToStr(RealNumber3)
    if ImaginaryNumber2 < lastImaginary then
      ImaginaryNumber2:= StrToFloat(Edit2.Text) + Rate
    Edit2.Text := FloatToStr(ImaginaryNumber2)
    if RealNumber3 > LastReal then
      RealNumber3 = StrToFloat(Edit1.Text) - Rate;
    Edit1.Text = FloatToStr(RealNumber3);
    if ImaginaryNumber2 > lastImaginary then
      ImaginaryNumber2 = StrToFloat(Edit2.Text) - Rate
    Edit2.Text := FloatToStr(ImaginaryNumber2)
    Chart1ColorMapSeries3.Active = True
    if (RealNumber3 = LastReal) and (ImaginaryNumber2 = LastImaginary) then
      Timer1.Enabled = False
  end
end
```

Design data dictionary and data structures

Here is a table of the variables which are absolutely essential for my program to run. My program uses no databases, so all of the data will be stored inside the following variables and arrays when the program is executed.

Variable	Data type	Description	Validation
Imaginary number (Note this value is found by converting the string value entered by the user into a text box, into a real data type)	Real	This will be used to store the imaginary number which is part of the constant (C) in the function $f(z) = z^2 + c$	If any input is given other than a numerical value then set the variable to 0
Real number (Note this value is found by converting the string value entered by the user into a text box, into a real data type)	Real	This will be used to store the Real number which is part of the constant (C) in the function $f(z) = z^2 + c$	If any input is given other than a numerical value then set the variable to 0
Maximum Iterations (Note this value is found by converting the string value entered by the user into a text box, into a integer data type)	Integer	This will be used to store the maximum number of iterations the function can operate on the constant for before it is forced to stop	If any input is given other than a numerical value then set the variable to 100. The Vale must also never become below 0 otherwise the set won't be generated.
ZCoordinate(Note this value is found by converting the string value entered by the user into a text box, into a real data type)	Real	This will be used to temporarily store all of the z values of the function which is used.	For efficiency only values inside the complex plane should be stored temporally stored in this variable.
Iterations	Integer	This is used to keep track of how many times the function using a specific Z-value has been used on the constant (c).	This value must never go over the maximum number of iterations which the user has defined.
ZIterations	Integer	This is used to record the number of times the function is used in the procedure which is used to find the user defined Z coordinates path.	This value must never go beyond the maximum amount of iterations.
ZRealNumber	Array [1..12] of Real	This is the real part of the value of Z defined by the user.	If any input is given other than a numerical value then the program sets the variable to 0.
ZImaginaryNumber	Array [1..12] of Real	This is the Imaginary part of the value of Z defined by the user.	If any input is given other than a numerical value then the program sets the variable to 0.
AnRealNumber	Real	This is the last Real Number the animation ends on.	If any input is given other than a numerical value then the program sets the variable to 0.
AnImaginaryNumber	Real	This is the last imaginary Number the animation ends on.	If any input is given other than a numerical value then the program sets the variable to 0.
Chart1ColorMapSeries1.Active, Chart1ColorMapSeries1.Active Chart1ColorMapSeries1.Active	Boolean	This is used to alter whether the program generates the graph or not. Each of these map series represent a specific function.	No validation required.

Security

The final program does not save any data what so ever so there is no need for security features such as passwords, pins or encryption.

Hardware Specifications

The hardware that will be identified in this section will all be used to allow the program to work and function.

Hardware needed	Input/Output/Storage/Other Device	Use
Keyboard	Input device	A keyboard will be required to type in all of the values necessary for making the program function. Some of these values include the constant, user defined Z-values and the maximum number of iterations the user wants.
Monitor	Output device	Probably the most essential piece of hardware required by the program. A monitor is required to display the user interface and not forgetting the actual Julia set and Z path.
Mouse	Input device	A mouse is required to navigate through the program by clicking on the appropriate text boxes and Buttons when necessary. It is also required to move and zoom into the Julia set so it is able to display it's never ending characteristics.
Hard drive	Storage	A hard drive or any other form of storage device such as an SSD will be required to store the executable file which should not take up much storage (At least 35MB-50mb).
Processor	Other	I would recommend at least a dual core processor due to the sheer amount of coordinates being plotted which should take up a fair amount of the computers CPU to produce them.
Memory	Other	A computer with at least 2GB of memory should be able to run this program as not a lot of memory should be taken up when the program produces the Set.

Test Strategy

When my program has been fully complete and presented to my end user, it is highly necessary that it should be tested so that when a user uses it, all of the program's features work correctly without any errors.

1. Text box validation.
   The program is going to have to allow the user to only be allowed to type in correct values for their desired purpose. This means when they type into for example a text box used for assigning a value to the real number variable, it must only accept real data types and must reject all other sorts, such as strings and characters. The way I will get my end user to record their findings will be in a table like this.

Program Text Box	String/Char Allowed (Yes/No)	Does the program crash?
Real Number text box		
Imaginary Number Text box		
etc		

2. Julia set reliability testing
   As producing Julia sets is the main function of my program, it is absolutely essential that the sets produced are completely accurate. To test this I will ask my user to type in some of their own values for the constant, producing the set and then enter those same constants into an online Julia set generator to see if they gain the same set as my program. Here is how I will record the results.

Constant	Online Generator Url	Is the Julia set the same for both programs?
-0.6 + 0.4i		
etc		

3. Z-path reliability testing
   The path of most Z coordinates are very random and can be hard to tell if they are correct or not. To test the paths, I will ask the end user to come up with 3 Z coordinates and work out for each the first few iterations to see if they match the result on the screen. Hers how I will record the results.

Z-Value	Iterations 1 result	Iterations 1 result	Iterations 1 result
0.3-0.4i			
etc			

4. Button testing
   My program uses multiple buttons which alter many parts of my program like for example, generating the set, changing the constant by a certain amount and changing the number of iterations. To test the reliability of said buttons, I will ask the user to test each button and then record what the result of pressing the button is. I

will then write if this is correct or not. The results will be recorded in a table resembling this.

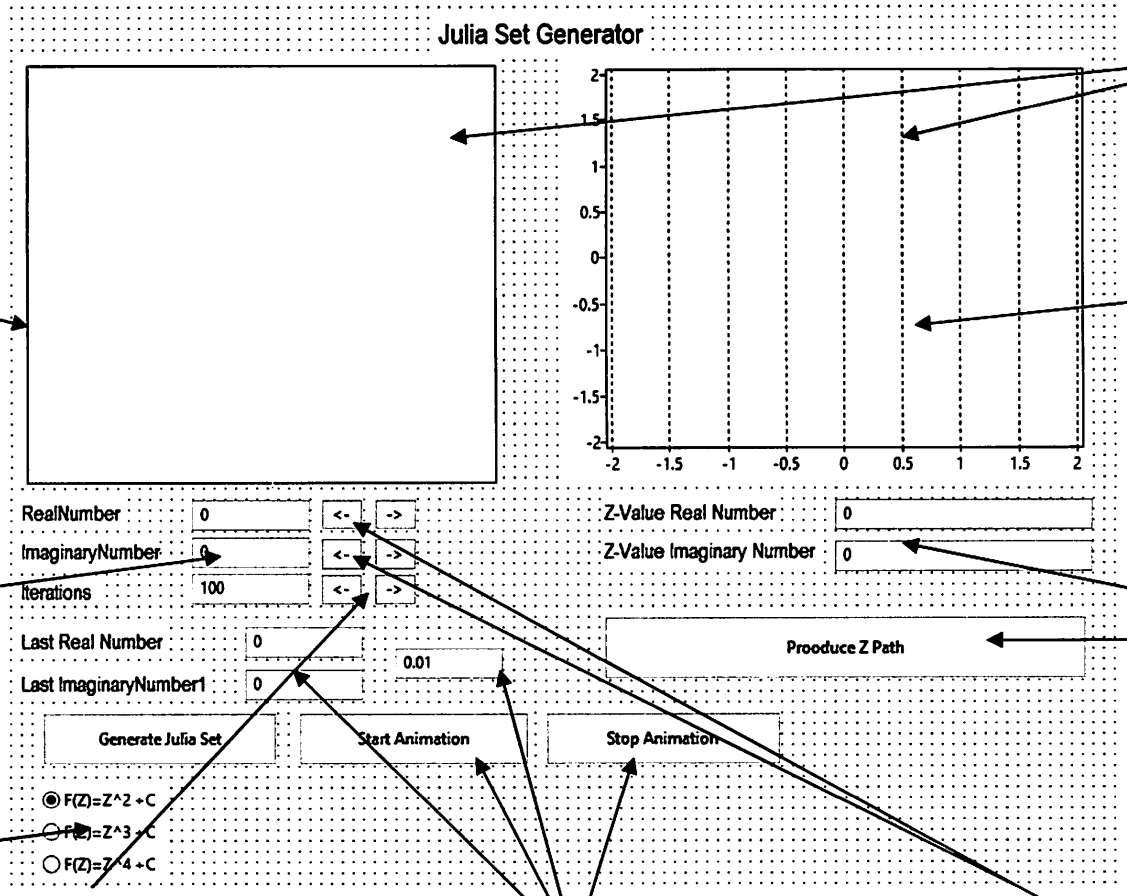
Button	Result	Is this correct?
Increase iterations button		
etc		

5. Animation testing
   In order for the animation part of my program to work, my program requires the user to enter a constant for the animations to end on. This will be the basis of our testing. When the animation has stopped we will enter said user defined constant into an online Julia set generator and check to see if this is the correct set. Here is how I will record this set of results.

Last Constant	Online Generator Url	Is the last Julia set the same for both programs?
-0.6 + 0.4i		
etc		

Human Computer Interface

This section will include a possible interface for my program which will guide the user through the process of getting the program to function.



These two graphs are placed together so the user is able to compare the path of the z coordinate to the Julia set produced.

This graph will be used to display the Julia set generated due to the user defined constant.

This graph will be used to display the path of the user defined Z coordinate.

These boxes are allow the user to change the value of the imaginary number, real number and the number of iterations.

These boxes allow the user to enter the real and imaginary number of the Z coordinate they want to find the path of. If they were to exit these boxes with nothing typed in then they will be set to 0.

These check boxes here are used to determine which function will be used to generate the Julia set. Note, the Z path feature only works for the function $F(Z) = Z^2 + C$. This also determines which function will be used in the animation.

When these buttons are clicked the number of iterations will change by a certain amount. This can be very useful because if the user zooms into the set completely then the user can click the increase button to increase the depth of the Julia set so they may continue zooming in.

These text boxes are used to enter the value of a constant to which the initial value of C will travel to. The box which displays 0.01 is used to determine how much the constant will be changed by, every interval until the initial constant is equal to the final constant. The Buttons 'Start Animation' and 'Stop Animation' are used to start and stop the animation at any point

When these buttons are clicked the values of the real and imaginary numbers are either decreased or increased depending on the user's choice.

Technical solution

The program I have created which is ran by the code I have written below, is used to generate Julia sets as well as other useful features such as generating the path of a user defined Z value which can be used to display how the Julia set is produced as well as features which alter the general colour scheme of the Julia set.

Descriptions of the main and important types of code

Code which generates the Julia set

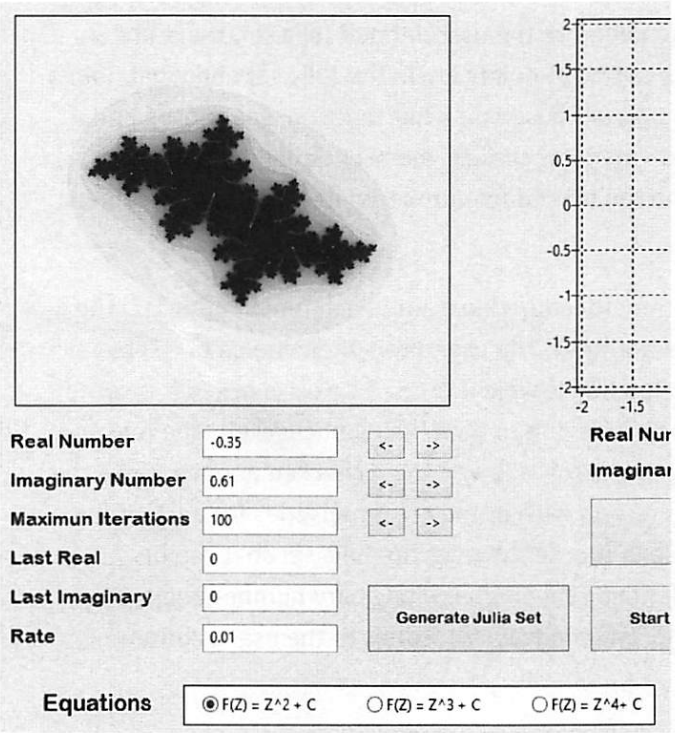
There are 7 procedures in total which are used to generate the user defined Julia set. there are 3 procedures which use a specific function to work out what points are in the Julia set and what points that are not, 3 procedures used to generate the Julia set based on what function was chosen and a procedure which is used to code a button so that when it is clicked, the user defined real number, imaginary number (Which form the constant) and the maximum number of iterations are obtained so the Julia set itself can generated.

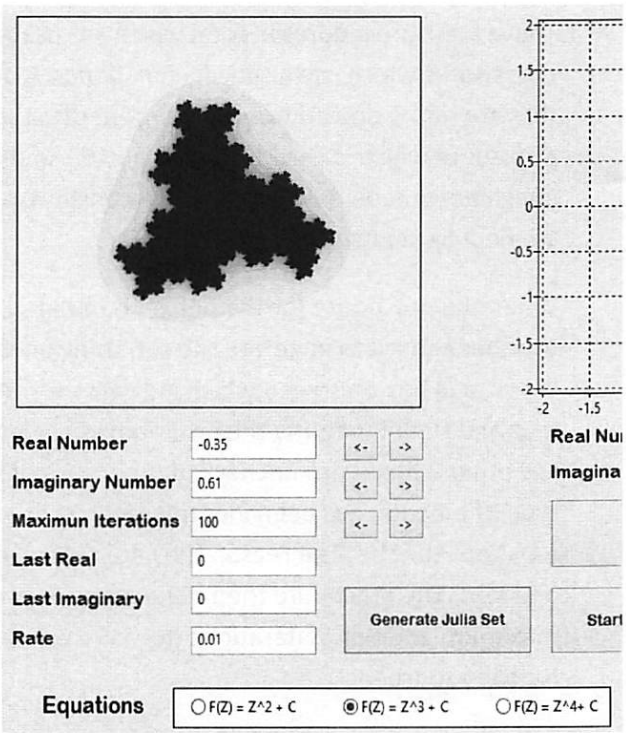
When the procedure for the button is initiated, the procedure looks for which function for F(Z) the user has chosen to generate the set, using an if statement. The user alters their choice of f(Z) by checking a box on screen which indicates which function is which. When a box is checked it is set to true and simultaneously the other boxes are set to false due to them being unchecked. The reason the other 2 boxes are unchecked is because only one box is allowed to be checked at once. Once the type of function has been identified, the procedure sets each of the colormapseries for each of the functions to false. The reason they are set to false is in order to reset the Julia set so it can be updated. The procedure then obtains the user defined real number, imaginary number and the maximum number of iterations from the user and sets the colormapseries of the user's chosen function to true.

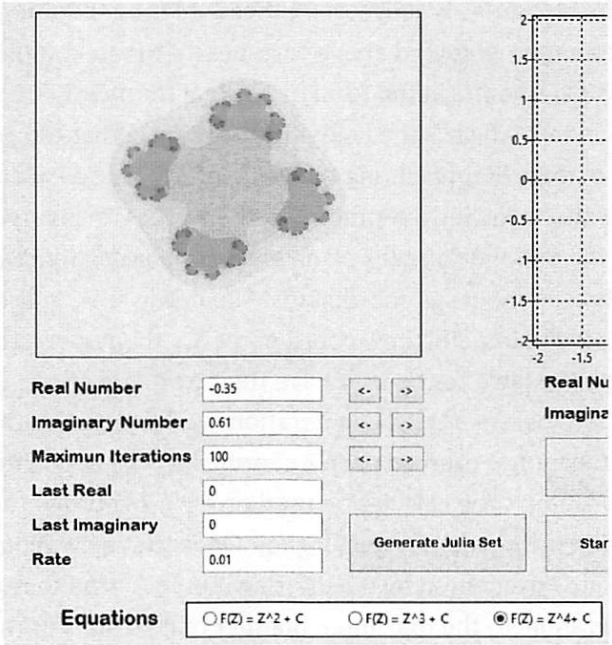
When the specific colormapseries is active, the procedure for generating the Julia set for the chosen function uses the information obtained above to generate the Julia set. The procedure I have used to generate the set has 3 parameters in the form of AX(Real Number Coordinate), AY(Imaginary Number Coordinate) and AZ which is the colour of the point. What the procedure does is go through every single complex number(Representing the value of Z) inside a square of area 4 where the origin is the centre and does the following operations on them. The procedure obtains the first real and imaginary number in the defined complex plane and uses a procedure called DoublePoint which combines the two sets of numbers as coordinates. The newly assigned coordinates are then assigned to a variable called 'coordinates'. The new coordinates are then operated on by the user's chosen function. Each function available to the user have their own procedure, which is used to operate on the Coordinates with two parameters called iterations and Z. What 'iterations' does is record the number of times the function is used on each Z coordinate. The only difference between each procedure for each function is the calculation used on each Z coordinate with the calculation getting more complicated when the power of z gets higher. Once the calculation has been carried out the parameter 'iterations' is incremented by one. A repeat loop is used to repeat the calculation until the number of iterations is equal to the maximum number of iterations or the Z coordinates has escaped towards infinity. The procedure will know if the point escapes towards infinity if it goes outside a

radius of 2. It does this by using the equation of a circle $(x_1 - x_2) + (y_1 - y_2) = r^2$ with x_1 being equal to the Z coordinates' Real Number, y_1 being the Z coordinates' Imaginary Number, x_2 and $y_2$are equal to 0 and the radius squared being equal to 4 due to $2^2 = 4$. Once the function has stopped operating on the Z coordinate, the colormapseries for the function works out if the Initial Z coordinate is inside the Julia set by determining if the number of iterations performed by the function is equal to the maximum number of iterations determined by the end user. If so then the point is assigned the colour black. If not then the number of iterations of the point determines the points colour by dividing it by the maximum number of iterations.

Here are some Screenshots of what the code achieves using the same constant for each function







Code for Julia Set Generation

```
procedure TForm1.GenerateSetButtonClick(Sender: TObject);
//When this button is clicked this procedure is initialised
begin
 if ZSquaredBox.Checked = True then
  //if the user chooses the function F(Z)=z^2 + c then initialise the following statements
 begin
  ZSquaredGenerator.Active := False;
  //Color map series for function F(Z)=z^2 + c is set to false
  ZCubedGenerator.Active := False;
  //Color map series for function F(Z)=z^3 + c is set to false
  Z4Generator.Active := False;
  //Color map series for function F(Z)=z^4 + c is set to false
  RealNumber3 := StrToFloat(RealNumber.Text);
  //The Real Number is obtained from the user
  ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text);
  //TheImaginaryNumber is obtained from the user
  IterationsMax := StrToInt(Iterations.Text);
  //The maximum number of iterations is obtained fromt the user
  ZSquaredGenerator.Active := True;
  //Color map series for function F(Z)=z^2 + c is set to True and the Julia set is generated
 end;
 if ZCubedBox.Checked = True then
  //If the user choose the function F(Z)=z^3 + c then initialise the following statements
 begin
  ZSquaredGenerator.Active := False;
  //Color map series for function F(Z)=z^2 + c is set to false
```

```
  ZCubedGenerator.Active := False;
  //Color map series for function F(Z)=z^3 + c is set to false
  Z4Generator.Active := False;
  //Color map series for function F(Z)=z^4 + c is set to false
  RealNumber3 := StrToFloat(RealNumber.Text);
  //The Real Number is obtained from the user
  ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text);
  //TheImaginaryNumber is obtained from the user
  IterationsMax := StrToInt(Iterations.Text);
  //The maximum number of iterations is obtained fromt the user
  ZCubedGenerator.Active := True;
  //Color map series for function F(Z)=z^3 + c is set to True and the Julia set is generated
end;
if Z4Box.Checked = True then
  //if the user chooses the function F(Z)=z^4 + c then initialise the following statements
begin
  ZSquaredGenerator.Active := False;
  //Color map series for function F(Z)=z^2 + c is set to false
  ZCubedGenerator.Active := False;
  //Color map series for function F(Z)=z^3 + c is set to false
  Z4Generator.Active := False;
  //Color map series for function F(Z)=z^4 + c is set to false
  RealNumber3 := StrToFloat(RealNumber.Text);
  //The Real Number is obtained from the user
  ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text);
  //TheImaginaryNumber is obtained from the user
  IterationsMax := StrToInt(Iterations.Text);
```

```
  //The maximum number of iterations is obtained fromt the user
  Z4Generator.Active := True;
  //Color map series for function F(Z)=z^4 + c is set to True and the Julia set is generated
 end;
end;
```

```
procedure Function1(out Iterations: integer; z: TDoublePoint);
//Used to determine the number of iterations performed by the function F(Z)=z^2 + c
begin
 Iterations := 0; //Iterations is reset to 0
 repeat
  Z := DoublePoint(sqr(z.X) - sqr(z.Y), 2 * z.X * z.Y); //The Z coordinate is Squared
  Z := DoublePoint(RealNumber3, ImaginaryNumber2) + Z;
  // The constant is added to Z^2
  Iterations := Iterations + 1; //Iterations is incremented by 1
 until (Iterations = IterationsMax) or (sqr(z.X) + sqr(z.Y) > Limit);
 // If iterations is equal to the maximum number of iterations or the z point has escaped the radius
of 2 then the procedure stops
 // This method uses the equation the circle to determine if a z point is outside the radius of 2
end;

procedure TForm1.ZSquaredGeneratorCalculate(const AX, AY: double; out AZ: double);
//This procedure goes through every single Z point in the radius of 2, which get operated on by the
following statements
var
```

```
  FunctionIterations: integer;
  Coordinates: TDoublePoint;
begin
  Coordinates := DoublePoint(AX, AY);
  //The Z value's real and imaginary numbers are combined into a set of coordinates
  Function1(FunctionIterations, Coordinates);
  //F(Z)=z^2 + c is used on the ZCoordinates to fined the number of iterations performed
  begin
    if FunctionIterations = IterationsMax then
      //If the number of iterations performed by the function is equal to the maximum number of
iterations then
      AZ := -1 //The Z coordinate is assigned the colour black
    else
      AZ := FunctionIterations / IterationsMax;

    //The Z coordinate is assigned a colour based on the result of this calculation
  end;
end;

procedure Function2(out Iterations: integer; z: TDoublePoint);
//Used to determine the number of iterations performed by the function F(Z)=z^3 + c
begin
  Iterations := 0; //Iterations is reset to 0
  repeat
    Z := Doublepoint((z.X * z.X * z.X) - 3 * (z.Y * z.Y * z.X),
      -(z.Y * z.Y * z.Y) + 3 * (z.X * z.X * z.Y)); //The Z coordinate is Cubed
    Z := DoublePoint(RealNumber3, ImaginaryNumber2) + Z;
    // The constant is added to Z^3
```

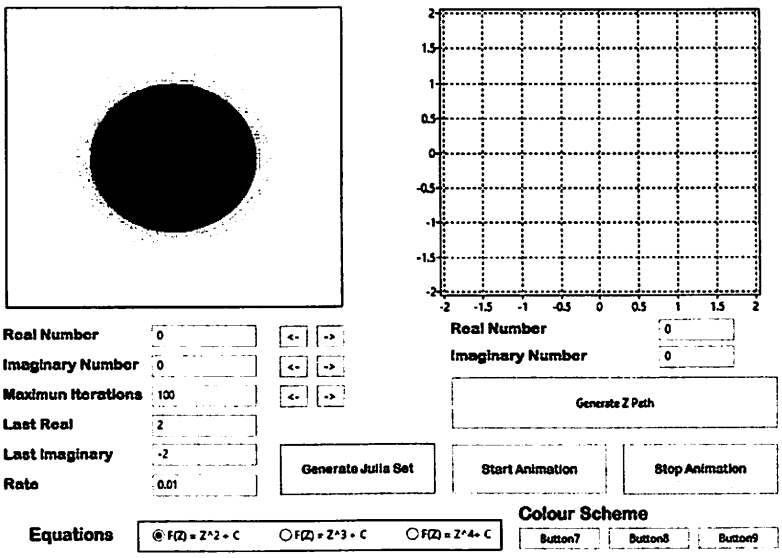
```
  Iterations := Iterations + 1; //Iterations is incremented by 1
 until (Iterations = IterationsMax) or (sqr(z.X) + sqr(z.Y) > Limit);
 // If iterations is equal to the maximum number of iterations or the z point has escaped the radius
of 2 then the procedure stops
 // This method uses the equation the circle to determine if a z point is outside the radius of 2
end;


procedure TForm1.Z4GeneratorCalculate(const AX, AY: double; out AZ: double);
//This procedure goes through every single Z point in the radius of 2, which get operated on by the
following statements
var
 FunctionIterations: integer;
 Coordinates: TDoublePoint;
begin
 Coordinates := DoublePoint(AX, AY);
 //The Z value's real and imaginary numbers are combined into a set of coordinates
 Function2(FunctionIterations, Coordinates);
 //F(Z)=z^3 + c is used on the ZCoordinates to fined the number of iterations performed
 begin
  if FunctionIterations = IterationsMax then
   //If the number of iterations performed by the function is equal to the maximum number of
iterations then
   AZ := -1 //The Z coordinate is assigned the colour black
  else
   AZ := FunctionIterations / IterationsMax;
  //The Z coordinate is assigned a colour based on the result of this calculation
 end;
end;
```

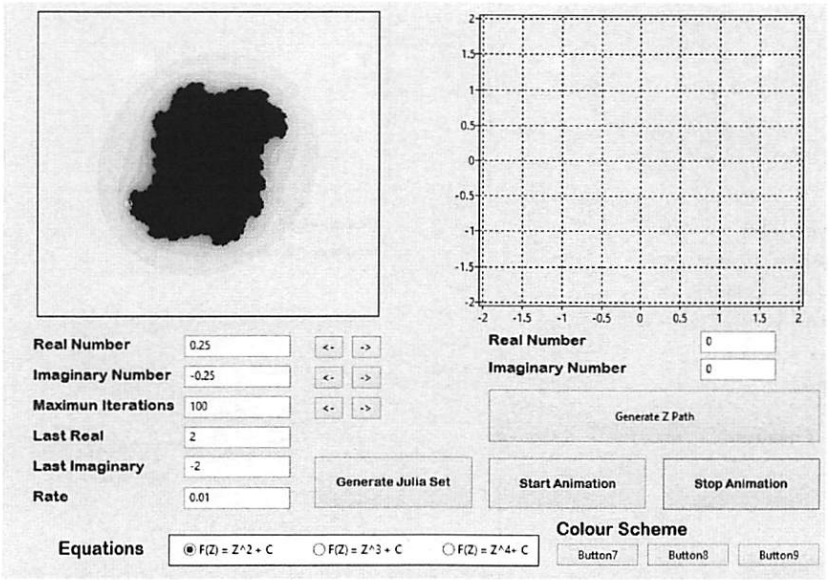
```
procedure Function3(out Iterations: integer; z: TDoublePoint);
//Used to determine the number of iterations performed by the function F(Z)=z^3 + c
begin
 Iterations := 0; //Iterations is reset to 0
 repeat
  Z := Doublepoint((z.X * z.X * z.X * z.X) - 6 * (z.Y * z.Y * z.X * Z.X) +
   (z.Y * z.Y * z.Y * z.Y), -4 * (z.Y * z.Y * z.Y * Z.X) + 4 *
   (z.X * z.X * z.Y * z.X));//The Z coordinate is changed by the power of 4
  Z := DoublePoint(RealNumber3, ImaginaryNumber2) + Z; // The constant is added to Z^3
  Iterations := Iterations + 1;//Iterations is incremented by 1
 until (Iterations = IterationsMax) or (sqr(z.X) + sqr(z.Y) > Limit);
 // This method uses the equation the circle to determine if a z point is outside the radius of 2
 // If iterations is equal to the maximum number of iterations or the z point has escaped the radius
of 2 then the procedure stops
end;


procedure TForm1.ZCubedGeneratorCalculate(const AX, AY: double; out AZ: double);
var
 FunctionIterations: integer;
 Coordinates: TDoublePoint;
begin
 Coordinates := DoublePoint(AX, AY);
 //The Z value's real and imaginary numbers are combined into a set of coordinates
 Function3(FunctionIterations, Coordinates);
 //F(Z)=z^4 + c is used on the ZCoordinates to fined the number of iterations performed
 begin
  if FunctionIterations = IterationsMax then
```

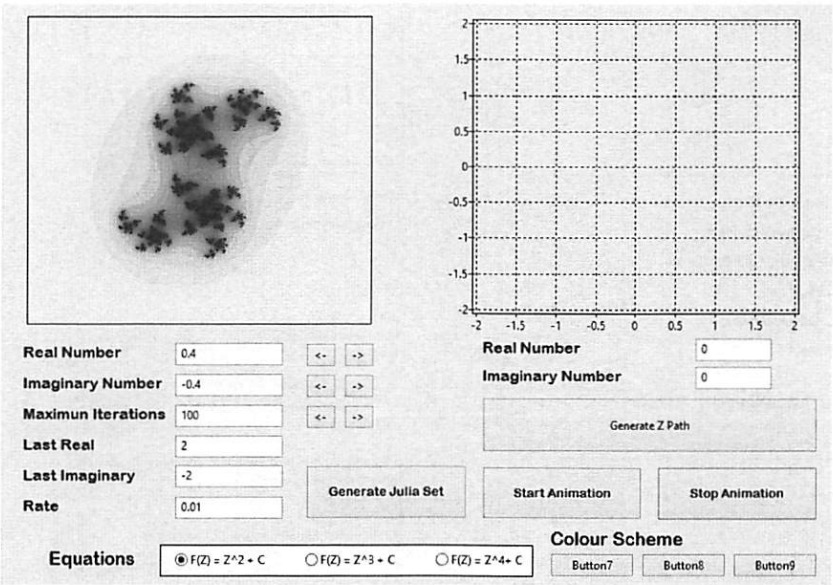
```
    //If the number of iterations performed by the function is equal to the maximum number of
iterations then

    AZ := -1 //The Z coordinate is assigned the colour black

   else

    AZ := FunctionIterations / IterationsMax;

   //The Z coordinate is assigned a colour based on the result of this calculation

  end;

end;
```

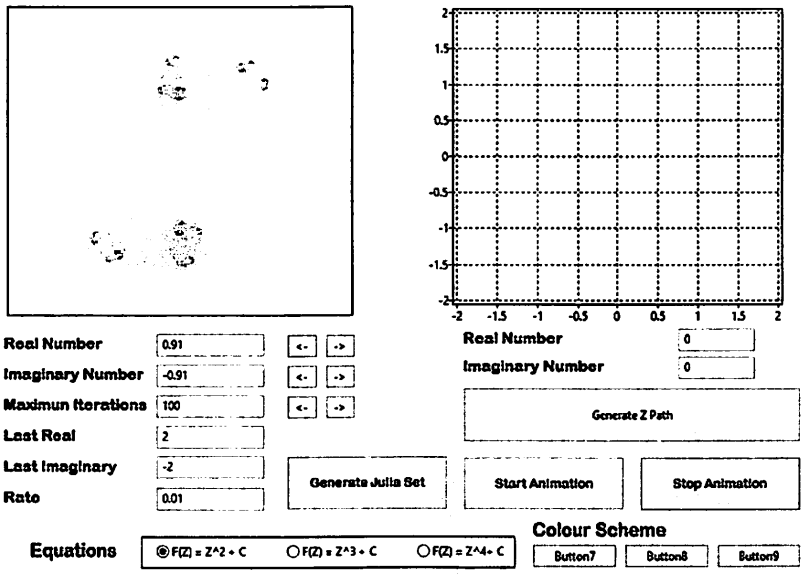
Animation of a changing Julia set

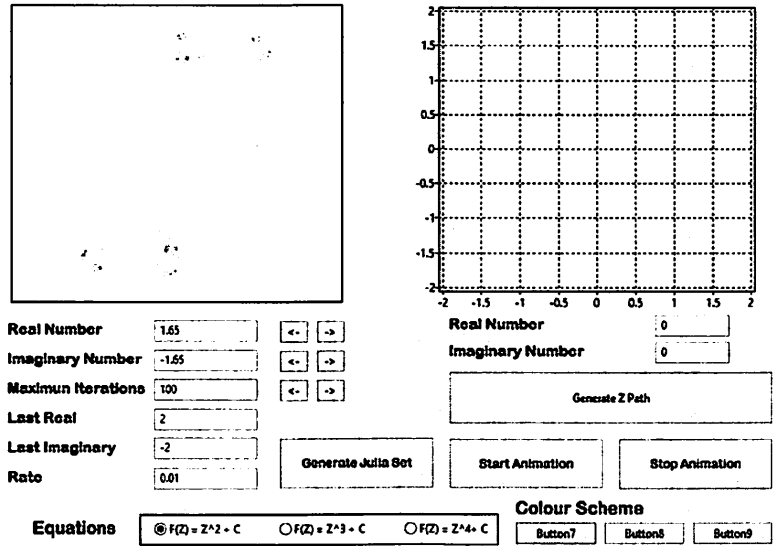
The animation of a Julia set with a changing constant, uses fairly similar code when it comes to generating each Julia set, except the code introduces the usage of a component in Lazarus known as TTimer and also the need for variables which store the last real number and the last imaginary number of the animation. The process of the producing an animation starts of very similarly to generating a static Julia set where the first procedure, when initiated, obtains the constant's real and imaginary number and also the maximum number of iterations. Where it begins to differ is when it also obtains the last real and imaginary numbers the animation will end on. We also set the timer to true so it is now active. The procedure TForm1.Timer1Timer is then repeated until the Timer is set to false when a set of conditions are met. Once initialised the procedure determines which function the user has chosen using 3 if statements (One for each function). Once the function has been determined, the amount the constant is changed by is obtained and is assigned to the variable 'rate', and all the colormapseries of all the functions of z are set to false, meaning they are inactive. If the initial constant's real number is smaller than the last real number, then the real number is increased by the value of the variable, 'rate' and the opposite occurs if the initial real number is larger than the last real number, meaning the initial real number will decrease by the rate value. The same happens with the initial imaginary number, If smaller than the last imaginary number it increases and if larger it decreases. Once increased or decreased the colormapseries is temporarily set to become active and generate the set on screen. The procedure is then repeated until the initial constant's real and imaginary numbers, are the same as the last real and imaginary numbers. Once this condition has been met the timer is set to false meaning it is inactive which causes the animation to stop. Below are some screen shots of the animation in progress.

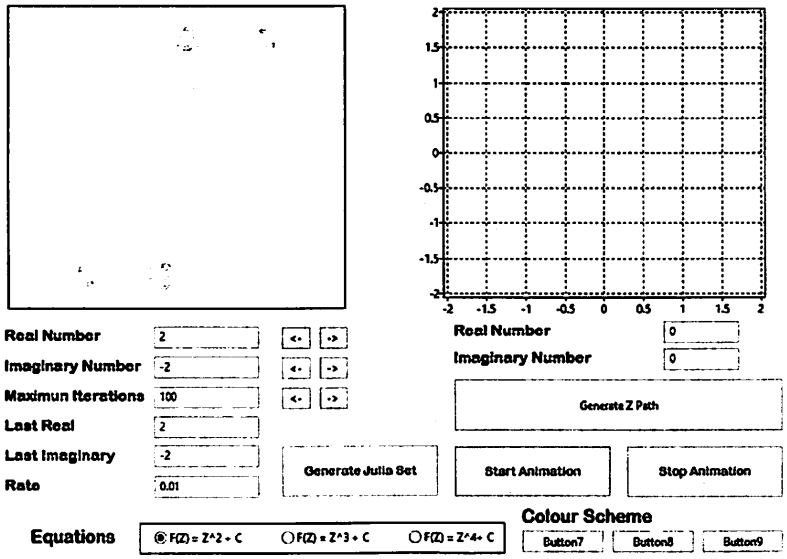



Real Number
0.25
Imaginary Number
-0.25
Maximun Iterations
100
Last Real
2
Last Imaginary
-2
Rate
0.01
<-
->
Generate Julia Set
Real Number
0
Imaginary Number
0
Generate Z Path
Start Animation
Stop Animation
Equations
F(Z) = Z^2 + C
F(Z) = Z^3 + C
F(Z) = Z^4+ C
Colour Scheme
Button7
Button8
Button9


Real Number
0.4
Imaginary Number
-0.4
Maximun Iterations
100
Last Real
2
Last Imaginary
-2
Rate
0.01
<-
->
Generate Julia Set
Real Number
0
Imaginary Number
0
Generate Z Path
Start Animation
Stop Animation
Equations
F(Z) = Z^2 + C
F(Z) = Z^3 + C
F(Z) = Z^4+ C
Colour Scheme
Button7
Button8
Button9


Real Number
0.91
Imaginary Number
-0.91
Maximun Iterations
100
Last Real
2
Last Imaginary
-2
Rate
0.01
Real Number
0
Imaginary Number
0
Generate Z Path
Generate Julia Set
Start Animation
Stop Animation
Colour Scheme
Equations
F(Z) = Z^2 + C
F(Z) = Z^3 + C
F(Z) = Z^4+ C
Button7
Button8
Button9


Real Number
1.65
Imaginary Number
-1.65
Maximun Iterations
100
Last Real
2
Last Imaginary
-2
Rate
0.01
Real Number
0
Imaginary Number
0
Generate Z Path
Generate Julia Set
Start Animation
Stop Animation
Colour Scheme
Equations
F(Z) = Z^2 + C
F(Z) = Z^3 + C
F(Z) = Z^4+ C
Button7
Button8
Button9


Real Number
2
Imaginary Number
-2
Maximun Iterations
100
Last Real
2
Last Imaginary
-2
Rate
0.01
Real Number
0
Imaginary Number
0
Generate Z Path
Generate Julia Set
Start Animation
Stop Animation
Colour Scheme
Equations
F(Z) = Z^2 + C
F(Z) = Z^3 + C
F(Z) = Z^4+ C
Button7
Button8
Button9

As you can see from the first screenshot the initial constant is 0 + 0i and the constant its meant to reach is 2 - 2i. As you go through the screens shots you can see the constant changing gradually towards the last constant until it actually reaches it, causing the animation to stop.

There is also a procedure which stops the animation midway through called TForm1.StopAnimationClick which changes the timer to false causing it to become inactive.

Animation Code

```
procedure TForm1.Timer1Timer(Sender: TObject);

var

 Rate: real;

begin

 if ZSquaredBox.Checked = True then

  //If the user chooses the function F(Z)=z^2 + c then initialise the following statements

 begin

  Rate := Strtofloat(RateBox.Text);

  //The amount the constant is changed by is obtained from the user

  ZSquaredGenerator.Active := False;

  //Color map series for function F(Z)=z^2 + c is set to false

  ZCubedGenerator.Active := False;

  //Color map series for function F(Z)=z^3 + c is set to false

  Z4Generator.Active := False;

  //Color map series for function F(Z)=z^4 + c is set to false

  if RealNumber3 < LastRealr then

   // If the intial real number is smaller than the last real number then

   RealNumber3 := StrToFloat(RealNumber.Text) + Rate;

  //Increase the real number by the value of 'rate'

  RealNumber.Text := FloatToStr(RealNumber3);
```

```
//Update the real number shown in the text box
if ImaginaryNumber2 < lastImaginaryr then
  // If the intial Imaginary number is smaller than the last Imaginary number then
  ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text) + Rate;
// Increase the imaginary number by the value of 'rate'
ImaginaryNumber.Text := FloatToStr(ImaginaryNumber2);
//Update the imaginary nunber in the text box
if RealNumber3 > LastRealr then
  // If the intial real number is larger than the last real number then
  RealNumber3 := StrToFloat(RealNumber.Text) - Rate;
//Decrease the real number by the value of 'rate'
RealNumber.Text := FloatToStr(RealNumber3);
//Update the real number shown in the text box
if ImaginaryNumber2 > lastImaginaryr then
  // If the intial Imaginary number is larger than the last Imaginary number then
  ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text) - Rate;
// decrease the imaginary number by the value of 'rate'
ImaginaryNumber.Text := FloatToStr(ImaginaryNumber2);
//Update the imaginary nunber in the text box
ZSquaredGenerator.Active := True;
//Color map series for function F(Z)=z^2 + c is set to false meaning the julia set is
generated for a time
if (RealNumber3 = LastRealr) and (ImaginaryNumber2 = LastImaginaryr) then
  //if the intial constant is equal to the last constant then
begin
  Timer1.Enabled := False;//the timer is stopped and the animation stops
```

```
 end;
end;
if ZCubedBox.Checked = True then
 //If the user chooses the function F(Z)=z^3 + c then initialise the following statements
begin
 Rate := Strtofloat(RateBox.Text);
 //The amount the constant is changed by is obtained from the user
 ZSquaredGenerator.Active := False;
 //Color map series for function F(Z)=z^2 + c is set to false
 ZCubedGenerator.Active := False;
 //Color map series for function F(Z)=z^3 + c is set to false
 Z4Generator.Active := False;
 //Color map series for function F(Z)=z^4 + c is set to false
 if RealNumber3 < LastRealr then
  RealNumber3 := StrToFloat(RealNumber.Text) + Rate;
 RealNumber.Text := FloatToStr(RealNumber3);
 if ImaginaryNumber2 < lastImaginaryr then
  ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text) + Rate;
 ImaginaryNumber.Text := FloatToStr(ImaginaryNumber2);
 if RealNumber3 > LastRealr then
  RealNumber3 := StrToFloat(RealNumber.Text) - Rate;
 RealNumber.Text := FloatToStr(RealNumber3);
 if ImaginaryNumber2 > lastImaginaryr then
  ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text) - Rate;
 ImaginaryNumber.Text := FloatToStr(ImaginaryNumber2);
 ZCubedGenerator.Active := True;
```

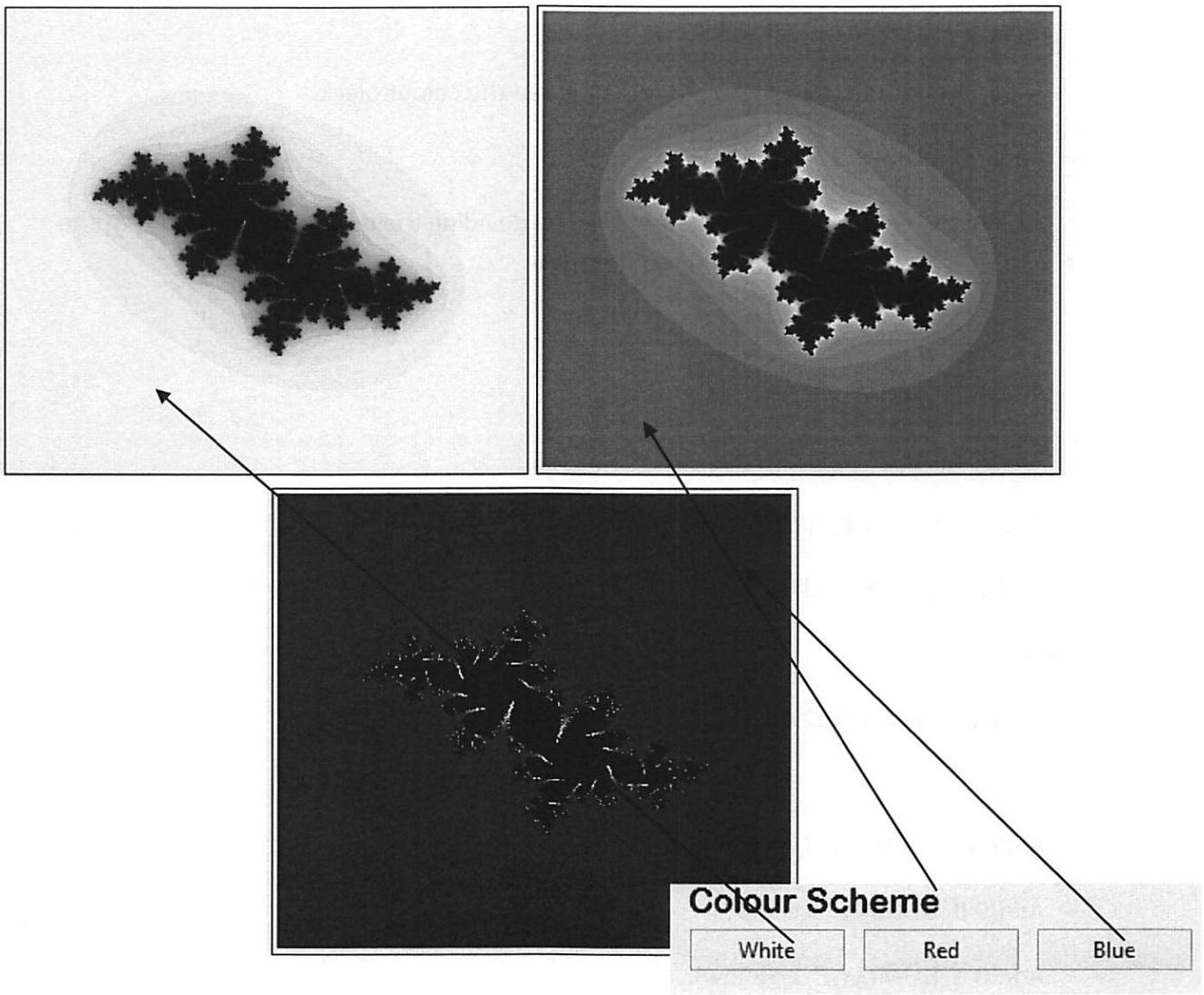
```
  if (RealNumber3 = LastRealr) and (ImaginaryNumber2 = LastImaginaryr) then
  begin
    Timer1.Enabled := False;
  end;
end;
if Z4Box.Checked = True then
  //If the user chooses the function F(Z)=z^4 + c then initialise the following statements
begin
  Rate := Strtofloat(RateBox.Text);
  //The amount the constant is changed by is obtained from the user
  ZSquaredGenerator.Active := False;
  //Color map series for function F(Z)=z^2 + c is set to false
  ZCubedGenerator.Active := False;
  //Color map series for function F(Z)=z^3 + c is set to false
  Z4Generator.Active := False;
  //Color map series for function F(Z)=z^4 + c is set to false
  if RealNumber3 < LastRealr then
    RealNumber3 := StrToFloat(RealNumber.Text) + Rate;
  RealNumber.Text := FloatToStr(RealNumber3);
  if ImaginaryNumber2 < lastImaginaryr then
    ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text) + Rate;
  ImaginaryNumber.Text := FloatToStr(ImaginaryNumber2);
  if RealNumber3 > LastRealr then
    RealNumber3 := StrToFloat(RealNumber.Text) - Rate;
  RealNumber.Text := FloatToStr(RealNumber3);
  if ImaginaryNumber2 > lastImaginaryr then
```

```
    ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text) - Rate;
   ImaginaryNumber.Text := FloatToStr(ImaginaryNumber2);
   Z4Generator.Active := True;
   if (RealNumber3 = LastRealr) and (ImaginaryNumber2 = LastImaginaryr) then
   begin
    Timer1.Enabled := False;
   end;
  end;
end;

procedure TForm1.StartAnimationButtonClick(Sender: TObject);
//This procedure is used when the button needed to start the animaiton is clicked
begin
  Timer1.Enabled := True;//Used to set the timer to true meaning it is now active
  IterationsMax := StrToInt(Iterations.Text);
  //This obtains the maximum number of iterations from the user
  LastRealr := strtofloat(LastReal.Text);//The last real number is obtained
  LastImaginaryr := strtofloat(LastImaginary.Text);
  //The Last imaginary Number is obtained
  RealNumber3 := StrToFloat(RealNumber.Text);//The intial real number is obtained
  ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text);
  //The initial Imaginary number is obtained
end;

procedure TForm1.RealNumberEnter(Sender: TObject);
begin
```

```
  Timer1.Enabled := False;

  //The timer is set to false so it is now inactive if the user tries to change the number
midway through

end;


procedure TForm1.StopAnimationClick(Sender: TObject);

begin

  Timer1.Enabled := False;

  //The timer is set to false so it is now inactive and the animations has stopped

end;


procedure TForm1.ImaginaryNumberEnter(Sender: TObject);

begin

  Timer1.Enabled := False;

  //The timer is set to false so it is now inactive and the aniamtions has stopped

end;
```

Colour Scheme Changing

My program contains multiple colour schemes for which the user can choose from. This part is relatively simple. For each colour source a specific colour is assigned an number, for example in the first colour source the colour white is given the value 0 and blue is given the value 0.1. A point in the Julia set is given its colour based on the number of iterations taken for it to escape the radius of 2 and the maximum number of iterations. In order to determine the colour the number of iterations is divided by the maximum number of iterations which in turn gives the value which represents the specific colour. For example a z value which takes two iterations to escape the radius of 2 divided by the maximum number of iterations which may be 100 would give the result 0.02. The colour of the point is a strong white but with a tint of blue due to it not being exactly zero and instead being slightly nearer to 0.1. There are 3 colour schemes in total with each colour being assigned a specific value. The procedures used to determine the user's chosen colour scheme each program a separate button so that when one of them is clicked the colour source of each of the colour map series is changed to the users chosen colour scheme represented by a new colour source.



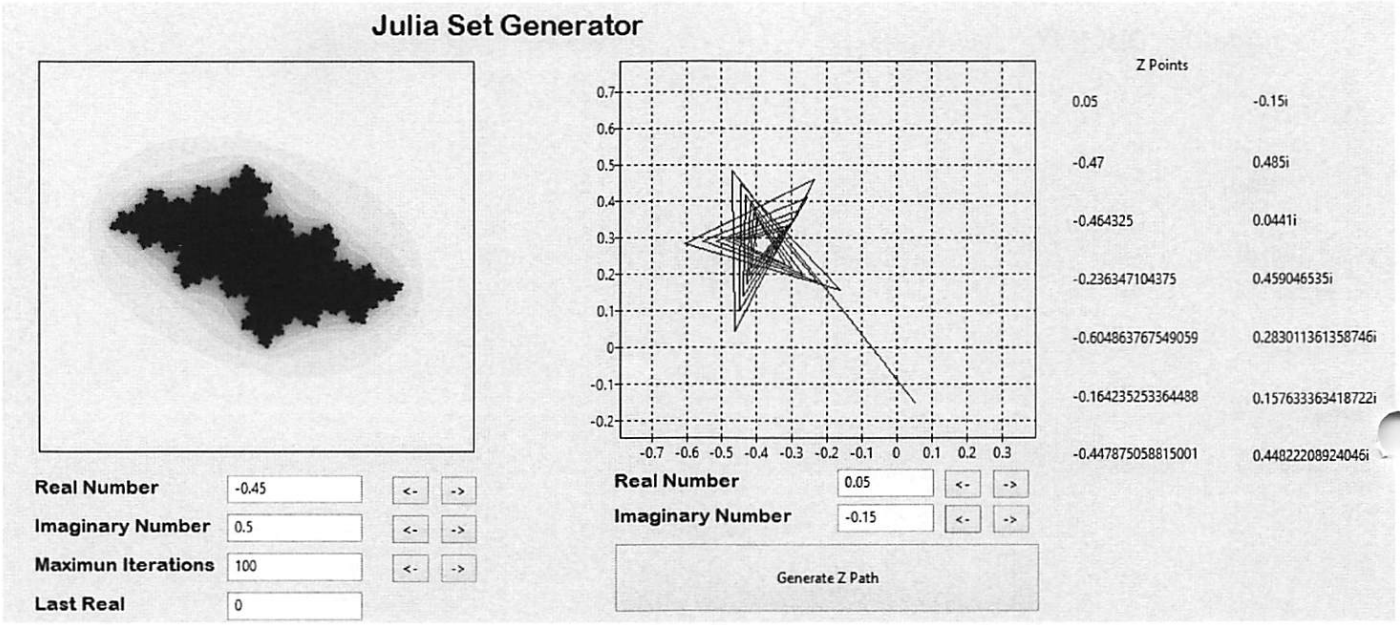
Colour Scheme Code

```
procedure TForm1.PopulateColorSource;
const
  DUMMY = 0.0;
begin
  with ListChartSource1 do // Each colour below is assigned a value
  begin
    Add(-1.0, DUMMY, '', clBlack);
    //If a Z point is in the set the point is assigned the colour black
    Add(0.0, DUMMY, '', clWhite);
    //All of the colour below are determined by deviding the number of iterations of the
function by the maximum number of iterations
    Add(0.2, DUMMY, '', clBlue);
    Add(0.4, DUMMY, '', clPurple);
    Add(0.6, DUMMY, '', clRed);
    Add(0.8, DUMMY, '', clGreen);
    Add(1, DUMMY, '', clYellow);
  end;
  with ListChartSource2 do
  begin
    Add(-1.0, DUMMY, '', clBlack);
    Add(0.0, DUMMY, '', clRed);
    Add(0.1, DUMMY, '', clYellow);
```

```
    Add(0.2, DUMMY, '', clGreen);
    Add(0.3, DUMMY, '', clblue);
    Add(0.4, DUMMY, '', clPurple);

  end;
  with ListChartSource3 do
  begin
    Add(-1.0, DUMMY, '', clBlack);
    Add(0.0, DUMMY, '', clNavy);
    Add(0.2, DUMMY, '', clBlue);
    Add(0.4, DUMMY, '', clAqua);
    Add(0.6, DUMMY, '', clWhite);

  end;
end;
```

Z Path Generation

To generate a Z path, these two following procedures were absolutely necessary in order to produce an accurate path of a Z point. The procedure TForm1.GenerateZPathClick is uninitiated when the button used to generate the Z path is clicked. When clicked the procedure first makes sure the line series which generates the graph is reset, so it no longer displays the previous Z path. The real and imaginary number for the z value are then obtained from the user and are used as the initial z values, by being assigned the first value of the array 'ZReal' and 'ZImaginary'. The constant is also then obtained from the user. The function GetZPath is then initialised using the user defined Real and imaginary number for the constant as parameters. A for loop is then used to store newly calculate Z values to the array until 30 Z points have been stored. Inside the for loop, the function $F(Z) = Z^2 + c$ is applied to the Z value forming a new one which is then stored in the array. Once all 30 points have been found, the procedure ends and the points are then plotted. A for loop is used to plot every point stored in the array using a procedure called addxy. Once all points have been plotted, each Z point is displayed on screen as a complex number by converting each Z real and imaginary number into a string.



As you can see, the Z path spirals in a never ending loop in the shape of a star. This proves that the Z coordinate is inside the Julia set. To the left you can see the values of the first few points.

Z path generator Code

```
procedure GetZPath(ConstantR, ConstantI: real);
//This procedure is used to calculate all the Z points needed to produce the Z path
begin
 for Count := 2 to 30 do //The for loop ends when all 30 Z points have been calculated
 begin
  ZReal[Count] := ((ZReal[Count - 1] * ZReal[Count - 1]) -
   (ZImaginary[Count - 1] * ZImaginary[Count - 1]) + ConstantR);
  //The Z point's real number is operated on by the function and the result is stored in the
array
  ZImaginary[Count] := ((2 * ZReal[Count - 1] * ZImaginary[Count - 1]) + ConstantI);
  //The Z point's imaginary number is operated on by the function and the result is stored
in the array
 end;
end;


procedure TForm1.GenerateZPathClick(Sender: TObject);
// This procedure is used to actually plot the points
var
 ConstantR, ConstantI: real;
 //Real and imaginary parts of the constant defined by the user
begin
 ZPathGeneratorlineseries1.Clear; //The Z path is reset
 ZReal[1] := StrToFloat(ZRealNumber.Text);
 //The Z point's initial Real Number is obtained
 ZImaginary[1] := StrToFloat(ZImaginaryNumber.Text);
```

```
//The Z point's initial Imaginary Number is obtained
ConstantR := StrToFloat(RealNumber.Text); //The constant's real number is obtained
ConstantI := StrToFloat(ImaginaryNumber.Text);
//The constant's Imaginary number is obtained
GetZPath(ConstantR, ConstantI);
//The procedure usef to calculate the Z points for plotting is initiated
for Count := 1 to 30 do
begin
  ZPathGeneratorLineSeries1.addxy(ZReal[Count], ZImaginary[Count]);
  //All 30 of the points are plotted
end;
Label12.Caption := floattostr(ZReal[1]);
//The Real Number of the first few plotted points are converted into a string and are
shown on screen
Label13.Caption := floattostr(ZReal[2]);
Label14.Caption := floattostr(ZReal[3]);
Label15.Caption := floattostr(ZReal[4]);
Label16.Caption := floattostr(ZReal[5]);
Label17.Caption := floattostr(ZReal[6]);
Label18.Caption := floattostr(ZReal[7]);
Label19.Caption := floattostr(ZImaginary[1]) + 'i';
//The Imaginary Number of the first few plotted points are converted into a string and are
shown on screen
Label20.Caption := floattostr(ZImaginary[2]) + 'i';
Label21.Caption := floattostr(ZImaginary[3]) + 'i';
Label22.Caption := floattostr(ZImaginary[4]) + 'i';
Label24.Caption := floattostr(ZImaginary[5]) + 'i';
```

```
  Label25.Caption := floattostr(ZImaginary[6]) + 'i';
  Label26.Caption := floattostr(ZImaginary[7]) + 'i';
end;
```

Full Code with notes

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, TAGraph, TAFuncSeries, TASources, Forms,
  Controls, Graphics, Dialogs, StdCtrls, TAChartUtils, TAGeometry, ExtCtrls,
  Menus, ActnList, MaskEdit, TATypes, TATools, TASeries, TAMultiSeries,
  TARadialSeries, Types, Math, TACustomSeries;

type

  { TForm1 }

  TForm1 = class(TForm)
    DecreaseZImaginary: TButton;
    DecreaseReal: TButton;
    DecreaseImaginary: TButton;
    DecreaseIterations: TButton;
    DecreaseZReal: TButton;
    IncreaseZImaginary: TButton;
    IncreaseIterations: TButton;
    IncreaseButton: TButton;
    IncreaseReal: TButton;
    Button7: TButton;
```

```
Button8: TButton;
Button9: TButton;
IncreaseReal1: TButton;
Label11: TLabel;
Label12: TLabel;
Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
Label16: TLabel;
Label17: TLabel;
Label18: TLabel;
Label19: TLabel;
Label20: TLabel;
Label21: TLabel;
Label22: TLabel;
Label23: TLabel;
Label24: TLabel;
Label25: TLabel;
Label26: TLabel;
ListChartSource2: TListChartSource;
ListChartSource3: TListChartSource;
ZRealNumber: TEdit;
ZImaginaryNumber: TEdit;
GenerateZPath: TButton;
Label10: TLabel;
Label8: TLabel;
Label9: TLabel;
RateBox: TEdit;
StopAnimation: TButton;
```

```
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label7: TLabel;
LastReal: TEdit;
LastImaginary: TEdit;
GenerateSetButton: TButton;
Chart1: TChart;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Shape1: TShape;
Timer1: TTimer;
ZSquaredBox: TRadioButton;
ZCubedBox: TRadioButton;
Z4Box: TRadioButton;
StartAnimationButton: TButton;
ZSquaredGenerator: TColorMapSeries;
ZCubedGenerator: TColorMapSeries;
Z4Generator: TColorMapSeries;
ZPathGenerator: TChart;
ZPathGeneratorLineSeries1: TLineSeries;
RealNumber: TEdit;
ImaginaryNumber: TEdit;
Iterations: TEdit;
ListChartSource1: TListChartSource;
```

```
procedure Button7Click(Sender: TObject);
procedure Button8Click(Sender: TObject);
procedure Button9Click(Sender: TObject);
procedure DecreaseImaginaryClick(Sender: TObject);
procedure DecreaseIterationsClick(Sender: TObject);
procedure DecreaseRealClick(Sender: TObject);
procedure DecreaseZImaginaryClick(Sender: TObject);

procedure DecreaseZRealClick(Sender: TObject);
procedure GenerateSetButtonClick(Sender: TObject);
procedure GenerateZPathClick(Sender: TObject);

procedure ImaginaryNumberEnter(Sender: TObject);
procedure ImaginaryNumberExit(Sender: TObject);
procedure IncreaseButtonClick(Sender: TObject);
procedure IncreaseIterationsClick(Sender: TObject);

procedure IncreaseReal1Click(Sender: TObject);
procedure IncreaseRealClick(Sender: TObject);
procedure IncreaseZImaginaryClick(Sender: TObject);

procedure IterationsExit(Sender: TObject);
procedure LastImaginaryExit(Sender: TObject);
procedure LastRealExit(Sender: TObject);
procedure RateBoxExit(Sender: TObject);

procedure RealNumberEnter(Sender: TObject);
```

```
  procedure RealNumberExit(Sender: TObject);
  procedure StartAnimationButtonClick(Sender: TObject);
  procedure StopAnimationClick(Sender: TObject);
  procedure Timer1Timer(Sender: TObject);

  procedure Z4GeneratorCalculate(const AX, AY: double; out AZ: double);
  procedure ZCubedGeneratorCalculate(const AX, AY: double; out AZ: double);

  procedure ZSquaredGeneratorCalculate(const AX, AY: double; out AZ: double);
  procedure FormCreate(Sender: TObject);
  procedure PopulateColorSource;

 private
  { private declarations }
 public
  { public declarations }
 end;

var
 Form1: TForm1;
 ImaginaryNumber2: real;
 RealNumber3: real;
 IterationsMax: integer;
 LastImaginaryR: real;
 LastRealR: real;
 ZReal: array [1..30] of real;
 ZImaginary: array[1..30] of real;
 Count: integer;
 RealNumber : Tform1;
```

```
  ImaginaryNumber : TForm1;
  ZSquaredGenerator : TForm1;
  ZCubedGenerator : TForm1;
  Z4Generator : Tform1;
const
  Radius = 2;
  Limit = Sqr(Radius); // X^2 + Y^2 = R^2

implementation

{$R *.lfm}

{ TForm1 }

procedure TForm1.RealNumberExit(Sender: TObject);
//once you stop typing in the Constants real number box the following statements occur
var
  a: integer;
  Length1: integer;
begin
  if RealNumber.Text = '' then //If Nothing is typed in the real number is set to 0
  begin
    RealNumber.Text := '0';
  end;
  A := Ord(leftstr(RealNumber.Text, 1)[1]);
  //if the first char is a letter then the real number is set to Zero
  //Note this doesm't completely mean strings can be entered eg if a number then a letter is entered and error
message will occur
  //but as long as the user clicks yes they can change the real number to a numerical value with out the
program crashing
```

```
  Length1 := Length(RealNumber.Text);
  if ((A) > 59) or ((A) < 45) or (length1 > 5) or
    //if the string length is larger than 5 then the real number is reset to 0
    (leftstr(RealNumber.Text, 1)[1] = '.') then
  begin
    RealNumber.Text := '0';
  end;
end;


procedure TForm1.ImaginaryNumberExit(Sender: TObject);
//once you stop typing in the Constants imaginary number box the following statements occur
var
  a: integer;
  Length1: integer;
begin
  if ImaginaryNumber.Text = '' then
  begin
    ImaginaryNumber.Text := '0';
  end;
  A := Ord(leftstr(ImaginaryNumber.Text, 1)[1]);
  //if the first char is a letter then the Imaginary number is set to Zero
  //Note this doesm't completely mean strings can be entered eg if a number then a letter is entered and error
message will occur
  //but as long as the user clicks yes they can change the Imaginary number to a numerical value without the
program crashing
  Length1 := Length(ImaginaryNumber.Text);
  if ((A) > 59) or ((A) < 45) or
    (length1 > 5) or//if the string length is larger than 5 then the imaginary number is reset to 0
```

```
  (leftstr(ImaginaryNumber.Text, 1)[1] = '.') then
 begin
  ImaginaryNumber.Text := '0';
 end;
end;

procedure TForm1.IncreaseIterationsClick(Sender: TObject);
//This is used to increase the maximum number of iterations by 50
begin
 ZSquaredGenerator.Active := False;
 ZCubedGenerator.Active := False;
 Z4Generator.Active := False;
 IterationsMax := StrToInt(Iterations.Text) + 50;
 if IterationsMax <= 0 then
  // if the maximum number of iterations falls below zero the maximum number of iterations is changed to
100
 begin
  Iterations.Text := '100';
  IterationsMax := 100;
 end;
 Iterations.Text := IntToStr(IterationsMax);
 if ZSquaredBox.Checked = True then
  ZSquaredGenerator.Active := True;
 if ZCubedBox.Checked = True then
  ZCubedGenerator.Active := True;
 if Z4Box.Checked = True then
  Z4Generator.Active := True;
end;
```

```
procedure TForm1.DecreaseIterationsClick(Sender: TObject);
//This is used to decrease the maximum number of iterations by 50
begin
 ZSquaredGenerator.Active := False;
 ZCubedGenerator.Active := False;
 Z4Generator.Active := False;
 IterationsMax := StrToInt(Iterations.Text) - 50;
 // if the maximum number of iterations falls below or is equal to zero the maximum number of iterations is
changed to 100
 if IterationsMax <= 0 then
 begin
  Iterations.Text := '100';
  IterationsMax := 100;
 end;
 Iterations.Text := IntToStr(IterationsMax);
 if ZSquaredBox.Checked = True then
  ZSquaredGenerator.Active := True;
 if ZCubedBox.Checked = True then
  ZCubedGenerator.Active := True;
 if Z4Box.Checked = True then
  Z4Generator.Active := True;
end;

procedure TForm1.LastImaginaryExit(Sender: TObject);
var
 a: integer;
 Length1: integer;
begin
 if lastImaginary.Text = '' then
```

```
  begin
    lastImaginary.Text := '0';
  end;
  A := Ord(leftstr(lastImaginary.Text, 1)[1]);

  Length1 := Length(LastImaginary.Text);
  if ((A) > 59) or ((A) < 45) or (length1 > 5) or
    (leftstr(LastImaginary.Text, 1)[1] = '.') then
  begin
    LastImaginary.Text := '0';
  end;
end;

procedure TForm1.LastRealExit(Sender: TObject);
var
  a: integer;
  Length1: integer;
begin
  if lastReal.Text = '' then
  begin
    lastReal.Text := '0';
  end;
  A := Ord(leftstr(lastReal.Text, 1)[1]);

  Length1 := Length(lastReal.Text);
  if ((A) > 59) or ((A) < 45) or (length1 > 5) or
    (leftstr(lastReal.Text, 1)[1] = '.') then
  begin
    lastReal.Text := '0';
```

```
  end;
end;

procedure TForm1.RateBoxExit(Sender: TObject);
var
  a: integer;
  Length1: integer;
begin
  if RateBox.Text = '' then
  begin
    RateBox.Text := '0';
  end;
  A := Ord(leftstr(RateBox.Text, 1)[1]);

  Length1 := Length(RateBox.Text);
  if ((A) > 59) or ((A) < 45) or (length1 > 5) or
    (leftstr(RateBox.Text, 1)[1] = '.') then
  begin
    RateBox.Text := '0';
  end;
end;

procedure TForm1.IterationsExit(Sender: TObject);
var
  a: integer;
  Length1: integer;
begin
  if Iterations.Text = '' then
  begin
```

```
    Iterations.Text := '100';
  end;
  A := Ord(leftstr(Iterations.Text, 1)[1]);


  Length1 := Length(Iterations.Text);
  if ((A) > 59) or ((A) < 45) or (length1 > 8) or
    (leftstr(Iterations.Text, 1)[1] = '.') or (leftstr(Iterations.Text, 1)[1] = '-') then
  begin
    Iterations.Text := '100';
    if StrtoFloat(Iterations.Text) < 0 then
      Iterations.Text := '100';
  end;
end;




procedure TForm1.GenerateSetButtonClick(Sender: TObject);
//When this button is clicked this procedure is initialised
begin
  if ZSquaredBox.Checked = True then
    //if the user chooses the function F(Z)=z^2 + c then initialise the following statements
  begin
    ZSquaredGenerator.Active := False;
    //Color map series for function F(Z)=z^2 + c is set to false
    ZCubedGenerator.Active := False;
    //Color map series for function F(Z)=z^3 + c is set to false
    Z4Generator.Active := False;
    //Color map series for function F(Z)=z^4 + c is set to false
```

```
  RealNumber3 := StrToFloat(RealNumber.Text);
  //The Real Number is obtained from the user
  ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text);
  //TheImaginaryNumber is obtained from the user
  IterationsMax := StrToInt(Iterations.Text);
  //The maximum number of iterations is obtained fromt the user
  ZSquaredGenerator.Active := True;
  //Color map series for function F(Z)=z^2 + c is set to True and the Julia set is generated
 end;
 if ZCubedBox.Checked = True then
  //If the user choose the function F(Z)=z^3 + c then initialise the following statements
 begin
  ZSquaredGenerator.Active := False;
  //Color map series for function F(Z)=z^2 + c is set to false
  ZCubedGenerator.Active := False;
  //Color map series for function F(Z)=z^3 + c is set to false
  Z4Generator.Active := False;
  //Color map series for function F(Z)=z^4 + c is set to false
  RealNumber3 := StrToFloat(RealNumber.Text);
  //The Real Number is obtained from the user
  ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text);
  //TheImaginaryNumber is obtained from the user
  IterationsMax := StrToInt(Iterations.Text);
  //The maximum number of iterations is obtained fromt the user
  ZCubedGenerator.Active := True;
  //Color map series for function F(Z)=z^3 + c is set to True and the Julia set is generated
 end;
 if Z4Box.Checked = True then
  //if the user chooses the function F(Z)=z^4 + c then initialise the following statements
```

```
 begin
  ZSquaredGenerator.Active := False;
  //Color map series for function F(Z)=z^2 + c is set to false
  ZCubedGenerator.Active := False;
  //Color map series for function F(Z)=z^3 + c is set to false
  Z4Generator.Active := False;
  //Color map series for function F(Z)=z^4 + c is set to false
  RealNumber3 := StrToFloat(RealNumber.Text);
  //The Real Number is obtained from the user
  ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text);
  //TheImaginaryNumber is obtained from the user
  IterationsMax := StrToInt(Iterations.Text);
  //The maximum number of iterations is obtained fromt the user
  Z4Generator.Active := True;
  //Color map series for function F(Z)=z^4 + c is set to True and the Julia set is generated
 end;
end;


procedure Function1(out Iterations: integer; z: TDoublePoint);
//Used to determine the number of iterations performed by the function F(Z)=z^2 + c
begin
 Iterations := 0; //Iterations is reset to 0
 repeat
  Z := DoublePoint(sqr(z.X) - sqr(z.Y), 2 * z.X * z.Y); //The Z coordinate is Squared
  Z := DoublePoint(RealNumber3, ImaginaryNumber2) + Z;
  // The constant is added to Z^2
```

```
  Iterations := Iterations + 1; //Iterations is incremented by 1
 until (Iterations = IterationsMax) or (sqr(z.X) + sqr(z.Y) > Limit);
 // If iterations is equal to the maximum number of iterations or the z point has escaped the radius of 2 then
the procedure stops
 // This method uses the equation the circle to determine if a z point is outside the radius of 2
end;


procedure TForm1.ZSquaredGeneratorCalculate(const AX, AY: double; out AZ: double);
//This procedure goes through every single Z point in the radius of 2, which get operated on by the following
statements
var
 FunctionIterations: integer;
 Coordinates: TDoublePoint;
begin
 Coordinates := DoublePoint(AX, AY);
 //The Z value's real and imaginary numbers are combined into a set of coordinates
 Function1(FunctionIterations, Coordinates);
 //F(Z)=z^2 + c is used on the ZCoordinates to fined the number of iterations performed
 begin
  if FunctionIterations = IterationsMax then
   //If the number of iterations performed by the function is equal to the maximum number of iterations then
   AZ := -1 //The Z coordinate is assigned the colour black
  else
   AZ := FunctionIterations / IterationsMax;


  //The Z coordinate is assigned a colour based on the result of this calculation
 end;
end;


procedure Function2(out Iterations: integer; z: TDoublePoint);
```

```
//Used to determine the number of iterations performed by the function F(Z)=z^3 + c
begin
 Iterations := 0; //Iterations is reset to 0
 repeat
  Z := Doublepoint((z.X * z.X * z.X) - 3 * (z.Y * z.Y * z.X),
   -(z.Y * z.Y * z.Y) + 3 * (z.X * z.X * z.Y)); //The Z coordinate is Cubed
  Z := DoublePoint(RealNumber3, ImaginaryNumber2) + Z;
  // The constant is added to Z^3
  Iterations := Iterations + 1; //Iterations is incremented by 1
 until (Iterations = IterationsMax) or (sqr(z.X) + sqr(z.Y) > Limit);
 // If iterations is equal to the maximum number of iterations or the z point has escaped the radius of 2 then
the procedure stops
 // This method uses the equation the circle to determine if a z point is outside the radius of 2
end;


procedure TForm1.Z4GeneratorCalculate(const AX, AY: double; out AZ: double);
//This procedure goes through every single Z point in the radius of 2, which get operated on by the following
statements
var
 FunctionIterations: integer;
 Coordinates: TDoublePoint;
begin
 Coordinates := DoublePoint(AX, AY);
 //The Z value's real and imaginary numbers are combined into a set of coordinates
 Function2(FunctionIterations, Coordinates);
 //F(Z)=z^3 + c is used on the ZCoordinates to fined the number of iterations performed
 begin
  if FunctionIterations = IterationsMax then
   //If the number of iterations performed by the function is equal to the maximum number of iterations then
   AZ := -1 //The Z coordinate is assigned the colour black
```

```
  else
    AZ := FunctionIterations / IterationsMax;
  //The Z coordinate is assigned a colour based on the result of this calculation
 end;
end;

procedure Function3(out Iterations: integer; z: TDoublePoint);
//Used to determine the number of iterations performed by the function F(Z)=z^3 + c
begin
 Iterations := 0; //Iterations is reset to 0
 repeat
  Z := Doublepoint((z.X * z.X * z.X * z.X) - 6 * (z.Y * z.Y * z.X * Z.X) +
    (z.Y * z.Y * z.Y * z.Y), -4 * (z.Y * z.Y * z.Y * Z.X) + 4 *
    (z.X * z.X * z.Y * z.X));//The Z coordinate is changed by the power of 4
  Z := DoublePoint(RealNumber3, ImaginaryNumber2) + Z; // The constant is added to Z^3
  Iterations := Iterations + 1;//Iterations is incremented by 1
 until (Iterations = IterationsMax) or (sqr(z.X) + sqr(z.Y) > Limit);
 // This method uses the equation the circle to determine if a z point is outside the radius of 2
 // If iterations is equal to the maximum number of iterations or the z point has escaped the radius of 2 then
the procedure stops
end;

procedure TForm1.ZCubedGeneratorCalculate(const AX, AY: double; out AZ: double);
var
 FunctionIterations: integer;
 Coordinates: TDoublePoint;
begin
 Coordinates := DoublePoint(AX, AY);
 //The Z value's real and imaginary numbers are combined into a set of coordinates
```

```
  Function3(FunctionIterations, Coordinates);
  //F(Z)=z^4 + c is used on the ZCoordinates to fined the number of iterations performed
  begin
   if FunctionIterations = IterationsMax then
    //If the number of iterations performed by the function is equal to the maximum number of iterations then
    AZ := -1 //The Z coordinate is assigned the colour black
   else
    AZ := FunctionIterations / IterationsMax;
   //The Z coordinate is assigned a colour based on the result of this calculation
  end;
end;



procedure TForm1.DecreaseImaginaryClick(Sender: TObject);
//This procedure is used to decrease the imaginary number by 0.05
begin
  ZSquaredGenerator.Active := False;
  //Color map series for function F(Z)=z^2 + c is set to false
  ZCubedGenerator.Active := False;
  //Color map series for function F(Z)=z^3 + c is set to false
  Z4Generator.Active := False;
  //Color map series for function F(Z)=z^2 + c is set to false
  ImaginaryNumber2 := StrTofloat(ImaginaryNumber.Text) - 0.05;
  //The imaginary number is decreased
  ImaginaryNumber.Text := floatToStr(ImaginaryNumber2);
  //The text box for the imaginary number is updated
  IterationsMax := StrToInt(Iterations.Text);
  if ZSquaredBox.Checked = True then
```

```
  ZSquaredGenerator.Active := True;
 //Color map series for function F(Z)=z^2 + c is set to true
 if ZCubedBox.Checked = True then
  ZCubedGenerator.Active := True;
 //Color map series for function F(Z)=z^3 + c is set to true
 if Z4Box.Checked = True then
  Z4Generator.Active := True;
 //Color map series for function F(Z)=z^4 + c is set to true
end;


procedure TForm1.IncreaseButtonClick(Sender: TObject);
begin
 ZSquaredGenerator.Active := False;
 //Color map series for function F(Z)=z^2 + c is set to false
 ZCubedGenerator.Active := False;
 //Color map series for function F(Z)=z^3 + c is set to false
 Z4Generator.Active := False;
 //Color map series for function F(Z)=z^2 + c is set to false
 ImaginaryNumber2 := StrTofloat(ImaginaryNumber.Text) + 0.05;
 //The imaginary number is increased
 ImaginaryNumber.Text := floatToStr(ImaginaryNumber2);
 //The text box for the imaginary number is updated
 IterationsMax := StrToInt(Iterations.Text);
 if ZSquaredBox.Checked = True then
  ZSquaredGenerator.Active := True;
 //Color map series for function F(Z)=z^2 + c is set to true
 if ZCubedBox.Checked = True then
```

```
  ZCubedGenerator.Active := True;
 //Color map series for function F(Z)=z^3 + c is set to true
 if Z4Box.Checked = True then
  Z4Generator.Active := True;
 //Color map series for function F(Z)=z^4 + c is set to true
end;

procedure TForm1.IncreaseRealClick(Sender: TObject);
begin
 ZSquaredGenerator.Active := False;
 //Color map series for function F(Z)=z^2 + c is set to false
 ZCubedGenerator.Active := False;
 //Color map series for function F(Z)=z^3 + c is set to false
 Z4Generator.Active := False;
 //Color map series for function F(Z)=z^4 + c is set to false
 RealNumber3 := StrTofloat(RealNumber.Text) + 0.05;//The Real number is increased
 RealNumber.Text := floatToStr(RealNumber3);
 //The text box for the Real number is updated
 IterationsMax := StrToInt(Iterations.Text);
 if ZSquaredBox.Checked = True then
  ZSquaredGenerator.Active := True;
 //Color map series for function F(Z)=z^2 + c is set to true
 if ZCubedBox.Checked = True then
  ZCubedGenerator.Active := True;
 //Color map series for function F(Z)=z^3 + c is set to true
 if Z4Box.Checked = True then
  Z4Generator.Active := True;
 //Color map series for function F(Z)=z^4 + c is set to true
end;
```

```
procedure TForm1.DecreaseRealClick(Sender: TObject);
begin
 ZSquaredGenerator.Active := False;
 //Color map series for function F(Z)=z^2 + c is set to false
 ZCubedGenerator.Active := False;
 //Color map series for function F(Z)=z^3 + c is set to false
 Z4Generator.Active := False;
 //Color map series for function F(Z)=z^4 + c is set to false
 RealNumber3 := StrTofloat(RealNumber.Text) - 0.05;//The Real number is decreased
 RealNumber.Text := floatToStr(RealNumber3);
 //The text box for the Real number is updated
 IterationsMax := StrToInt(Iterations.Text);
 if ZSquaredBox.Checked = True then
  ZSquaredGenerator.Active := True;
 //Color map series for function F(Z)=z^2 + c is set to true
 if ZCubedBox.Checked = True then
  ZCubedGenerator.Active := True;
 //Color map series for function F(Z)=z^3 + c is set to true
 if Z4Box.Checked = True then
  Z4Generator.Active := True;
 //Color map series for function F(Z)=z^4 + c is set to true
end;


procedure TForm1.Timer1Timer(Sender: TObject);
var
 Rate: real;
```

```
begin
 if ZSquaredBox.Checked = True then
  //If the user chooses the function F(Z)=z^2 + c then initialise the following statements
 begin
  Rate := Strtofloat(RateBox.Text);
  //The amount the constant is changed by is obtained from the user
  ZSquaredGenerator.Active := False;
  //Color map series for function F(Z)=z^2 + c is set to false
  ZCubedGenerator.Active := False;
  //Color map series for function F(Z)=z^3 + c is set to false
  Z4Generator.Active := False;
  //Color map series for function F(Z)=z^4 + c is set to false
  if RealNumber3 < LastRealr then
   // If the intial real number is smaller than the last real number then
   RealNumber3 := StrToFloat(RealNumber.Text) + Rate;
  //Increase the real number by the value of 'rate'
  RealNumber.Text := FloatToStr(RealNumber3);
  //Update the real number shown in the text box
  if ImaginaryNumber2 < lastImaginaryr then
   // If the intial Imaginary number is smaller than the last Imaginary number then
   ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text) + Rate;
  // Increase the imaginary number by the value of 'rate'
  ImaginaryNumber.Text := FloatToStr(ImaginaryNumber2);
  //Update the imaginary nunber in the text box
  if RealNumber3 > LastRealr then
   // If the intial real number is larger than the last real number then
   RealNumber3 := StrToFloat(RealNumber.Text) - Rate;
  //Decrease the real number by the value of 'rate'
  RealNumber.Text := FloatToStr(RealNumber3);
```

```
  //Update the real number shown in the text box
  if ImaginaryNumber2 > lastImaginaryr then
    // If the intial Imaginary number is larger than the last Imaginary number then
    ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text) - Rate;
  // decrease the imaginary number by the value of 'rate'
  ImaginaryNumber.Text := FloatToStr(ImaginaryNumber2);
  //Update the imaginary nunber in the text box
  ZSquaredGenerator.Active := True;
  //Color map series for function F(Z)=z^2 + c is set to false meaning the julia set is generated for a time
  if (RealNumber3 = LastRealr) and (ImaginaryNumber2 = LastImaginaryr) then
    //if the intial constant is equal to the last constant then
  begin
    Timer1.Enabled := False;//the timer is stopped and the animation stops
  end;
end;
if ZCubedBox.Checked = True then
  //If the user chooses the function F(Z)=z^3 + c then initialise the following statements
begin
  Rate := Strtofloat(RateBox.Text);
  //The amount the constant is changed by is obtained from the user
  ZSquaredGenerator.Active := False;
  //Color map series for function F(Z)=z^2 + c is set to false
  ZCubedGenerator.Active := False;
  //Color map series for function F(Z)=z^3 + c is set to false
  Z4Generator.Active := False;
  //Color map series for function F(Z)=z^4 + c is set to false
  if RealNumber3 < LastRealr then
    RealNumber3 := StrToFloat(RealNumber.Text) + Rate;
  RealNumber.Text := FloatToStr(RealNumber3);
```

```
if ImaginaryNumber2 < lastImaginaryr then
  ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text) + Rate;
ImaginaryNumber.Text := FloatToStr(ImaginaryNumber2);
if RealNumber3 > LastRealr then
  RealNumber3 := StrToFloat(RealNumber.Text) - Rate;
RealNumber.Text := FloatToStr(RealNumber3);
if ImaginaryNumber2 > lastImaginaryr then
  ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text) - Rate;
ImaginaryNumber.Text := FloatToStr(ImaginaryNumber2);
ZCubedGenerator.Active := True;
if (RealNumber3 = LastRealr) and (ImaginaryNumber2 = LastImaginaryr) then
begin
  Timer1.Enabled := False;
end;
end;
if Z4Box.Checked = True then
 //If the user chooses the function F(Z)=z^4 + c then initialise the following statements
begin
 Rate := Strtofloat(RateBox.Text);
 //The amount the constant is changed by is obtained from the user
 ZSquaredGenerator.Active := False;
 //Color map series for function F(Z)=z^2 + c is set to false
 ZCubedGenerator.Active := False;
 //Color map series for function F(Z)=z^3 + c is set to false
 Z4Generator.Active := False;
 //Color map series for function F(Z)=z^4 + c is set to false
 if RealNumber3 < LastRealr then
  RealNumber3 := StrToFloat(RealNumber.Text) + Rate;
 RealNumber.Text := FloatToStr(RealNumber3);
```

```
  if ImaginaryNumber2 < lastImaginaryr then
    ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text) + Rate;
  ImaginaryNumber.Text := FloatToStr(ImaginaryNumber2);
  if RealNumber3 > LastRealr then
    RealNumber3 := StrToFloat(RealNumber.Text) - Rate;
  RealNumber.Text := FloatToStr(RealNumber3);
  if ImaginaryNumber2 > lastImaginaryr then
    ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text) - Rate;
  ImaginaryNumber.Text := FloatToStr(ImaginaryNumber2);
  Z4Generator.Active := True;
  if (RealNumber3 = LastRealr) and (ImaginaryNumber2 = LastImaginaryr) then
  begin
    Timer1.Enabled := False;
  end;
 end;
end;

procedure TForm1.StartAnimationButtonClick(Sender: TObject);
//This procedure is used when the button needed to start the animaiton is clicked
begin
 Timer1.Enabled := True;//Used to set the timer to true meaning it is now active
 IterationsMax := StrToInt(Iterations.Text);
 //This obtains the maximum number of iterations from the user
 LastRealr := strtofloat(LastReal.Text);//The last real number is obtained
 LastImaginaryr := strtofloat(LastImaginary.Text);
 //The Last imaginary Number is obtained
 RealNumber3 := StrToFloat(RealNumber.Text);//The intial real number is obtained
 ImaginaryNumber2 := StrToFloat(ImaginaryNumber.Text);
 //The initial Imaginary number is obtained
```

```
end;

procedure TForm1.RealNumberEnter(Sender: TObject);
begin
  Timer1.Enabled := False;
  //The timer is set to false so it is now inactive if the user tries to change the number midway through
end;

procedure TForm1.StopAnimationClick(Sender: TObject);
begin
  Timer1.Enabled := False;
  //The timer is set to false so it is now inactive and the animations has stopped
end;

procedure TForm1.ImaginaryNumberEnter(Sender: TObject);
begin
  Timer1.Enabled := False;
  //The timer is set to false so it is now inactive and the aniamtions has stopped
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  PopulateColorSource; // this procedure is used to give the Z coordinates their colour
end;

procedure TForm1.PopulateColorSource;
const
  DUMMY = 0.0;
begin
```

```
with ListChartSource1 do // Each colour below is assigned a value
begin
  Add(-1.0, DUMMY, '', clBlack);
  //If a Z point is in the set the point is assigned the colour black
  Add(0.0, DUMMY, '', clWhite);
  //All of the colour below are determined by deviding the number of iterations of the function by the
maximum number of iterations
  Add(0.2, DUMMY, '', clBlue);
  Add(0.4, DUMMY, '', clPurple);
  Add(0.6, DUMMY, '', clRed);
  Add(0.8, DUMMY, '', clGreen);
  Add(1, DUMMY, '', clYellow);
end;
with ListChartSource2 do
begin
  Add(-1.0, DUMMY, '', clBlack);
  Add(0.0, DUMMY, '', clRed);
  Add(0.1, DUMMY, '', clYellow);
  Add(0.2, DUMMY, '', clGreen);
  Add(0.3, DUMMY, '', clblue);
  Add(0.4, DUMMY, '', clPurple);

end;
with ListChartSource3 do
begin
  Add(-1.0, DUMMY, '', clBlack);
  Add(0.0, DUMMY, '', clNavy);
  Add(0.2, DUMMY, '', clBlue);
  Add(0.4, DUMMY, '', clAqua);
```

```
  Add(0.6, DUMMY, '', clWhite);


 end;
end;


procedure TForm1.Button7Click(Sender: TObject);
//This procedure is used to produce julia sets using a White colour scheme
begin
 ZSquaredGenerator.ColorSource := ListChartSource1;
 //This chart source has white colour scheme
 ZCubedGenerator.ColorSource := ListChartSource1;
 Z4Generator.ColorSource := ListChartSource1;
end;





procedure TForm1.Button8Click(Sender: TObject);
//This procedure is used to produce julia sets using a red colour scheme
begin
 ZSquaredGenerator.ColorSource := ListChartSource2;
 //This chart source has red colour scheme
 ZCubedGenerator.ColorSource := ListChartSource2;
 Z4Generator.ColorSource := ListChartSource2;
end;


procedure TForm1.Button9Click(Sender: TObject);
//This procedure is used to produce julia sets using a blue colour scheme
begin
```

```
 ZSquaredGenerator.ColorSource := ListChartSource3;
 //This chart source has blue colour scheme
 ZCubedGenerator.ColorSource := ListChartSource3;
 Z4Generator.ColorSource := ListChartSource3;
end;


procedure GetZPath(ConstantR, ConstantI: real);
//This procedure is used to calculate all the Z points needed to produce the Z path
begin
 for Count := 2 to 30 do //The for loop ends when all 30 Z points have been calculated
 begin
  ZReal[Count] := ((ZReal[Count - 1] * ZReal[Count - 1]) -
   (ZImaginary[Count - 1] * ZImaginary[Count - 1]) + ConstantR);
  //The Z point's real number is operated on by the function and the result is stored in the array
  ZImaginary[Count] := ((2 * ZReal[Count - 1] * ZImaginary[Count - 1]) + ConstantI);
  //The Z point's imaginary number is operated on by the function and the result is stored in the array
 end;
end;


procedure TForm1.GenerateZPathClick(Sender: TObject);
// This procedure is used to actually plot the points
var
 ConstantR, ConstantI: real;
 //Real and imaginary parts of the constant defined by the user
begin
```

```
ZPathGeneratorlineseries1.Clear; //The Z path is reset
ZReal[1] := StrToFloat(ZRealNumber.Text);
//The Z point's initial Real Number is obtained
ZImaginary[1] := StrToFloat(ZImaginaryNumber.Text);
//The Z point's initial Imaginary Number is obtained
ConstantR := StrToFloat(RealNumber.Text); //The constant's real number is obtained
ConstantI := StrToFloat(ImaginaryNumber.Text);
//The constant's Imaginary number is obtained
GetZPath(ConstantR, ConstantI);
//The procedure usef to calculate the Z points for plotting is initiated
for Count := 1 to 30 do
begin
 ZPathGeneratorLineSeries1.addxy(ZReal[Count], ZImaginary[Count]);
 //All 30 of the points are plotted
end;
Label12.Caption := floattostr(ZReal[1]);
//The Real Number of the first few plotted points are converted into a string and are shown on screen
Label13.Caption := floattostr(ZReal[2]);
Label14.Caption := floattostr(ZReal[3]);
Label15.Caption := floattostr(ZReal[4]);
Label16.Caption := floattostr(ZReal[5]);
Label17.Caption := floattostr(ZReal[6]);
Label18.Caption := floattostr(ZReal[7]);
Label19.Caption := floattostr(ZImaginary[1]) + 'i';
//The Imaginary Number of the first few plotted points are converted into a string and are shown on screen
Label20.Caption := floattostr(ZImaginary[2]) + 'i';
Label21.Caption := floattostr(ZImaginary[3]) + 'i';
Label22.Caption := floattostr(ZImaginary[4]) + 'i';
Label24.Caption := floattostr(ZImaginary[5]) + 'i';
```

```
 Label25.Caption := floattostr(ZImaginary[6]) + 'i';
 Label26.Caption := floattostr(ZImaginary[7]) + 'i';
end;



procedure TForm1.DecreaseZRealClick(Sender: TObject);
//This procedure increases the Real number of the user defined Z value when a specific button is clicked

var
 ConstantR, ConstantI: real;
begin
 ZPathGeneratorlineseries1.Clear;
 ZPathGeneratorLineSeries1.Active := False;
 ZReal[1] := StrToFloat(ZRealNumber.Text) - 0.05;
 //The Z value's real Number is decreased by 0.05
 ZImaginary[1] := StrToFloat(ZImaginaryNumber.Text);
 ZRealNumber.Text := FloattoStr(ZReal[1]);
 ConstantI := StrToFloat(ImaginaryNumber.Text);
 ConstantR := StrToFloat(RealNumber.Text);
 GetZPath(ConstantR, ConstantI);
 for Count := 1 to 30 do
 begin
  ZPathGeneratorLineSeries1.addxy(ZReal[Count], ZImaginary[Count]);
 end;
 ZPathGeneratorLineSeries1.Active := True;
 Label12.Caption := floattostr(ZReal[1]);
 Label13.Caption := floattostr(ZReal[2]);
 Label14.Caption := floattostr(ZReal[3]);
```

```
  Label15.Caption := floattostr(ZReal[4]);
  Label16.Caption := floattostr(ZReal[5]);
  Label17.Caption := floattostr(ZReal[6]);
  Label18.Caption := floattostr(ZReal[7]);
  Label19.Caption := floattostr(ZImaginary[1]) + 'i';
  Label20.Caption := floattostr(ZImaginary[2]) + 'i';
  Label21.Caption := floattostr(ZImaginary[3]) + 'i';
  Label22.Caption := floattostr(ZImaginary[4]) + 'i';
  Label24.Caption := floattostr(ZImaginary[5]) + 'i';
  Label25.Caption := floattostr(ZImaginary[6]) + 'i';
  Label26.Caption := floattostr(ZImaginary[7]) + 'i';
end;

procedure TForm1.IncreaseReal1Click(Sender: TObject);
//This procedure decreases the Real number of the user defined Z value when a specific button is clicked
var
  ConstantR, ConstantI: real;
begin
  ZPathGeneratorlineseries1.Clear;
  ZReal[1] := StrToFloat(ZRealNumber.Text) + 0.05;
  //The Z value's real Number is increased by 0.05
  ZImaginary[1] := StrToFloat(ZImaginaryNumber.Text);
  ZRealNumber.Text := FloattoStr(ZReal[1]);
  ConstantI := StrToFloat(ImaginaryNumber.Text);
  ConstantR := StrToFloat(RealNumber.Text);
  GetZPath(ConstantR, ConstantI);
  for Count := 1 to 30 do
  begin
    ZPathGeneratorLineSeries1.addxy(ZReal[Count], ZImaginary[Count]);
```

```
  end;
  Label12.Caption := floattostr(ZReal[1]);
  Label13.Caption := floattostr(ZReal[2]);
  Label14.Caption := floattostr(ZReal[3]);
  Label15.Caption := floattostr(ZReal[4]);
  Label16.Caption := floattostr(ZReal[5]);
  Label17.Caption := floattostr(ZReal[6]);
  Label18.Caption := floattostr(ZReal[7]);
  Label19.Caption := floattostr(ZImaginary[1]) + 'i';
  Label20.Caption := floattostr(ZImaginary[2]) + 'i';
  Label21.Caption := floattostr(ZImaginary[3]) + 'i';
  Label22.Caption := floattostr(ZImaginary[4]) + 'i';
  Label24.Caption := floattostr(ZImaginary[5]) + 'i';
  Label25.Caption := floattostr(ZImaginary[6]) + 'i';
  Label26.Caption := floattostr(ZImaginary[7]) + 'i';
end;

procedure TForm1.IncreaseZImaginaryClick(Sender: TObject);
//This procedure increases the Imaginary number of the user defined Z value when a specific button is clicked
var
  ConstantR, ConstantI: real;
begin
  ZPathGeneratorlineseries1.Clear;
  ZReal[1] := StrToFloat(ZRealNumber.Text);
  ZImaginary[1] := StrToFloat(ZImaginaryNumber.Text) + 0.05;
  //The Z value's imaginary Number is increased by 0.05
  ZImaginaryNumber.Text := FloattoStr(ZImaginary[1]);
  ConstantI := StrToFloat(ImaginaryNumber.Text);
  ConstantR := StrToFloat(RealNumber.Text);
```

```
  GetZPath(ConstantR, ConstantI);
  for Count := 1 to 30 do
  begin
   ZPathGeneratorLineSeries1.addxy(ZReal[Count], ZImaginary[Count]);
  end;
  Label12.Caption := floattostr(ZReal[1]);
  Label13.Caption := floattostr(ZReal[2]);
  Label14.Caption := floattostr(ZReal[3]);
  Label15.Caption := floattostr(ZReal[4]);
  Label16.Caption := floattostr(ZReal[5]);
  Label17.Caption := floattostr(ZReal[6]);
  Label18.Caption := floattostr(ZReal[7]);
  Label19.Caption := floattostr(ZImaginary[1]) + 'i';
  Label20.Caption := floattostr(ZImaginary[2]) + 'i';
  Label21.Caption := floattostr(ZImaginary[3]) + 'i';
  Label22.Caption := floattostr(ZImaginary[4]) + 'i';
  Label24.Caption := floattostr(ZImaginary[5]) + 'i';
  Label25.Caption := floattostr(ZImaginary[6]) + 'i';
  Label26.Caption := floattostr(ZImaginary[7]) + 'i';
end;


procedure TForm1.DecreaseZImaginaryClick(Sender: TObject);
//This procedure decreases the Imaginary number of the user defined Z value when a specific button is clicked
var
  ConstantR, ConstantI: real;
begin
  ZPathGeneratorlineseries1.Clear;
```

```
ZReal[1] := StrToFloat(ZRealNumber.Text);
ZImaginary[1] := StrToFloat(ZImaginaryNumber.Text) - 0.05;
//The Z value's imaginary Number is decreased by 0.05
ZImaginaryNumber.Text := FloattoStr(ZImaginary[1]);
ConstantI := StrToFloat(ImaginaryNumber.Text);
ConstantR := StrToFloat(RealNumber.Text);
GetZPath(ConstantR, ConstantI);
for Count := 1 to 30 do
begin
 ZPathGeneratorLineSeries1.addxy(ZReal[Count], ZImaginary[Count]);
end;
Label12.Caption := floattostr(ZReal[1]);
Label13.Caption := floattostr(ZReal[2]);
Label14.Caption := floattostr(ZReal[3]);
Label15.Caption := floattostr(ZReal[4]);
Label16.Caption := floattostr(ZReal[5]);
Label17.Caption := floattostr(ZReal[6]);
Label18.Caption := floattostr(ZReal[7]);
Label19.Caption := floattostr(ZImaginary[1]) + 'i';
Label20.Caption := floattostr(ZImaginary[2]) + 'i';
Label21.Caption := floattostr(ZImaginary[3]) + 'i';
Label22.Caption := floattostr(ZImaginary[4]) + 'i';
Label24.Caption := floattostr(ZImaginary[5]) + 'i';
Label25.Caption := floattostr(ZImaginary[6]) + 'i';
Label26.Caption := floattostr(ZImaginary[7]) + 'i';

end;

end.
```

Julia Set End User Guide

How to install the program

The installation process is relatively simple, and these next steps should help you progress with it.

1. Please access the storage device or email you will be obtaining the program's files from.

2. If you are obtaining the files via email, please access the attached file and download it. Once downloaded please open the library where it has been downloaded. If from an external hard drive, drag the file into a chosen library.

3. Open the file, and you will see an executable name 'Julia Set Generator', you will need to right click on this executable and click 'Create short cut'. Once clicked, a short cut should be directly below the executable.

4. Drag the shortcut onto the desktop for easy access. This was the final stage and is now ready to be accessed.

How to operate the program

Contents

Part 1 - Julia Set Generation

This page explains how to generate Julia sets using this program by defining a constant and the maximum number of iterations the function is allowed use on the constant and Z values.

Part 2 - Creating a Julia Set animation

This page involves defining the initial constant and also the last real and imaginary number the animation is meant to end on. This in turn can produce very fascinating animations of A Julia set with a changing constant.

Part 3 - Changing the colour of the set

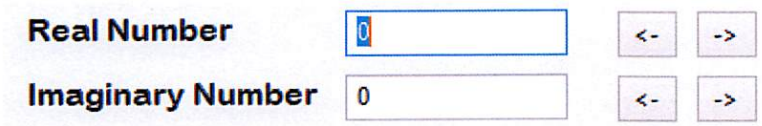
This part explains how to alter the colour style of the Julia sets

Part 4 - Generating the path of a Z coordinate

The final part shows how to generate the path of a user defined Z value and constant.

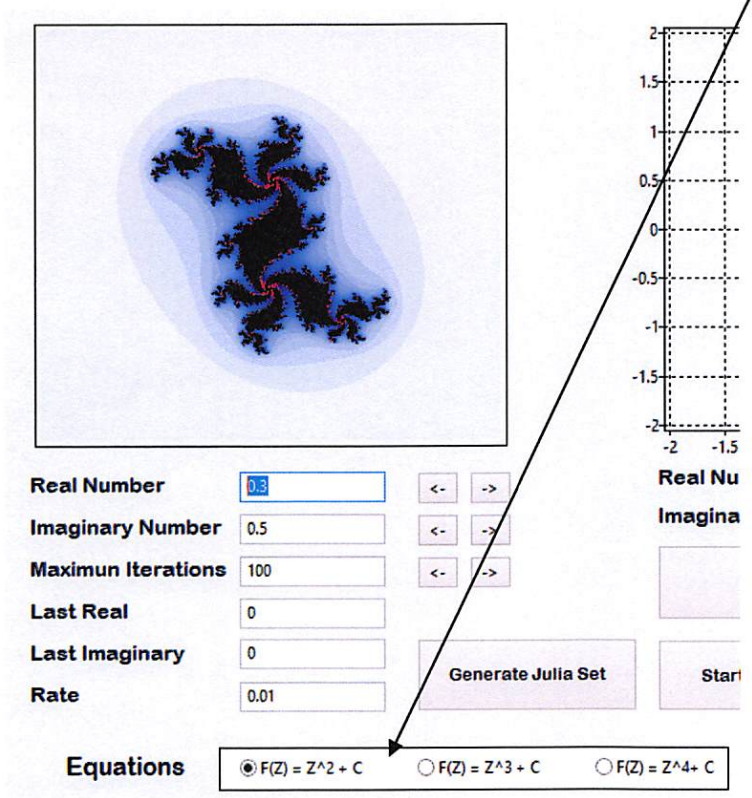
Part 1 - Julia Set Generation

1. The first step to generating a Julia set is to define a constant. As the constant consists of a real and imaginary number, enter suitable values for both in the boxes provided. You are also free to increase or decrease the real and imaginary number of the constant using the Arrows below.



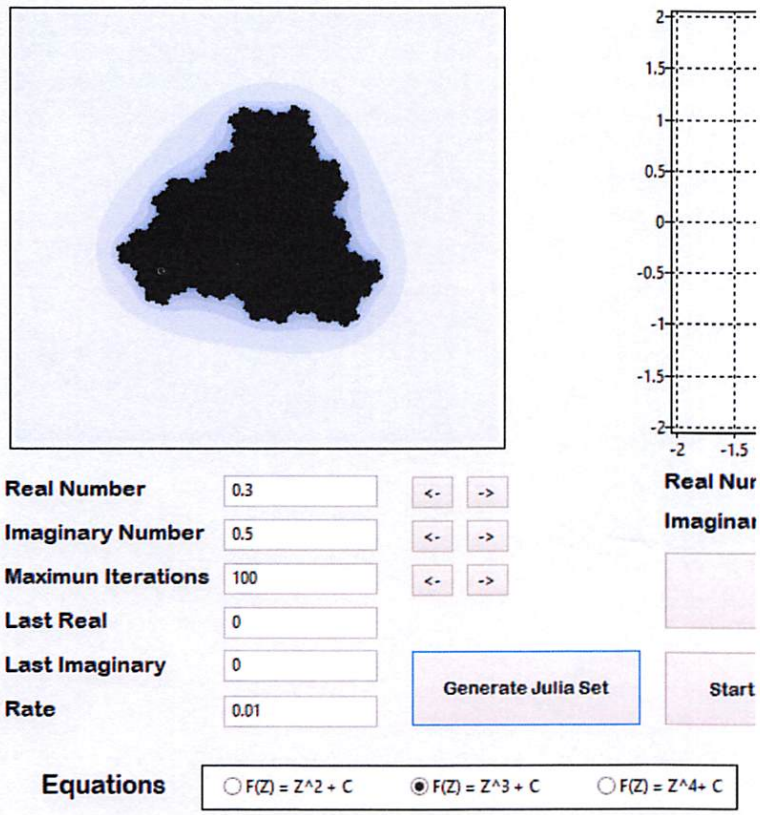
2. You are also able to change the overall depth of the image by altering the maximum number of iterations allowed. Note, increasing this value will reduce the performance drastically.

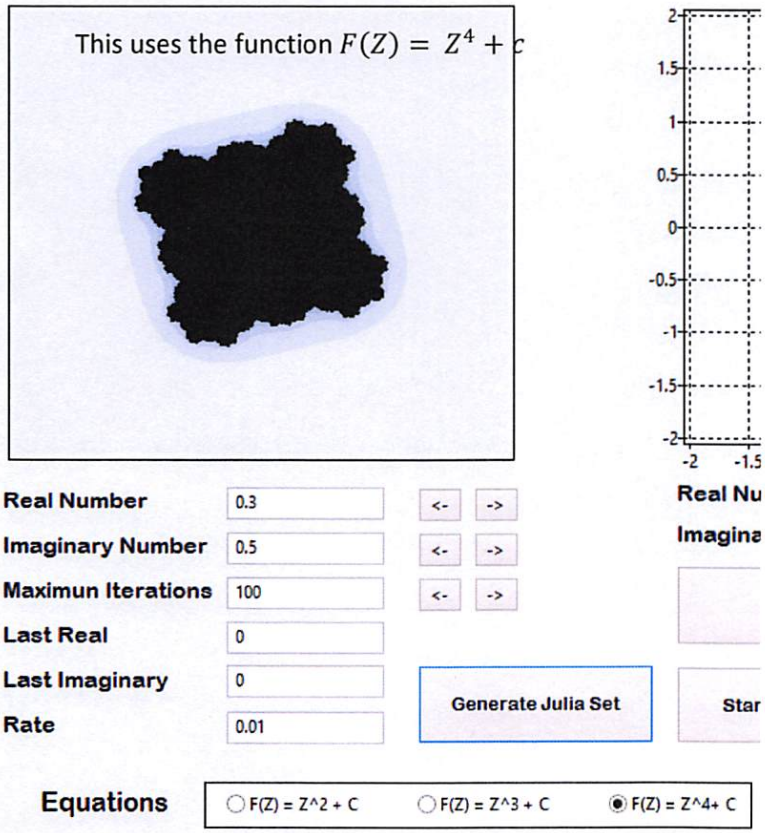
3. Once you have decided on the constant and the maximum number of iterations press the 'Generate Julia Set' button and the set will be generated. It is also possible to change the function used in the generation of the Julia by changing ~~the~~ which box is checked below. The example in the screenshot uses the constant 0.3 + 0.5i with 100 iterations.



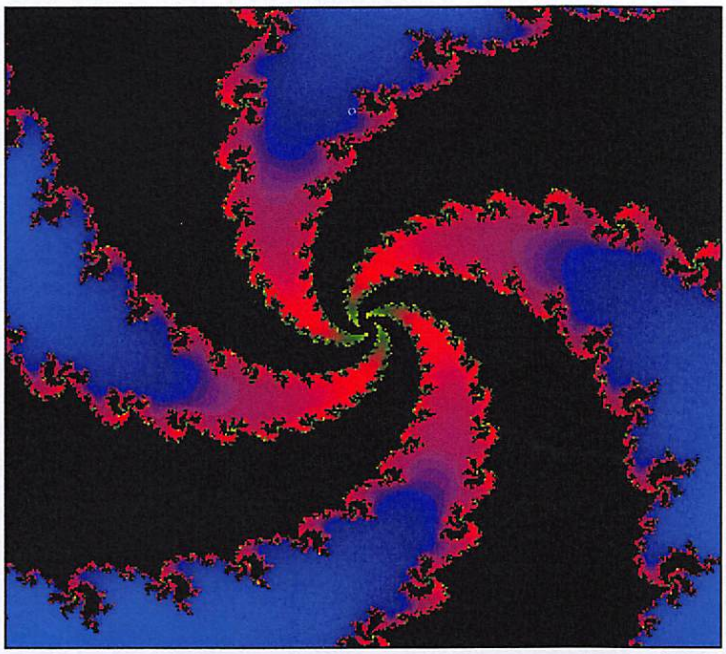
This uses the function $F(Z) = Z^2 + c$

This uses the function $F(Z) = Z^3 + c$



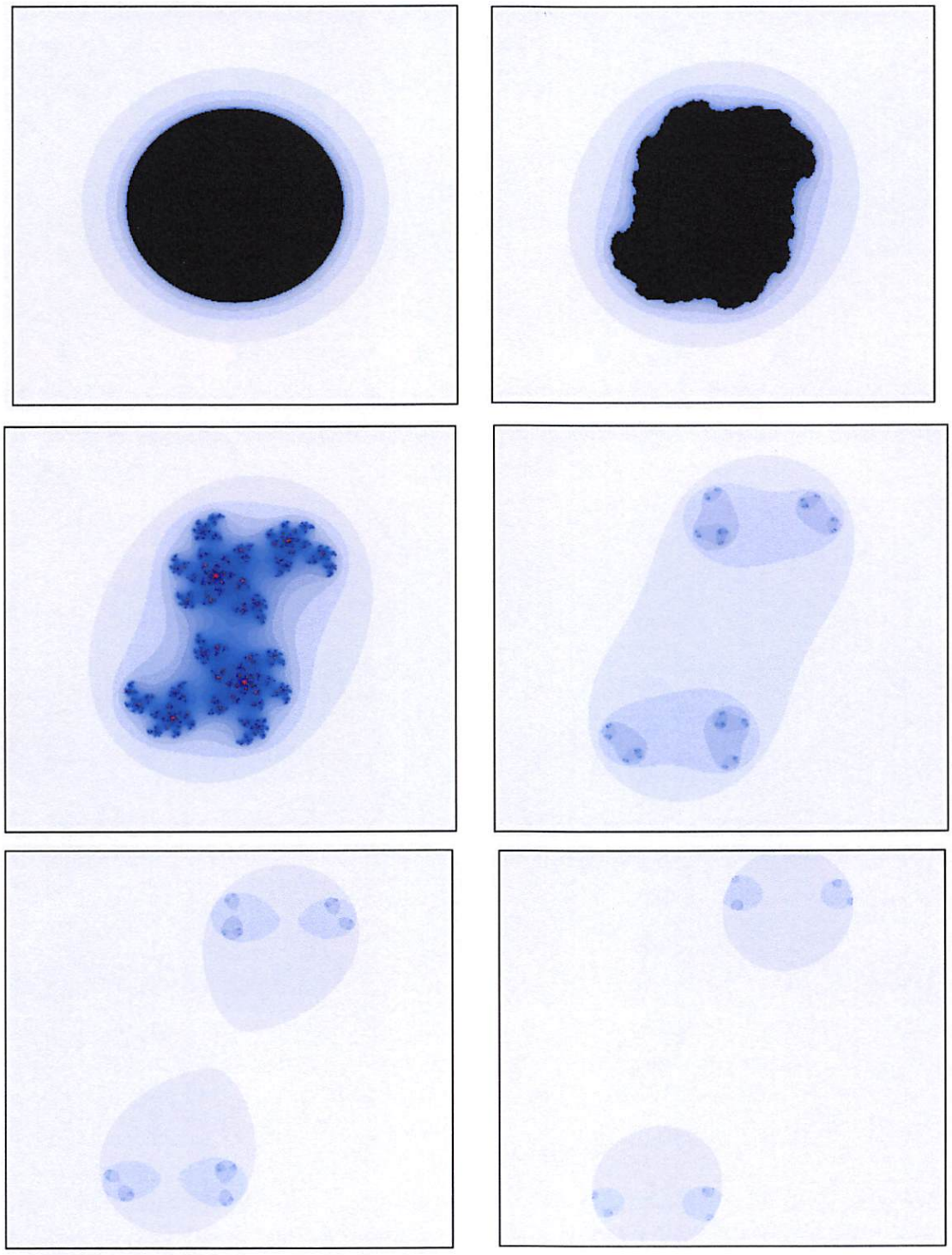


4. Once generated you are now able to z zoom and navigate the set. You can zoom into the image by simply right clicking and dragging downwards and to the right(Note try to drag in perfect squares so the image does not become distorted. To zoom out, do the opposite by dragging upwards to the left. To navigate simple right click and drag.



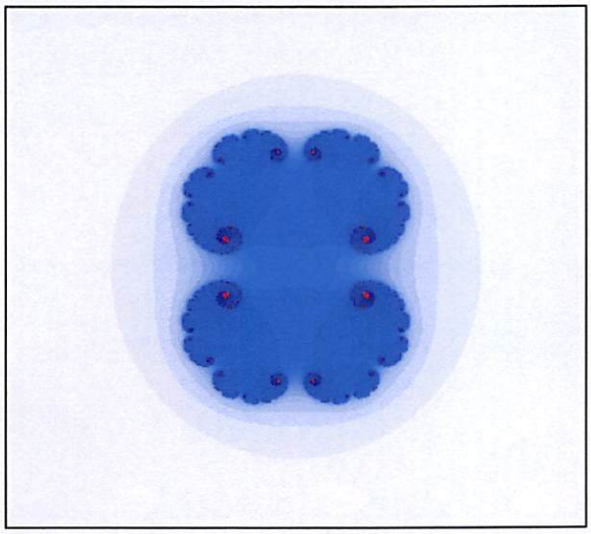
Part 2 - Julia Set Animation

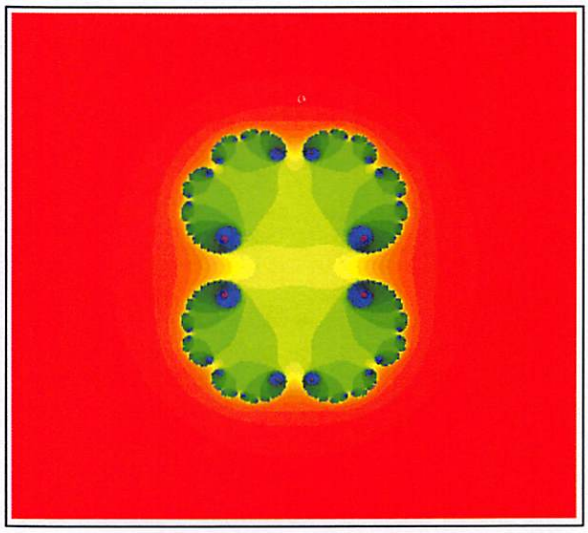
In order to produce an animation of a changing Julia set all you need to do is add a final real and imaginary number for the animation to finish on. You can do this by typing in sensible values into the labelled boxes. You are also able to choose the speed of the animation by altering the amount it changes by each interval. In this example the constant starts of as 0 +0i and will travel to the last constant 2 -2i. To start the animation click 'start animation' and to end it at any stage click 'end animation'.

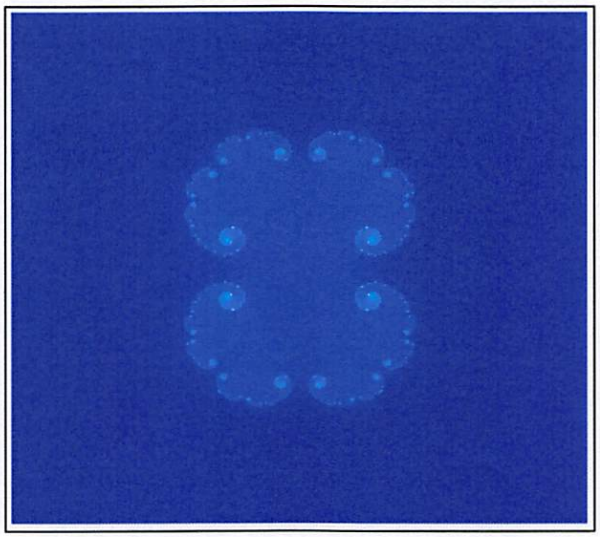


Part 3 - Colour Scheme Changing

For this simply click the button of the colour scheme you would like to see.

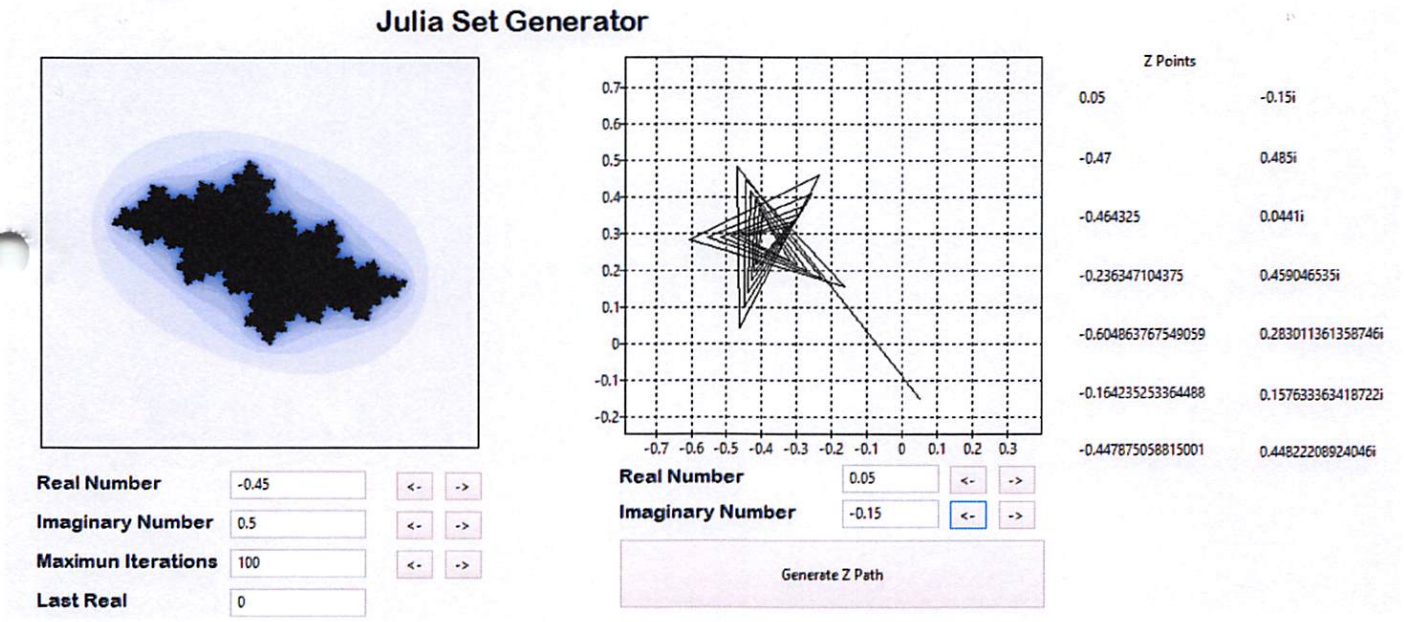






Part 4 - Generation of a Z value's path

To generate the path of a defined Z coordinate you first need to define the constant which can be entered into the same boxes used to generate the Julia set. You then need to define real and imaginary part of you chosen Z value. Once defined click the 'Generate Z path' button which in turn will generate the path. To the right you can see the values of the first few points and not forgetting the actual path which should show very chaotic behaviour depending on what constant and z value you chose.



Testing

Introduction

This section will demonstrate what the code I have written does so I am able to determine whether the outcome is correct or needs changing. If it needs changing I will attempt to edit the existing code so it would now produce a more suitable outcome. All of the test I have performed in this section are all necessary for the success of my program and also to determine whether all of my objectives have been met or not.

Testing strategy

1 - Julia Set Generation

This test will be used to determine that when a complex number is entered will it be able to produce the correct set. I will do this by entering a positive Complex number, a negative complex number and then a combination of the two, meaning a negative real number and a positive imaginary number. I will then check to see if the Julia set my program produces is the same as the Julia set produced in a different Julia set generator. If the same I would give the test a pass.

2 - Functions

This test will be used to determine if the user is able to produce accurate Julia sets with the 2 alternative functions provided by the program. I was notable to find any online Julia set generators which were able to produce Julia sets with an increasing value of Z, so instead I'm going to search for images of Julia sets produced by the alternate functions and then enter the same values into my program to see If I get the same result. Surprisingly I only found 2 images of Julia sets produced by the functions $F(z) = z^3 + c$and $F(z) = z^4 + c$.

3- Animation

For this test I will determine if an animation ends on the correct value determined by the user. I will do this very similarly to the first test by entering values for the last real and last imaginary number including a positive, negative and mixed complex number. If the last Julia set uses the same complex number as what the user typed in for the last complex number I will give the test a pass.

4 - Validation System

This will be used to determine that if a different data type is entered into a box other than the intended data type, it would automatically be set to 0. This will be done by first testing if the intended data types work correctly and then testing if a different data type crashes the program or turns to 0. In the case of the maximum number of iterations, if the user enters a

negative number the program should automatically turn to 100 so the Julia set is still generated.

5 - Z - Path

This will be used to see if the Path generator is capable of producing clear and accurate Z paths using a user defined Z complex number and Constant. I will test this by using multiple constants and Z values to see if a Z path is produced. I will then purposefully type in a Z value which escapes towards infinity to see if a Z path is still produced.

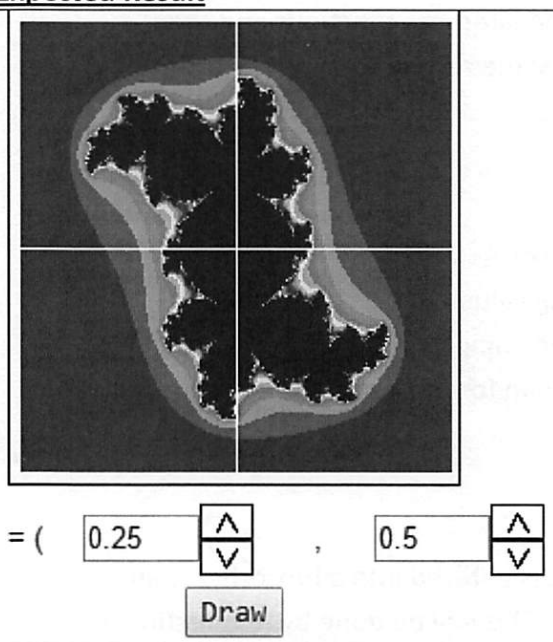
6- Colour Scheme Changing

This test very simply involves me checking the box for each colour scheme and then checking if the Julia set produced possess the colour scheme I have chosen.

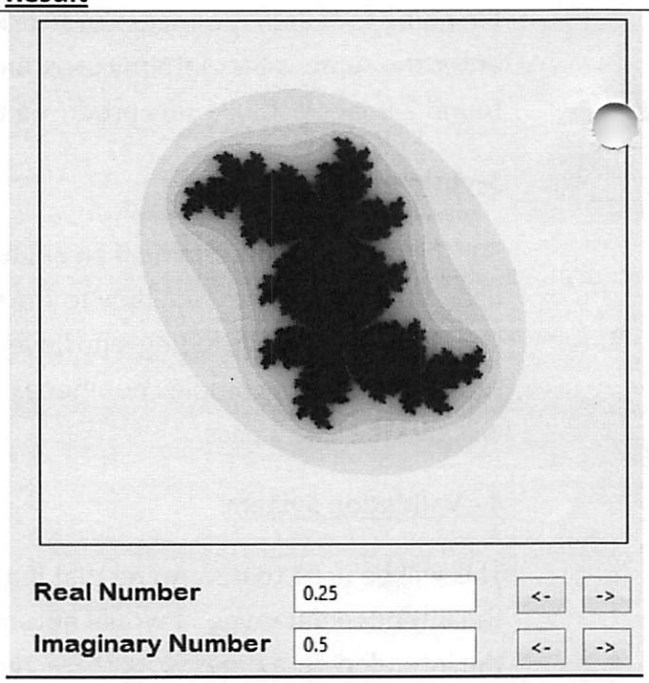
Test Section

Test Number	Description	Data	Expected Result	Pass/Fail	Result
1.1	This will test to see if the complex number 0.25 + 0.5i is able to produce the correct Julia set	0.25 + 0.5i	A Julia Set with the constant 0.25 0.5i	Pass	Below

Expected Result

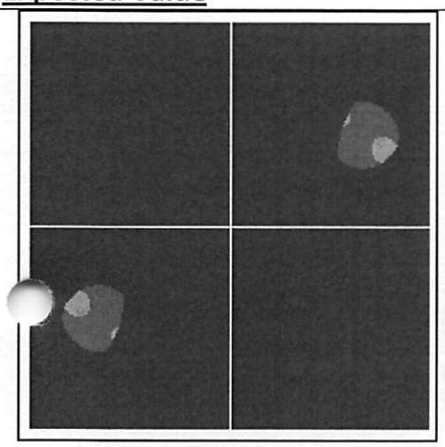


Result

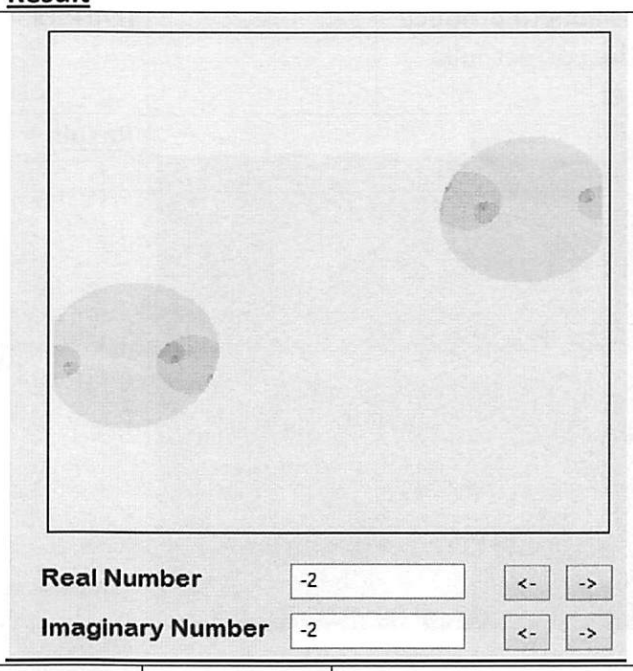


Test Number	Description	Data	Expected Result	Pass/Fail	Result
1.2	This will test to see if the complex number -2 2 - 0.5i is able to produce the correct Julia set	-2 - 2i	I expect to a Julia set with the constant -2 - 2i	Pass	Below

Expected Value

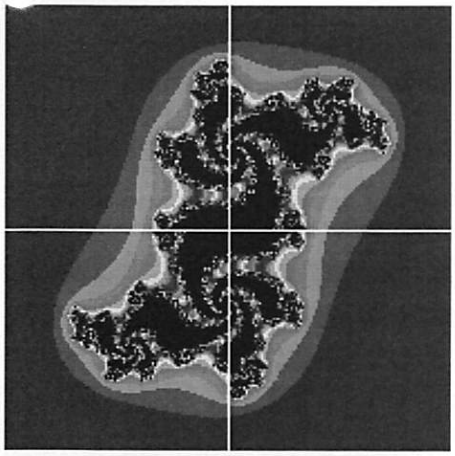


Result

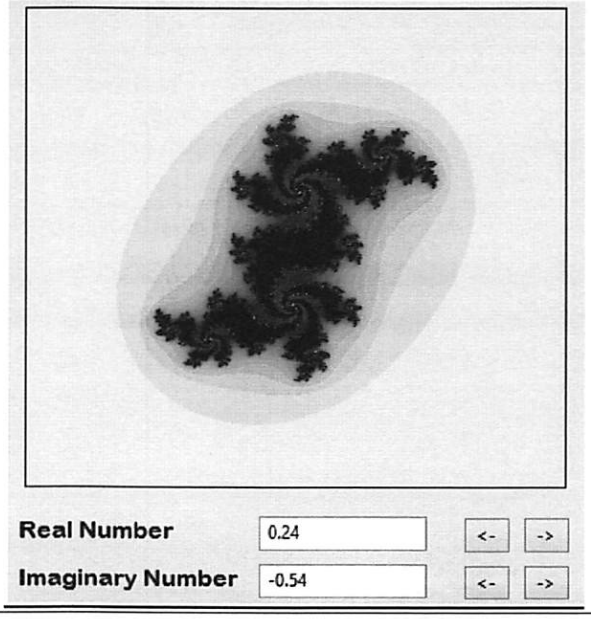


1.3	This will test to see if the complex number 0.24 - 0.54i is able to produce the correct Julia set	0.24 -0.54i	A Julia set with the constant 0.24 -0.54i	Pass	Below

Expected Value

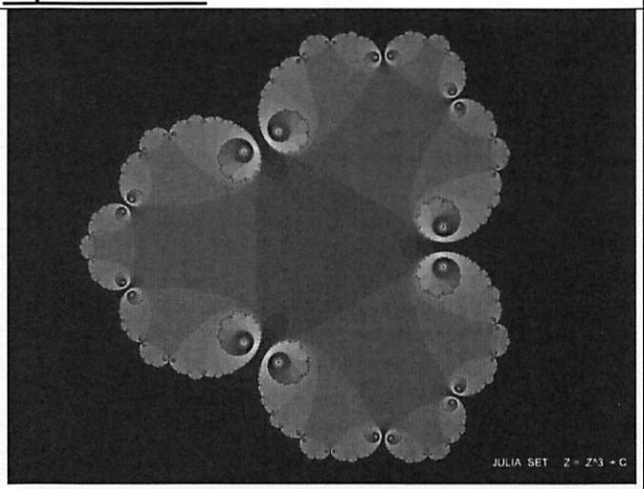


Result



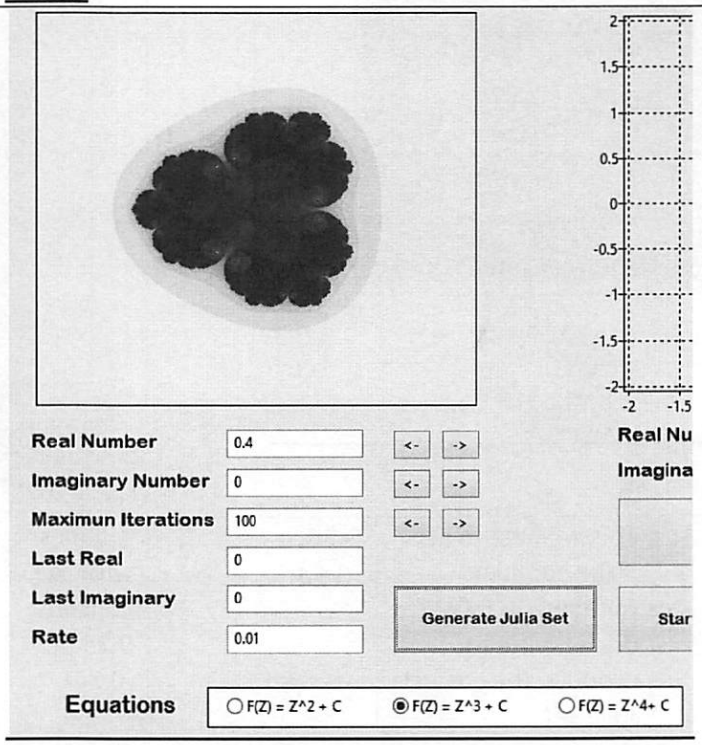
Test Number	Description	Data	Expected Result	Pass/Fail	Result
2.1	This will test to see if the complex number 0.4 + 0i substituted into the function $F(z) = z^3 + c$ is able to produce the correct Julia set	$F(z) = z^3 + c$ 0.4 + 0i	I expect to see a Julia set produced with the constant 0.4 + 0i	Pass	Below

Expected Value



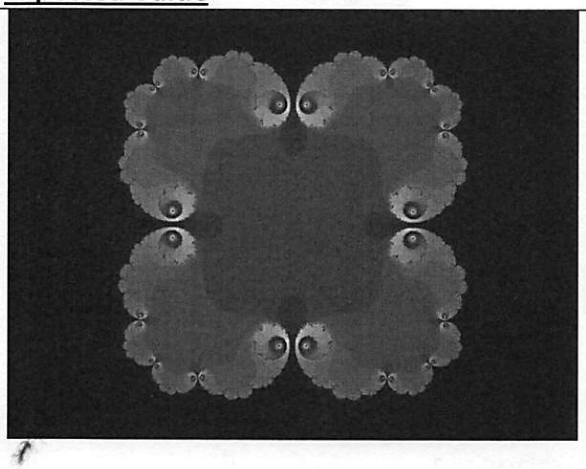
https://en.wikipedia.org/wiki/Julia_set#/media/File:JULIA3.jmb.jpg

Result



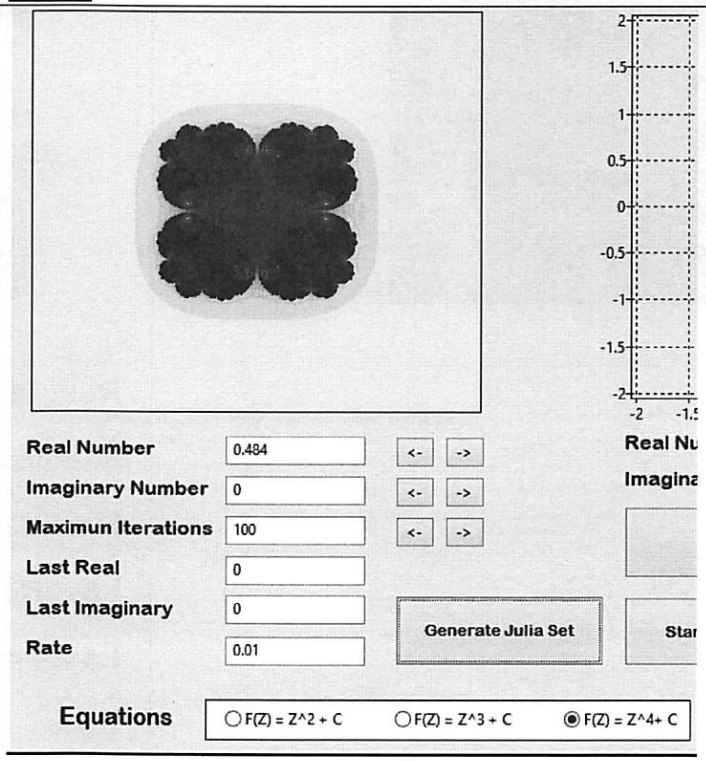
Test Number	Description	Data	Expected Result	Pass/Fail	Result
2.2	This will test to see if the complex number 0.484 + 0i substituted into the function $F(z) = z^4 + c$ is able to produce the correct Julia set	$F(z) = z^4 + c$ 0.484 + 0i	I expect to see a Julia set with the constant 0.484 + 0i	Pass	Below

Expected Value

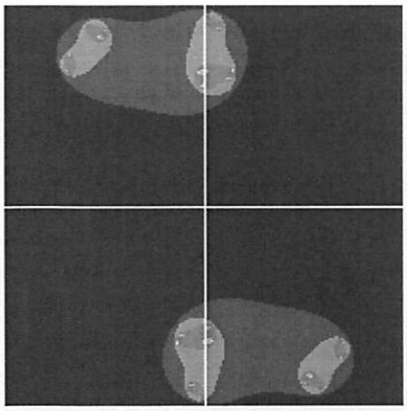
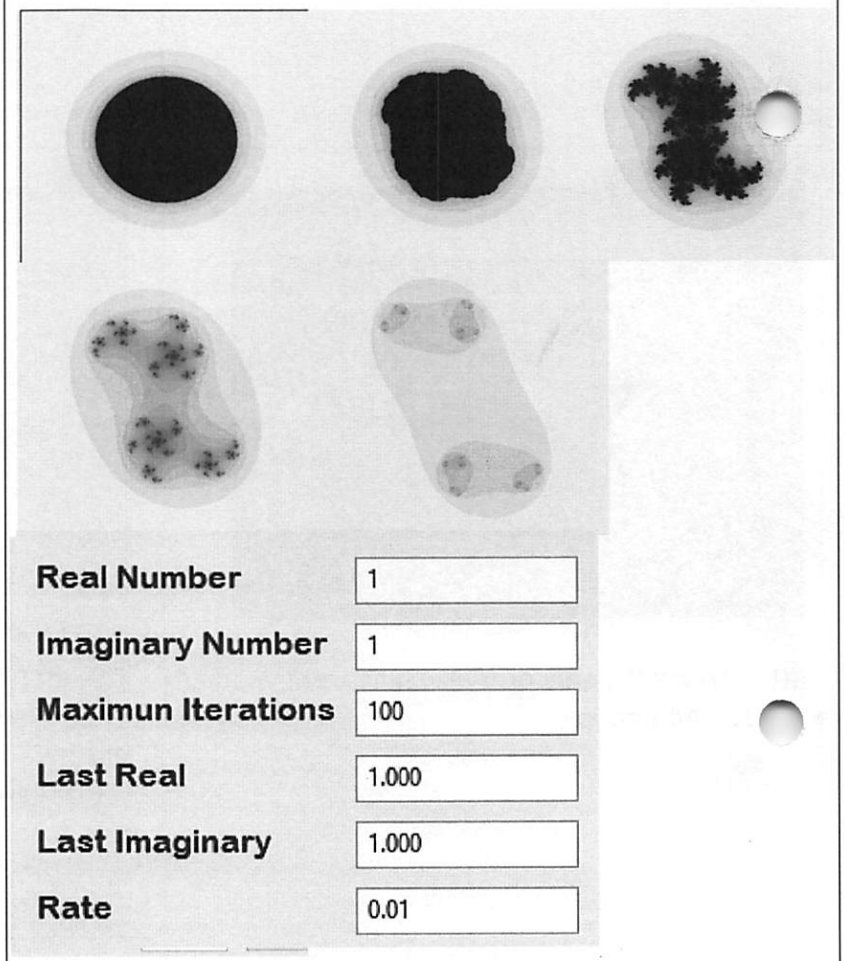


https://en.wikipedia.org/wiki/Julia_set#/media/F:JULIA4.jmb.jpg

Result

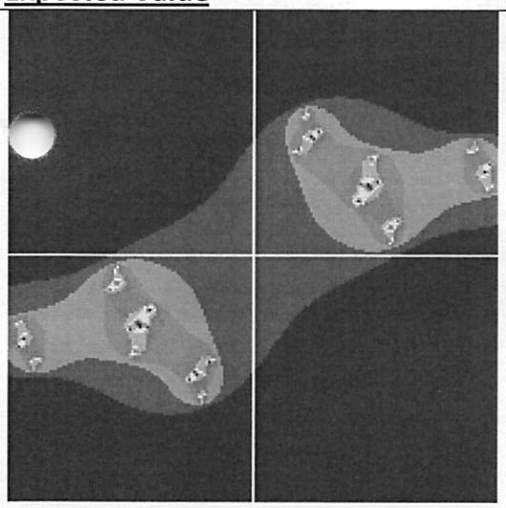


Test Number	Description	Data	Expected Result	Pass/Fail	Result
3.1	This will test to see if the animation of a changing Julia set will end on the complex number 1 + 1i	1 + 1i	I expect the animation to end on the Julia set with the constant 1+ 1i	Pass	Below

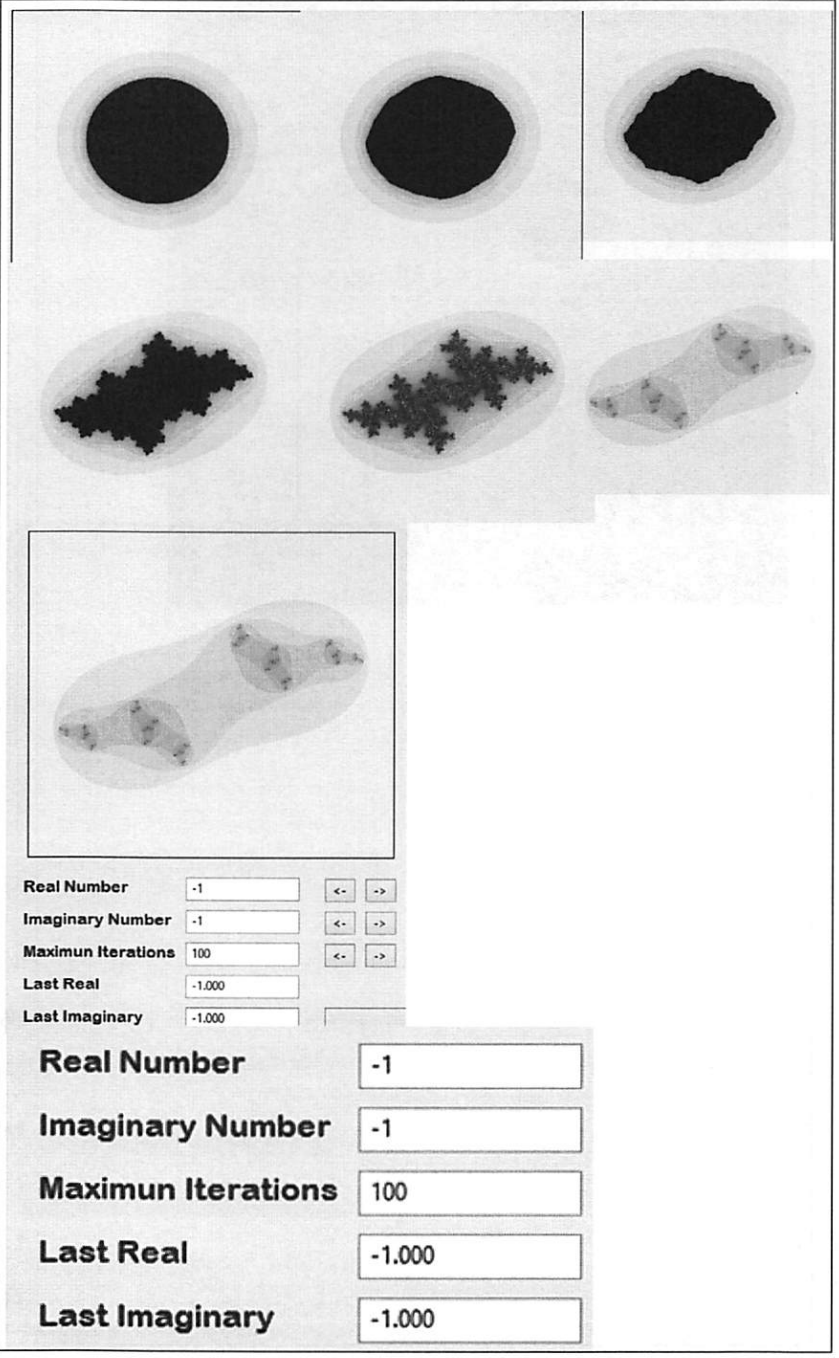
Expected Value	Result
	

Test Number	Description	Data	Expected Result	Pass /Fail	Result
3.2	This will test to see if the animation of a changing Julia set will end on the complex number -1 + -1i	-1 - 1i	I expect the animation to end on the Julia set with the constant -1 -1i.	Pass	Below

Expected Value

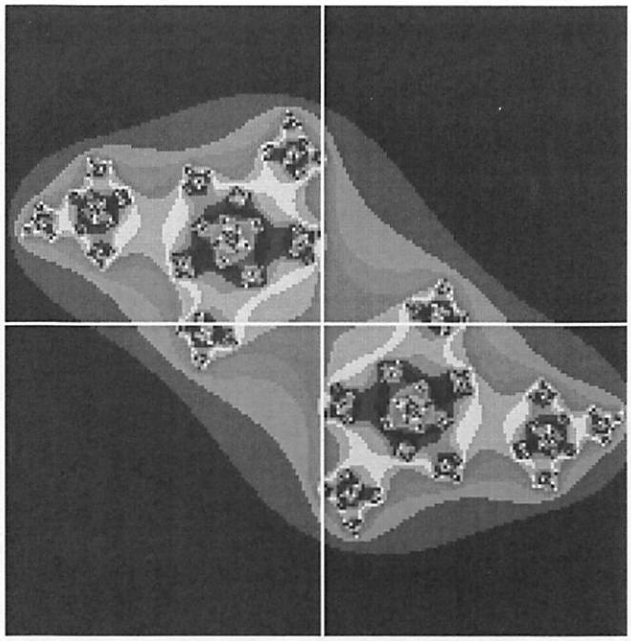


Result

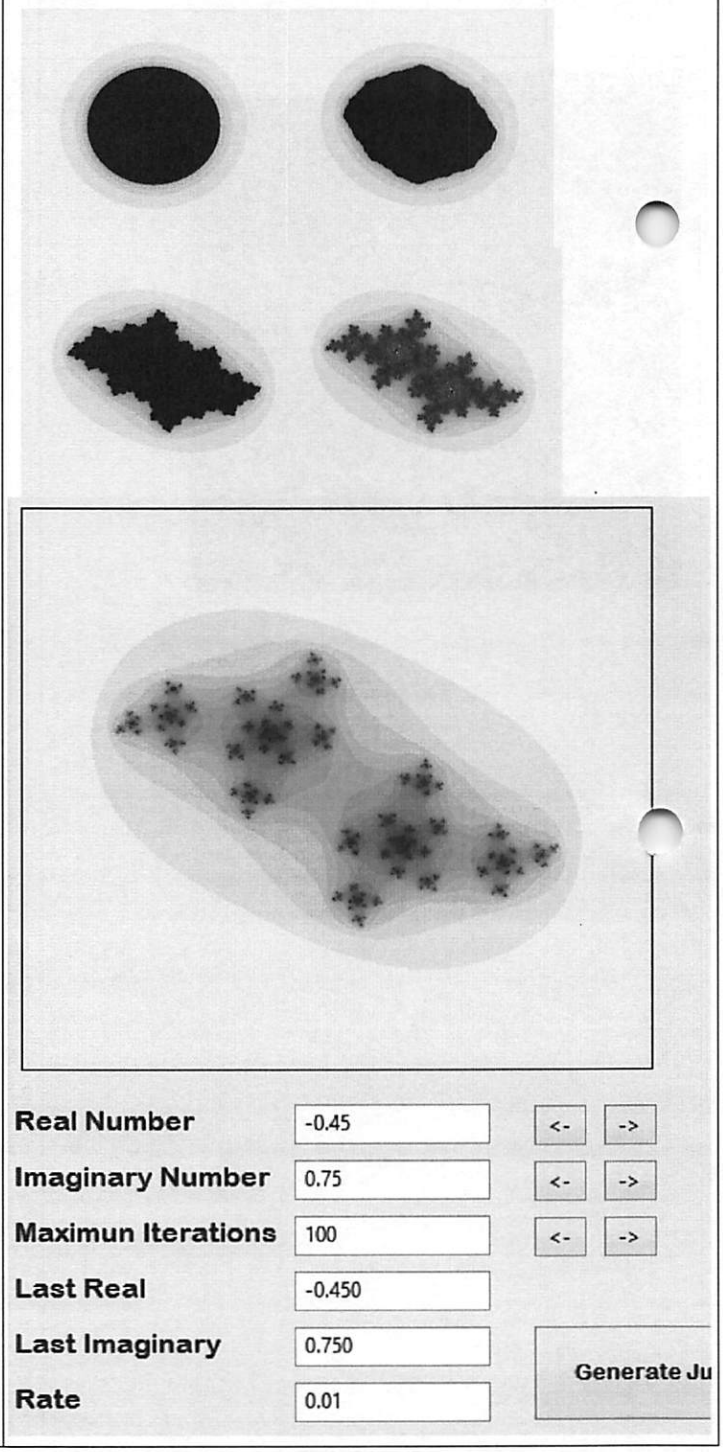


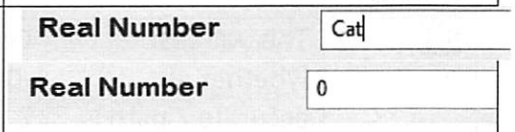
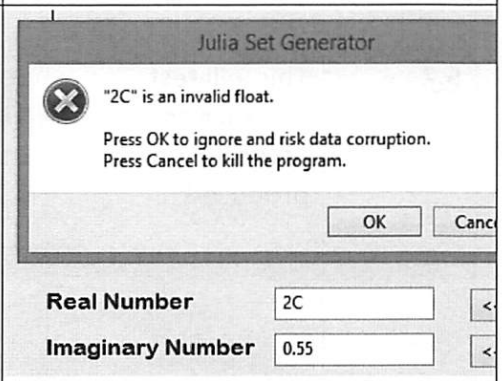
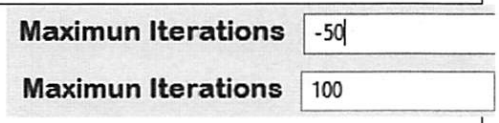
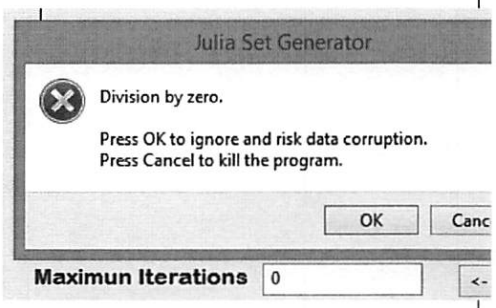
Test Number	Description	Data	Expected Result	Pass /Fail	Result
3.3	This will test to see if the animation of a changing Julia set will end on the complex number -0.45 + 0.75i	-0.45 + 0.75i	I expect the animation to end on the Julia set with the constant -0.45 + 0.75i	Pass	Below

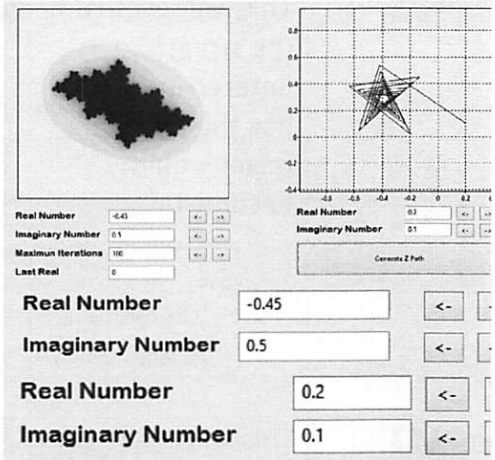
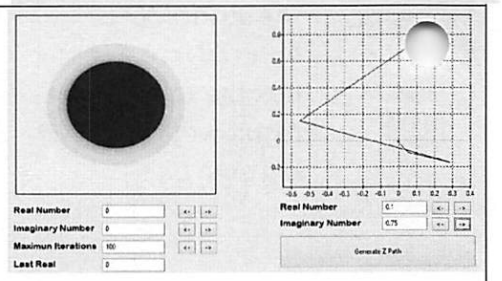
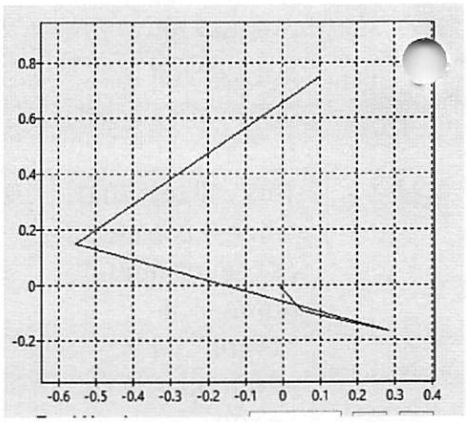
Expected Value



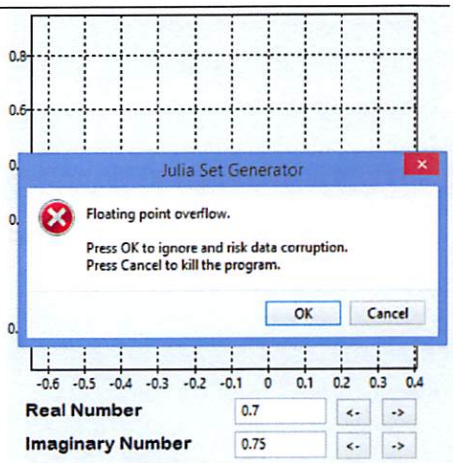
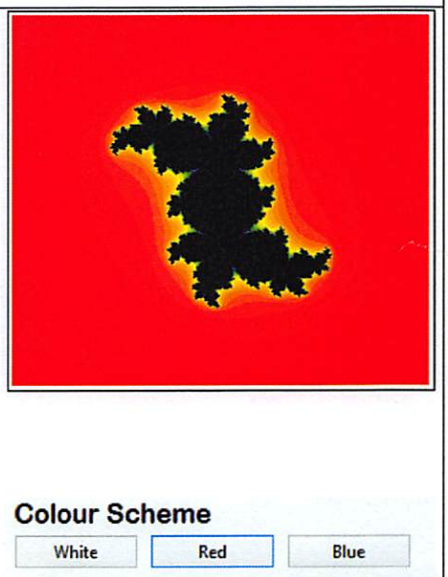
Result



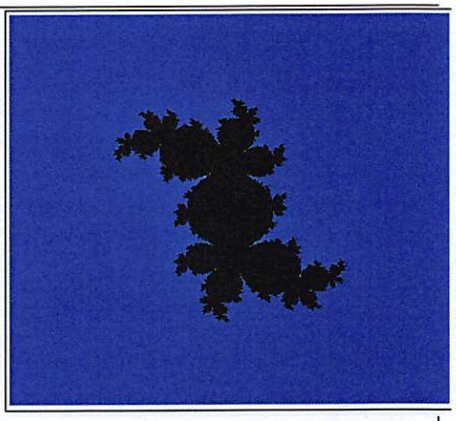
Test Number	Description	Data	Expected Result	Pass/ Fail	Result
4.11	This will test that if the string 'Cat' is entered into a text box the following result would become 0.	Cat	I expect the box which the value has been entered into to turn to 0	Pass	 For the Boxes for entering the user defined Imaginary and real number I forgot to add validation so it would produce an error message like below, anytime I typed in a string
4.12	This will test that if the string '2C' is entered into a text box the following result would become 0.	2C	I expect the box which the value has been entered into to turn to 0	Fail	
4.21	This will test that when a negative number is entered for the maximum number of iterations it is automatically set to 100	-50	For the Number of iterations to change to 100	Pass	
4.22	This will test that when 0 is entered as the maximum number of iterations it is automatically set to 100	0	For the Number of iterations to change to 100	Fail	

Test Number	Description	Data	Expected Result	Pass/ Fail	Result
5.1	This will test whether an accurate Z path is produced	Constant -0.45 + 0.5i Z Value 0.2 + 0.1i	I'm expecting a path in the shape of a 5 point star.	Pass	
5.2	This will test whether an accurate Z path is produced	Constant 0 + 0i Z Value 0.2 + 0.1i	I'm expecting a path which converges towards the value 0 + 0i.	Pass	  Visually it does appear to converge towards 0

testing

5.3	This will test whether an accurate Z path is still produced even if the Z value escapes towards infinity.	Constant 0 + 0i Z Value 0.7 + 0.7i		Fail	 Even though this is a fail the error message means that the Z value has escaped towards infinity. I think the reason behind this is that Lazarus can't handle the large gap between each point so it struggles to display them visually. The points I really care about how ever are the points inside the Julia set so it's not a huge loss as you only really see two points being plotted inside the radius of 2.
6.1	This will test whether the Julia set's colour scheme will change to the users selected one.	Red Colour Scheme	I'm expecting the Julia set to start off by displaying a red colour and as you go deeper into the image it starts to turn yellow and then purple	Pass	

testing

6.2	This will test whether the Julia set's colour scheme will change to the users selected one.	Blue Colour Scheme	I'm expecting the Julia set to start off by displaying a blue colour and as you go deeper into the image it starts to turn white		

Code Edits

Validation Code Edits for all complex number related boxes

Due to the original validation system being unreliable, I have created a new one which goes through every single character in the values entered into any of the boxes to see if anything other than a number, a decimal point and a negative symbol (for the first char) has been entered. If so depending on its position in the string and whether a negative or positive number has been entered the character is either assigned the value 0 or a decimal place. Here are the results of the new code with the actual code below them.

Test Number	Description	Data	Expected Outcome	Pass/Fail	Evidence
1.1	This will be used to test whether the program will accept the appropriate positive values of the real number text box.	0.25	0.250	Pass	**Real Number** 0.25 **Real Number** 0.250
1.2	This will be used to test whether the program will accept the appropriate Negative values of the real number text box.	-1.624	-1.624	Pass	**Real Number** -1.624 **Real Number** -1.624
1.3	This will be used to test whether the program will accept Numbers with more than 4 sig figs	-0.1722	-0.172	Pass	**Real Number** -0.1722 **Real Number** -0.172
1.4	This will be used to test whether the program will accept Strings	Cat	0.000	Pass	**Real Number** Cat **Real Number** 0.000

Test Number	Description	Data	Expected Outcome	Pass/Fail	Evidence
1.5	This will be used to test whether the program will accept Strings and Real data types combined	222c	2.200	Pass	**Real Number** 222c **Real Number** 2.200

All the tests above are relevant for the Imaginary Number, Last Real, Last Imaginary, Z Real and the Z Imaginary number as they use the same validation systems. The only difference For example, would be the validation for the Imaginary Number, would use ImaginaryNumber.text instead of RealNumber.Text. This is shown in the code below.

Code For Above Test

```
procedure TForm1.RealNumberExit(Sender: TObject);

//When the user exits the box the following statements happen

var

a: char;

  B: char;

  C: char;

  D: char;

  E: char;

  F: char;

  Length1: integer;

begin

ifRealNumber.Text = '' then //If Nothing is entered

RealNumber.Text := '0'; //The Real Number is set to 0

ifRealNumber.Text<> '' then //If the user has typed something in

begin

A :=RealNumber.Text[1];

   // Every character in the string value of the real number is obtained

B :=RealNumber.Text[2];
```

```
C :=RealNumber.Text[3];
D :=RealNumber.Text[4];
E :=RealNumber.Text[5];
F :=RealNumber.Text[6];
Length1 := Length(RealNumber.Text); //The length of the text is obtained
if (Ord(A) > 59) or (Ord(A) < 45) or ((A) = '.') or
    ((A) = ';') or ((A) = ':') or ((A) = '/') then
   //If the first char is not a number then
A := '0';//The first character is assigned the value 0
if (A <> '-') then //If the user inputs a number as the first char then
begin
B := '.';
   //the second char will be a decimal because I do not want the user to enter a value large than 9
if (Ord(C) > 59) or (Ord(C) < 45) or ((C) = '.') or ((C) = '-') or
    ((C) = ';') or ((C) = ':') or ((C) = '/') then
    //If the 3rd character is not a number then
begin
C := '0';//the third character is set to Zero
RealNumber.Text := A + B + C;
    //The String is updated by subbing the Zero in for the original C value
end;
if (Ord(D) > 59) or (Ord(D) < 45) or ((D) = '.') or ((D) = '-') or
    ((D) = ';') or ((D) = ':') or ((D) = '/') then
    //If the 4th character is not a number then
begin
D := '0'; //the 4th character is set to Zero
RealNumber.Text := A + B + C + D;
```

```
    //The String is updated by subbing the Zero in for the original 4th value
end;
if (Ord(E) > 59) or (Ord(E) < 45) or ((E) = '.') or ((E) = '-') or
    ((E) = ';') or ((E) = ':') or ((E) = '/') then
begin
E := '0';
RealNumber.Text := A + B + C + D + E;
end;
RealNumber.Text := A + B + C + D + E;//The Real Number never goes beyond 4 Sig Fif
end;
if (A = '-') then //If the user enters a negative number
begin
C := '.';
   //the third char would be the decimal value as i do not want numbers more than 2 or less than -2
if (Ord(B) > 59) or (Ord(B) < 45) or ((B) = '.') or ((B) = '-') or
    ((B) = ';') or ((B) = ':') or ((B) = '/') then
begin
B := '0';
RealNumber.Text := A + B + C;
end;
if (Ord(D) > 59) or (Ord(D) < 45) or ((D) = '.') or((D) = '-') or
    ((D) = ';') or ((D) = ':') or ((D) = '/') then
begin
D := '0';
RealNumber.Text := A + B + C + D;
end;
if (Ord(E) > 59) or (Ord(E) < 45) or ((E) = '.') or ((E) = '-') or
```

```
    ((E) = ';') or ((E) = ':') or ((E) = '/') then
begin
E := '0';
RealNumber.Text := A + B + C + D + E;
end;
if (Ord(F) > 59) or (Ord(F) < 45) or ((F) = '.') or ((F) = '-') or
    ((F) = ';') or ((F) = ':') or ((F) = '/') then
    //an extra character is used to make up for the one used for the - sign
begin
F := '0';
RealNumber.Text := A + B + C + D + E + F;
end;
RealNumber.Text := A + B + C + D + E + F;//RealNumber is updated.
end;
end;
end;


procedure TForm1.ImaginaryNumberExit(Sender: TObject);
//once you stop typing in the Constants imaginary number box the following statements occur
var
a: char;
 B: char;
 C: char;
 D: char;
 E: char;
 F: char;
```

```
 Length1: integer;
begin
ifImaginaryNumber.Text = '' then //If Nothing is entered
ImaginaryNumber.Text := '0'; //ImaginaryNumber is set to 0
ifImaginaryNumber.Text<> '' then //If the user has typed something in
begin
A :=ImaginaryNumber.Text[1];
  // Every character in the string value of the real number is obtained
B :=ImaginaryNumber.Text[2];
C :=ImaginaryNumber.Text[3];
D :=ImaginaryNumber.Text[4];
E :=ImaginaryNumber.Text[5];
F :=ImaginaryNumber.Text[6];
Length1 := Length(ImaginaryNumber.Text); //The length of the text is obtained
if (Ord(A) > 59) or (Ord(A) < 45) or ((A) = '.') or
     ((A) = ';') or ((A) = ':') or ((A) = '/') then
    //If the first char is not a number then
A := '0';//The first character is assigned the value 0
if (A <> '-') then //If the user inputs a number as the first char then
begin
B := '.';
    //the second char will be a decimal because I do not want the user to enter a value large than 9
if (Ord(C) > 59) or (Ord(C) < 45) or ((C) = '.') or ((C) = '-') or
     ((C) = ';') or ((C) = ':') or ((C) = '/') then
     //If the 3rd character is not a number then
begin
C := '0';//the third character is set to Zero
```

```
ImaginaryNumber.Text := A + B + C;
    //The String is updated by subbing the Zero in for the original C value
end;
if (Ord(D) > 59) or (Ord(D) < 45) or ((D) = '.') or ((D) = '-') or
    ((D) = ';') or ((D) = ':') or ((D) = '/') then
    //If the 4th character is not a number then
begin
D := '0'; //the 4th character is set to Zero
ImaginaryNumber.Text := A + B + C + D;
    //The String is updated by subbing the Zero in for the original 4th value
end;
if (Ord(E) > 59) or (Ord(E) < 45) or ((E) = '.') or ((E) = '-') or
    ((E) = ';') or ((E) = ':') or ((E) = '/') then
begin
E := '0';
ImaginaryNumber.Text := A + B + C + D + E;
end;
ImaginaryNumber.Text := A + B + C + D + E;//The ImaginaryNumber never goes beyond 4 Sig Fif
end;
if (A = '-') then //If the user enters a negative number
begin
C := '.';
   //the third char would be the decimal value as i do not want numbers more than 2 or less than -2
if (Ord(B) > 59) or (Ord(B) < 45) or ((B) = '.') or ((B) = '-') or
    ((B) = ';') or ((B) = ':') or ((B) = '/') then
begin
B := '0';
```

```
ImaginaryNumber.Text := A + B + C;
end;
if (Ord(D) > 59) or (Ord(D) < 45) or ((D) = '.') or((D) = '-') or
    ((D) = ';') or ((D) = ':') or ((D) = '/') then
begin
D := '0';
ImaginaryNumber.Text := A + B + C + D;
end;
if (Ord(E) > 59) or (Ord(E) < 45) or ((E) = '.') or ((E) = '-') or
    ((E) = ';') or ((E) = ':') or ((E) = '/') then
begin
E := '0';
ImaginaryNumber.Text := A + B + C + D + E;
end;
if (Ord(F) > 59) or (Ord(F) < 45) or ((F) = '.') or ((F) = '-') or
    ((F) = ';') or ((F) = ':') or ((F) = '/') then
    //an extra character is used to make up for the one used for the - sign
begin
F := '0';
ImaginaryNumber.Text := A + B + C + D + E + F;
end;
ImaginaryNumber.Text := A + B + C + D + E + F;//ImaginaryNumber is updated.
end;
end;
end;
procedure TForm1.ZRealNumberExit(Sender: TObject);
var
```

```
a: char;
   B: char;
   C: char;
   D: char;
   E: char;
   F: char;
   Length1: integer;
begin
ifZRealNumber.Text = '' then //If Nothing is entered
ZRealNumber.Text := '0'; //The Real Number is set to 0
ifZRealNumber.Text<> '' then //If the user has typed something in
begin
A :=ZRealNumber.Text[1];
   // Every character in the string value of the real number is obtained
B :=ZRealNumber.Text[2];
C :=ZRealNumber.Text[3];
D :=ZRealNumber.Text[4];
E :=ZRealNumber.Text[5];
F :=ZRealNumber.Text[6];
Length1 := Length(ZRealNumber.Text); //The length of the text is obtained
if (Ord(A) > 59) or (Ord(A) < 45) or ((A) = '.') or
      ((A) = ';') or ((A) = ':') or ((A) = '/') then
    //If the first char is not a number then
A := '0';//The first character is assigned the value 0
if (A <> '-') then //If the user inputs a number as the first char then
begin
B := '.';
```

```
    //the second char will be a decimal because I do not want the user to enter a value large than 9
if (Ord(C) > 59) or (Ord(C) < 45) or ((C) = '.') or ((C) = '-') or
    ((C) = ';') or ((C) = ':') or ((C) = '/') then
    //If the 3rd character is not a number then
begin
C := '0';//the third character is set to Zero
ZRealNumber.Text := A + B + C;
    //The String is updated by subbing the Zero in for the original C value
end;
if (Ord(D) > 59) or (Ord(D) < 45) or ((D) = '.') or ((D) = '-') or
    ((D) = ';') or ((D) = ':') or ((D) = '/') then
    //If the 4th character is not a number then
begin
D := '0'; //the 4th character is set to Zero
ZRealNumber.Text := A + B + C + D;
    //The String is updated by subbing the Zero in for the original 4th value
end;
if (Ord(E) > 59) or (Ord(E) < 45) or ((E) = '.') or ((E) = '-') or
    ((E) = ';') or ((E) = ':') or ((E) = '/') then
begin
E := '0';
ZRealNumber.Text := A + B + C + D + E;
end;
ZRealNumber.Text := A + B + C + D + E;//The Real Number never goes beyond 4 Sig Fif
end;
if (A = '-') then //If the user enters a negative number
begin
```

```
C := '.';
    //the third char would be the decimal value as i do not want numbers more than 2 or less than -2
if (Ord(B) > 59) or (Ord(B) < 45) or ((B) = '.') or ((B) = '-') or
     ((B) = ';') or ((B) = ':') or ((B) = '/') then
begin
B := '0';
ZRealNumber.Text := A + B + C;
end;
if (Ord(D) > 59) or (Ord(D) < 45) or ((D) = '.') or((D) = '-') or
     ((D) = ';') or ((D) = ':') or ((D) = '/') then
begin
D := '0';
ZRealNumber.Text := A + B + C + D;
end;
if (Ord(E) > 59) or (Ord(E) < 45) or ((E) = '.') or ((E) = '-') or
     ((E) = ';') or ((E) = ':') or ((E) = '/') then
begin
E := '0';
ZRealNumber.Text := A + B + C + D + E;
end;
if (Ord(F) > 59) or (Ord(F) < 45) or ((F) = '.') or ((F) = '-') or
     ((F) = ';') or ((F) = ':') or ((F) = '/') then
     //an extra character is used to make up for the one used for the - sign
begin
F := '0';
ZRealNumber.Text := A + B + C + D + E + F;
end;
```

```
ZRealNumber.Text := A + B + C + D + E + F;//RealNumber is updated.
end;
end;
end;

procedure TForm1.ZImaginaryNumberExit(Sender: TObject);
var
a: char;
  B: char;
  C: char;
  D: char;
  E: char;
  F: char;
  Length1: integer;
begin
ifZImaginaryNumber.Text = '' then //If Nothing is entered
ZImaginaryNumber.Text := '0'; //ImaginaryNumber is set to 0
ifZImaginaryNumber.Text<> '' then //If the user has typed something in
begin
A :=ZImaginaryNumber.Text[1];
   // Every character in the string value of the real number is obtained
B :=ZImaginaryNumber.Text[2];
C :=ZImaginaryNumber.Text[3];
D :=ZImaginaryNumber.Text[4];
E :=ZImaginaryNumber.Text[5];
F :=ZImaginaryNumber.Text[6];
Length1 := Length(ZImaginaryNumber.Text); //The length of the text is obtained
```

```
if (Ord(A) > 59) or (Ord(A) < 45) or ((A) = '.') or
      ((A) = ';') or ((A) = ':') or ((A) = '/') then
     //If the first char is not a number then
A := '0';//The first character is assigned the value 0
if (A <> '-') then //If the user inputs a number as the first char then
begin
B := '.';
     //the second char will be a decimal because I do not want the user to enter a value large than 9
if (Ord(C) > 59) or (Ord(C) < 45) or ((C) = '.') or ((C) = '-') or
      ((C) = ';') or ((C) = ':') or ((C) = '/') then
      //If the 3rd character is not a number then
begin
C := '0';//the third character is set to Zero
ZImaginaryNumber.Text := A + B + C;
      //The String is updated by subbing the Zero in for the original C value
end;
if (Ord(D) > 59) or (Ord(D) < 45) or ((D) = '.') or ((D) = '-') or
      ((D) = ';') or ((D) = ':') or ((D) = '/') then
      //If the 4th character is not a number then
begin
D := '0'; //the 4th character is set to Zero
ZImaginaryNumber.Text := A + B + C + D;
      //The String is updated by subbing the Zero in for the original 4th value
end;
if (Ord(E) > 59) or (Ord(E) < 45) or ((E) = '.') or ((E) = '-') or
      ((E) = ';') or ((E) = ':') or ((E) = '/') then
begin
```

```
E := '0';
ZImaginaryNumber.Text := A + B + C + D + E;
end;
ZImaginaryNumber.Text := A + B + C + D + E;//The ImaginaryNumber never goes beyond 4 Sig Fif
end;
if (A = '-') then //If the user enters a negative number
begin
C := '.';
    //the third char would be the decimal value as i do not want numbers more than 2 or less than -
2
if (Ord(B) > 59) or (Ord(B) < 45) or ((B) = '.') or ((B) = '-') or
    ((B) = ';') or ((B) = ':') or ((B) = '/') then
begin
B := '0';
ZImaginaryNumber.Text := A + B + C;
end;
if (Ord(D) > 59) or (Ord(D) < 45) or ((D) = '.') or((D) = '-') or
    ((D) = ';') or ((D) = ':') or ((D) = '/') then
begin
D := '0';
ZImaginaryNumber.Text := A + B + C + D;
end;
if (Ord(E) > 59) or (Ord(E) < 45) or ((E) = '.') or ((E) = '-') or
    ((E) = ';') or ((E) = ':') or ((E) = '/') then
begin
E := '0';
ZImaginaryNumber.Text := A + B + C + D + E;
end;
```

```
if (Ord(F) > 59) or (Ord(F) < 45) or ((F) = '.') or ((F) = '-') or
        ((F) = ';') or ((F) = ':') or ((F) = '/') then
        //an extra character is used to make up for the one used for the - sign
begin
F := '0';
ZImaginaryNumber.Text := A + B + C + D + E + F;
end;
ZImaginaryNumber.Text := A + B + C + D + E + F;//ImaginaryNumber is updated.
end;
end;
end;
```

Validation for the speed of the animation

This very similar to the previous methods of validation except no negative values are allowed so if a negative value is entered the rate is set to 0.01.

Description	Data	Expected Value	Pass/Fail	Evidence
This will test a value for the rate which I want to be accepted	0.023	0.023	Pass	**Rate** 0.023 **Rate** 0.023
This will test a value for the rate which is larger than the allowed value	0.51	0.01	Pass	**Rate** 0.51 **Rate** 0.01
This will test a value for the rate which is a – value.	-0.1	0.010	Pass	**Rate** -0.1 **Rate** 0.010
This will test a value for the rate which is a collection of letters which should not be accepted	Cat	0.01	Pass	**Rate** Cat **Rate** 0.01
This will test a value for the rate which as contains both numbers and different types of Chars	2Cat	0.01	Pass	**Rate** 2Cat **Rate** 0.01

Code For Rate Validation

```
procedure TForm1.RateBoxExit(Sender: TObject);
var
a: char;
  B: char;
  C: char;
  D: char;
  E: char;
  F: char;
  Length1: integer;
begin
ifRateBox.Text<> '' then
begin
A :=RateBox.Text[1];
B :=RateBox.Text[2];
C :=RateBox.Text[3];
D :=RateBox.Text[4];
E :=RateBox.Text[5];
F :=RateBox.Text[6];
Length1 := Length(RateBox.Text);
if (Ord(A) > 59) or (Ord(A) < 45) or ((A) = '.') or ((A) = '-') or
    //If any of the characters in the string are not numbers then they are set to 0
```

```
    ((A) = ';') or ((A) = ':') or ((A) = '/') then
A := '0';
if (A <> '-') then
begin
B := '.'; //B is always the decimal point
if (Ord(C) > 59) or (Ord(C) < 45) or ((C) = '.') or ((C) = '-') or
      ((C) = ';') or ((C) = ':') or ((C) = '/') then
begin
C := '0';
RateBox.Text := A + B + C;
end;
if (Ord(D) > 59) or (Ord(D) < 45) or ((D) = '.') or ((D) = '-') or
      ((D) = ';') or ((D) = ':') or ((D) = '/') then
begin
D := '0';
RateBox.Text := A + B + C + D;
end;
if (Ord(E) > 59) or (Ord(E) < 45) or ((E) = '.') or ((E) = '-') or
      ((E) = ';') or ((E) = ':') or ((E) = '/') then
begin
E := '0';
RateBox.Text := A + B + C + D + E;
end;
RateBox.Text := A + B + C + D + E;
end;
end;
```

```
if (RateBox.Text = '') or (StrToFloat(RateBox.Text) = 0) or
   (StrToFloat(RateBox.Text) > 0.5) then
   //If no value is entered or the rate is zero or is larger than 0.5 then
RateBox.Text := '0.01'; //The rate is assigned the value 0.01 end;
```

Maximum Number of Iterations Validation

The validation for the maximum number of iterations text box is almost identical to the validation used for the Rate of the animation. The only difference being the user can only type in a value with 3 digits with no decimal places.

Description	Data	Expected Value	Pass/Fail	Evidence
This will test a value for the max iterations which I want to be accepted	200	200	Pass	**Maximun Iterations** 200 **Maximun Iterations** 200
This will test a value for the max iterations which is larger than the allowed value	1000	100	Pass	**Maximun Iterations** 1000 **Maximun Iterations** 100
This will test a value for the max iterations which is a collection of letters which should not be accepted	Cat	100	Pass	**Maximun Iterations** Cat **Maximun Iterations** 100
This will test a value for the max iterations which as contains both numbers and different types of Chars	2Cat	200	Pass	**Maximun Iterations** 2Cat **Maximun Iterations** 200

```
procedure TForm1.IterationsExit(Sender: TObject);
var
a: char;
  B: char;
  C: char;
Length1: integer;
begin
ifRateBox.Text<> '' then
begin
A :=Iterations.Text[1];
B :=Iterations.Text[2];
C :=Iterations.Text[3];


Length1 := Length(Iterations.Text);
if (Ord(A) > 59) or (Ord(A) < 45) or ((A) = '.') or ((A) = '-') or
    ((A) = ';') or ((A) = ':') or ((A) = '/') then
A := '0';
if (A <> '-') then
begin
if (Ord(B) > 59) or (Ord(B) < 45) or ((B) = '.') or ((B) = '-') or
     ((B) = ';') or ((B) = ':') or ((B) = '/') then
begin
B := '0';
Iterations.Text := A + B;
end;
```

```
if (Ord(C) > 59) or (Ord(C) < 45) or ((C) = '.') or ((C) = '-') or
    ((C) = ';') or ((C) = ':') or ((C) = '/') then
begin
C := '0';
Iterations.Text := A + B + C;
end;
Iterations.Text := A + B + C;
end;
end;
if (Iterations.Text = '') or (StrToFloat(Iterations.Text) = 0) then
Iterations.Text := '100';
end;
```

Evaluation

Objective Completeness and Improvements/Potential Extensions

Objective Number	Objective	Completeness	Improvements/Potential Extensions
1.	To allow the user to change the colour of the generated Julia set.	The final program I have created does give the user the ability to change the colour of the Julia set by selecting one of 3 colour schemes. Overall this objective has **been complete.**	If I was to continue the project in the future I would definitely increase the number of colour schemes available to the user in order to increase interactivity.
2.	To allow the user to enter a constant of their choosing which in turn generates a clear and accurate Julia set.	The final program does give the user the ability to enter the complex number of their choice and in turn will generate the Julia set. Overall this objective has **been complete.**	I do not think there are any further improvements I could make to this objective.
3.	The program must provide a very reliable validation system.	Overall this objective has **been complete.** The Validation system works very well. If anything other than a number is entered into the program then that character is assigned the value 0 so value can still get accepted.	I don't think this objective requires any further Improvements.
4.	The program must allow the user to zoom into the generated Julia set.	Overall this objective has **been complete.**	I don't think this objective requires any further Improvements.
5.	The program must allow the user to generate a z path using a user defined z value.	Overall the objective has been **partly complete**. The user is able to display the Z path of a Z coordinate which is inside the Julia set. On the other hand, the program is unable to show points that escape towards infinity due to Lazarus not being able to handle the sheer spread of the Z points as the path travels towards infinity.	Aside from finding a way to allow the user to generate the path of a point escaping toward infinity. I would also like to investigate into producing Z paths for different functions and seeing how these Z paths behave.
6.	The Program must display each of the first few points beside the z path.	Overall this objective has **been complete.**	I don't think this objective requires any further Improvements.

Objective Number	**Objective**	**Completeness**	**Improvements/Potential Extensions**
7.	The program must be able to produce an animation which shows the way a Julia set changes with a changing constant.	Overall this objective has **been complete.** The program allows the user to create an animation showing a changing Julia set as well as allowing the user to change the speed of the animation.	There are no improvements that I feel need to be made forthis objective.
8.	The user must be able to change the function of the Julia set generated.	This proved to present a bit of a challenge when planning as using complex numbers in functions makes the calculation more complicated and a lot less predictable. As a substitute, I have provided the user with 3 already programmed functions so the user can see the difference when the function has an increasing value of Z. Overall this objective has **been complete.**	If I had enough time I would of really liked to look into allowing the user to actually type in a desired function and in turn display the Julia set of that function.

Feedback

Here is the feedback I have received about my program from my end user

Mark as unrea

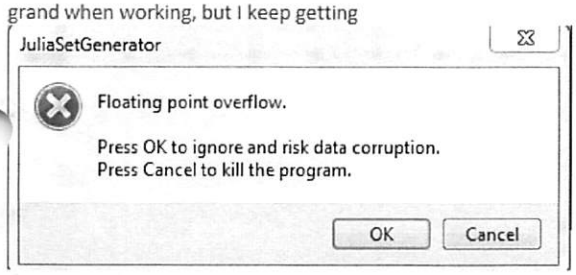
To:

Cc:

- You replied on 20/03/2017 19:43.

Dear

Many thanks for the program. Sorry I have been slow to respond. The pictures are very good. The z paths are the things I particularly wanted. These are grand when working, but I keep getting



Which is a shame because these are good.

The program is to display the finished Julia set rather than to explain how they are created, but the display is very good, showing how the sets change as c changes.

Thank you for this.

Overall the feedback I have receive appears to be fairly positive. ı appears to be very happy with generation of the Julia sets along with the additions of being able to change the colour scheme and the presence of an option to produce an animation. She also seemed to like the features which display the z path of the user defined z values which are inside the Julia set. She however would have liked the program to also display the paths of z values that escape towards infinity which is understandable as the user would be able to compare the rapid change in the paths between a point in the set, and a point outside. She also would have liked it to be more of a tool which gives the user more information about how the set is generated. If I was to have more time with the project, to address her feedback on the program maybe being more descriptive on how the Julia sets are formed, I would program a feature which displays the colour of the user defined Z point in the Julia set. I will find the colour by finding the number of iterations the point goes through before escaping the radius of 2. Once it has escaped I will use that and the maximum number of iterations in the calculation - ColourID = the number of iterations the function has gone through / Maximum number of iterations. Once the ID has been obtained I will use a new colour map series to display the colour so they can compare it to the generated Julia set.

Conclusion

As a whole I would consider my program a success. The majority of the features work without errors and produce very clear and accurate results and the validation seems to filter out anything that would cause the program to crash. The only thing I think really needs to be polished is the Z path generator, as it is not able to produce paths for values of Z which escape towards infinity.

In the future, if I was to continue with this project I would definitely be interested in adding a feature which allows the user to type in their own functions in order to generate their own Julia sets and also attempt to make the program more descriptive by allowing the user to view the colour assigned to the user defined Z value.

Bibliography

Url	Uses
http://wiki.lazarus.freepascal.org/TAChart_Tutorial:_ColorMapSeries,_Zooming	I used this to learn how to use a feature in Lazarus called TColorMapSeries which is used to generate the Julia sets.
http://wiki.freepascal.org/TAChart_Tutorial:_Getting_started	I used this to get a basic idea of how to use line series', so I was able to code and produce the Z paths.
https://www.youtube.com/watch?v=2AZYZ-L8m9Q	I used this You Tube video to help me understand the concepts of a Julia set such as understanding how a point is assigned a certain colour and how some points escape towards infinity and some converge on a point or spiral in an infinite loop.
https://www.youtube.com/watch?v=HAyDhBZ_oTA	I used this video overall to give me an idea on how to use TTimer which would in turn be used for coding the animation.