JAVA FOR THE STUDY OF EVOLUTION

VOLUME XV

THE HAND III

José del Carmen Rodríguez Santamaría
The EvolJava Community

Contents

Preface

The series *Java for the Study of Evolution* is directed to scientists that want to manage a serious but not excessively expensive tool to study evolution by direct experimentation under perfectly controlled conditions. These requirements cannot be met in nature but only in simulations and mathematical models. In consequence, the series has three main purposes:

1. To endow the community of **researchers in biology and evolution** with *high level programming*, enabling an accurate study of models and simulations of the most diverse nature.

2. To clearly show how this tool is used to study the fundamental questions of evolution.

3. To suggest that the study of Java could be *very fruitful* for **undergraduates** in biological sciences even more than calculus alone.

This is the 15th volume: We dedicate this volume to continue with our study of the hand and to prepare the terrain for enabling the falsification of the Evolutionary Theory.

Introduction

We dedicate this volume to continue with our study of the hand and to get more practice in the falsification of the Evolutionary Theory. This is work in progress.

To begin with, we propose in Chapter I a solution to the problem of the incompatibility of meshes that are endowed with Xforms and controls because these are inserted into the 3D graphic of the mesh. Various proposals are researched: to use a borderpane, two scenes, two stages, and two screens. We find that working with two stages solves the problem.

We use this solution to improve our presentation of the hand in Chapter II, in which we make better use of PCA (principal component analysis).

To explore the world of tools for the management of meshes we review some JavaFX material in Chapter III. Tools are researched in Chapter IV. Some partial positives results were achieved.

Thinking of big projects we retake the study of sensitive meshes in Chapter V. In this regard, JavaFX promises too wonderful methods but as yet (2016) it falls short of correct implementations. So, we are forced to work with the nails to get content with the simulation of the evolution of the simplest hand, which is a tail. We end by showing what we mean by random mutation in regard with the evolutionary explanation of the chimp-man transition of the hand.

Programs are zipped in the file `eJVol15P.zip`. This is a NetBeans project to be imported. It needs a file with the PCA mathematics. We use *JAMA : A Java Matrix Package*. The corresponding *.jar* file can be downloaded from (JAMA [1], 2015). To add it: right click over the name of the project under the projects' tab. Choose properties $->$ Libraries $->$ compile $->$ Add jar/Folder. The files with bones that was downloaded for Vol XIII also are needed and can be posited at an appropriate folder, anyone.

Bogotá, Colombia, *José Rodríguez*
May 2016

Chapter 1

JavaFX: compatibility of Xforms and controls

`Meshes and controls working together`

1 ***Introduction and purpose.*** *Meshes that are endowed with Xforms were found in the previous volume to be incompatible with controls because these are inserted into the 3D graphic of the mesh. In order to solve this trouble we research here various proposals: a borderpane, two scenes, two stages, and two screens. A borderpane is a pane whose borders are wide and so make room for diverse objects, say controls. The mesh would be located at the center of the pane. In regard with duplications, say of scenes, the first will be used to display the mesh while the other will host the diverse controls. Flip flopping from one scene to the other is made through the keyboard. If nothing works, we will extend the power of control of the keyboard whose compatibility with Xforms already has been asserted.*

1.1 Borderpane

A **borderpane** is a pane whose borders are wide and so make room for diverse objects, say controls. One has complete power to determine the border to which desired object will be added. Will this type of pane solve the posed problem of compatibility? Let us see.

2 ***Example of encoding of a JavaFX borderpane:***

You can see the code in program Program P2 BorderPaneExample.

3 ***Exercise.*** *Run the previous program and play with the code.*

4 ***Testing the compatibility of borderpanes with 3D mesh + Xforms.***

To test the compatibility of borderpanes with Xforms, we make an attempt to add a tetrahedron to the center of the pane to see what happens. The TetraheronTool class, that outputs a tetrahedron, was added to the package tools. The code is in Program P4 BorderPaneAndXform.

5 ***Exercise.*** *Run the previous program and play with the code. Play enough with the code to reaffirm else reject the belief of the Author that borderpanes and Xforms are incompatible.*

1.2 Two scenes

The JavaFX Scene class is the container for all content that you want to display. Since we have been unable of using a single scene to make Xforms and controls compatible, what about using two scenes?

6 ***Example of an encoding for two scenes: Program P6 TwoScenes.***

7 ***Exercise.*** *Run the previous program and play with the code.*

8 ***Exercise: Two scenes with hybrid flip-flopping.*** *Our plan is to dedicate one scene for the Xform and the other for controls. So, the first scene cannot have controls because of incompatibility problems. Thus, a question arises: how are we going to switch from one scene to the other? We can use the keyboard. So, study the method* `handleKeyboard(Scene scene)` *of the class* `Axes` *of the package* `Tools` *of this project and implement the posed task in a new program. You can see our answer in* `Program P8 TwoScenes2`*.*

9 ***Exercise.*** *Add to the previous code the possibility to use* `Ctrl + S` *to go forth and backwards from one scene into the other. Answer in* `Program P9 TwoScenes3`*.*

10 ***Exercise.*** *Use your recently developed program to test the compatibility of your new scheme with Xforms. Answer in* `Program P10 TwoScenes4`*.*

1.3 Two stages

Windows are called stages in JavaFX. So, we plan to use one stage to display the mesh and the other for controls. Our aspiration is to arrange the two windows one on the side of the other.

11 ***Exercise.*** *Research on Internet how to deal with two stages, copy, adapt and run the corresponding code. Answer in* `Program P11 TwoStages.`

12 ***Exercise.*** *Research on Internet how to bind multiple stages in JavaFX: a control in one stage must modify the second. Copy, adapt and run the corresponding code. Answer in* `Program P12 TwoStages2.`

13 ***A mesh viewer is built by the next code. The application shows two stages adding an Xform with a mesh to the first and a control to the second.*** **`Program P13 TwoStages3.`**

14 ***Exercise.*** *Run the previous program and play with the code.*

1.4 Dialogues

To communicate something to the User, we use Dialogues. For instance, one can make a translation slider into a zooming tool by means of a rotation. So, let us learn how to learn how to produce dialogues in JavaFX.

15 ***Exercise.*** *Research on Internet how to use a dialog to pass some information to the User in JavaFX but without using deprecated code. Answer in* `Program P15 DialoguesFX`.

1.5 Two stages with dialogues

Let us use JavaFX dialogues to improve our application with two stages.

16 ***Exercise.*** *Have your previous application to pop up a message with instructions to orient the graphic in such a way that a translation becomes a zooming out operation. Answer in* `Program P16 TwoStages4.`

1.6 Two screens

If meshes turn to be extremely complex, one possibility that might help is to use two or more screens.

17 ***Challenge.*** *Research on Internet how to deal with two or more screens, copy, adapt and run the corresponding code.*

1.7 Conclusion

The Author suffered a lot to develop the programs of this chapter. No surprise, suffering is the rule. Nevertheless, a serious and recurrent difficulty that was somehow stronger here was originated by programs that compile but produce incomplete and/or imperfect output. This is due in part to the `general` theory of complexity: syntax is not enough to judge the output of a program and the only way to know the behavior of a program is to run it under all possible initial conditions. Besides this, troubles are also due to the fact that Java is a big project and so solutions are targeted to solve problems under specified conditions and protocols but it happens that to make the solution stable against code variants, even tiny, might demand centuries of work. So, this is not done. The net result is the arising of the guru phenomenon: there are tight tricks that function and that are known or discovered by some few people. Some of them are gentle enough to teach the rest of us in, say, forums. Thanks a lot to them all.

We have said that we have suffered a lot. What does it mean? It means that many code variations do not compile, that some few do but somehow malfunction, say, an Xform gets deaf to the mouse. Or, in own words: there is no code development without a lot of bugs whose correction generate more bugs. At the extreme we have programs that admit no variation at all: you know the trick else you get nothing. Our experience in this chapter seems to fit this situation.

Now, how is our personal experience with JavaFX 8 related to the Evolutionary Theory? The link is that the genome is software: it conveys verbal instructions (that need a verbal code of interpretation) to built organisms from extant living raw material. The extreme perfection that we enjoy in own bodies clearly retro-predict that the genome was not easy to concoct. So, a question arises: Where does it come from?

One can explain everything by mere randomness. Nevertheless, the associated probabilities in regard with the genome are so small that at present time no one invokes randomness as the cause of life. Instead, evolution has proven to be a wonderful tool to solve many kind of problems and so people use to believe that evolution is the final non questionable explanation of our existence. Wrong: evolution is real and natural so, it must obey the laws of nature. In regard with the development of software the most mandatory one is the law of evolution of perfection. It is so elementary that it looks like a tautology: there is no evolution of perfection without evolution. What does this mean?

It means that many problems of perfection can be solved by evolution but through many local optima. Interesting problems leave behind too many local optima that are separated by huge changes the larger the nearer to perfection. Local optima correspond with imperfect, malformed and/or malfunctional organisms that

are in general damaged or worsened if one attempts to modify them. This is not seen in the fossil record nor in present populations. So, the Evolutionary Theory is obviously false.

It might be possible to argue that the man is more complex than the chimp and so the perfection of the man is more cogent and compelling than the perfection of the chimp. If we add evolution to this assertion, we immediately get a direct proof of the law of the evolution of perfection. It is here where we enter into action: it is the duty of the sole EvolJava Community to show that the chimp-man gap contains no evolution at all. This task is so daunting that all we do here might look as kicks of a baby. That is right, but we also have observed the baby keeps growing.

Chapter 2

Improving hand movements

`More natural movements of the hand`

18 ***Introduction and purpose.***

We admire by instinct the high quality of life. But when we try to replicate it with Java we always find the same result: half cooked products and frustration. One vivid experience that illustrates this situation was encountered in the previous volume when trying to mimic cylindrical mutations of the hand. We could accommodate mutant bones to recover the human pattern but when we commanded the hand to close and open, the structure was severely distorted. Many trials were done to correct this but nothing interesting was gotten. This failure produces in us a reinforcement of the conviction that the Evolutionary Theory is obviously false because too many imperfections along the fossil record and in our bodies are expected but not found. Now, our failure produces in this case a challenge because we know that we have not used the power of mathematics to the full: we still have to use PCA (Principal Component Analysis) to pursue quality. Our purpose is to work in the solution of this problem. Additionally, we plan to add the capability of controlling hand movements by means of a set of sliders.

2.1 Will PCA help?

PCA produces a group of axes that represent the overall dominant tendencies of the given 3D object. So, it is a must to test how powerful it is to resolve the problem of the definition of the correct axes of rotation of the diverse phalanges of the hand that movement could look natural.

19 ***Exercise.*** *Test PCA axes as axes of rotations of phalanges to modify program N48 of previous volume, XIV. To try it over phalanges of the index, finger 2, is enough. Our answer can be found in* `Program P19 CylindricalMutations5`*. Our results were disappointing.*

In hindsight and taking into account that every program of optimization can be restated in evolutionary terms, our failures have something to teach us:

20 ***Comment.*** *One more trial, one more failure. This is evolution when it is directed by a human mind to solve most problems. Or equivalently, a serious and wise effort must be made in general to succeed. Now, if that happens with us, how much more must happen with Darwinian Evolution. That is why the Evolutionary Theory is obviously false: tracks of recurrent failures must appear in the fossil record and in extant populations but none is found.*

21 ***Exercise.*** *Our mutant hands consists of bones of diverse thickness and so, thick ones may overlap with others. The result looks very bad. To remedy this situation make the metacarpals to ray away without overlapping nor great voids. The code can be seen in* `program P21 CylindricalMutations6`*.*

22 ***Exercise.*** *Bind movements of fingers to sliders just as it was done with the hand made out of boxes in Program M29 HandOnlyBoxess12 of Vol XIII. Use one stage to drw the hand and another to posit controls. Our answer can be seen in* `program P22 CylindricalMutations7`*. This was an easy exercise: in regard with the posed task, the developed software was found to be highly evolvable.*

2.2 Conclusion

With great hope we tried to get profit of PCA to make the movement of the hand more natural. But we failed: rotations of bones are defined by local geometry not by the overall pattern that is correctly captured by PCA.

Chapter 3

JavaFX: more UI controls

`ComboBoxes, trees, ...`

23 ***Introduction and purpose.***

We have been unable of producing a satisfactory solution to the problem of correctly draw mutant hands. A good idea in a case like this is to devise a tool to interactively manage the Xform properties of meshes in 3D space. To that aim, it would be good to know more about the options that are provided by JavaFX to make suitable controls.

3.1 ComboBoxes

We need in first place to know how to choose an item from a list in JavaFX 8, to choose a specific bone that must be relocated in 3D space. When the list is linear, this is done by a **combobox**:

24 ***JavaFX: combobox.*** *The JavaFX* `Program P24 ComboBoxSample` *allows to choose an item from a list.*

25 ***Exercise.*** *Run the previous program and play with the code.*

We already know how to choose an item from a linear list. But the structure of the hand is correctly described by a tree. So, we need not a simple list but one that describes a tree.

3.2 Trees with checkboxes and radioButtons

We will see how to enrich trees with various controls.

26 ***Exercise.*** *Search on Internet or in our previous work for the JavaFX code to encode and display a* ***tree****. You can see our answer in* `Program P26 TreeViewSample3.`

JavaFX allows embellishment of high functional value. Say, for the case of trees, we can add **checkboxes** to each item. These controls are used to define the value of a boolean property, say, to belong in a set.

27 ***Tree with checkboxes.*** *The following code presents a tree that is enriched with checkboxes. See* `Program P27 TreeCheckBox.`

28 ***Exercise.*** *Run the previous program and play with the code.*

We can pass from trees of strings to more complex classes as the following code shows:

29 ***Tree with checkboxes that accept instances of a complex class instead of strings. This is a marvelous example of evolvability: one can pass from simple to complex with the minimum effort. See*** **`Program P29 TreeCheckboxesPerson`****. Not shown: JavaFX also allows to display multiple information in the form of tables with the help of TreeTableView controls.***

30 ***Exercise.*** *Run the previous program and play with the code.*

31 ***Exercise.*** *In regard with a tree with checkboxes, research how to endow it with listeners of checkboxes else of treeItems. Our answer can be seen in* `Program P31 TreeWithCheckBoxesListeners.`

32 ***Exercise.*** *Research how to insert into a tree some radioButtons instead of checkboxes. Our answer is in* `Program P32 TreeWithRadioButtons.`

Let us see how to add intelligence to our controls. Intelligence means that the control offers its services just when these are needed.

3.3 Intelligent tools: tooltips and contextMenus

JavaFX allows with the help of the **ToolTip** class to pop up a specific message if one drags the mouse over a given item. This is gentle context dependent assistance. If it is well implemented, it looks as intelligent assistance.

33 ***The code for a toolTip can be seen in*** **`Program P33 ToolTip`**.

34 ***Exercise.*** *Run the previous program and play with the code.*

One might prefer to implement solicited context dependent help. This is done with the help of **contextMenus**.

35 ***Exercise.*** *Research on Internet for a code to endow a tree with contextMenus. Our answer can be seen in* `Program P35 TreeWithContextMenus.`

36 ***Exercise.*** *Compose a code to display a tree that represents a hand with the possibility to select a bone for some future task, say, to redefine its Xform parameters. Our answer is in* `Program P36 HandAsTree`.

3.4 Conclusion

Java and JavaFX allow for high sophistication of products. In general we are used to ignore that possibility but sometimes one perceives that functionality is also added as the posed examples with trees: these can be endowed with checkboxes and/or radioButtons and of course with specific listeners. Sophistication might demand huge amounts of resources but JavaFX seems to pay the price without passing a high cost to the User. This happens because JavaFX appeared late in the development of graphic technology and so it made profit on the achievements of many companies with a long history of hard work. Thanks them all.

On the other hand, my hand is marvelous. It has the structure of a tree. It is a marvel that is more clever than complex. Therefore, the Evolutionary Theory predicts a huge process of optimization to reach the actual pattern that you and me so much enjoy. So, we ask: where in the fossil record are the infinitely many tree structures that witness to the evolution of the hand? This question is too general so, we have been exploring a family of mutant hands that is given by cylindrical mutations. Our purpose is to show that this type of mutants hands are so numerous and varied that it is impossible to ignore that this family is not abundantly represented in the fossil record. The challenge is to convert this complaint into a Java study to demonstrate that the Evolutionary Theory is obviously false because that

theory predicts a fierce evolutionary process of the hand amids arboreal forms to finally output a single winner such as we see it today in our hands. But the fossil record lacks any evidence of such a process.

And, what about not arboreal structures say, a hand whose fingers form some cycles?

Chapter 4

A tool for Xforms

Using the mouse to define parameters

37 ***Objective.*** *In spite of his effort, the Author has been unable to devise a program to perfectly unfold the movements of mutant hands. Indeed, that has been too much work without chasing the prey. A change of strategy might function better. In that regard, our purpose for this chapter is to design an interactive tool for editing of Xform parameters of individual bones and of meshes in general.*

4.1 Tools and complexity

The existence of tools enriches the theory of complexity.

38 ***Tools, oracles and geometry.***

It is unthinkable to live without so many tools that we ordinarily use. Just think of hex keys. Tools function as hints to solve optimization problems: we use a spoon to intake liquids with maximal efficiency. Or, a calculator executes mathematical operations at a very low cost. The mathematical abstraction of a calculator is an oracle that functions instantaneously at no cost at all. Oracles diminish the complexity of a problem: programming gurus are oracles for the very difficult task of program development.

Given that the human brain serves as a fabric of oracles, we are forced to pose the following question: can we build a finite number of oracles to get rid of complexity in order to create a magical world in which dreams come true on time?

One is led by instinct to answer negatively to this question but what if we change the question to this: can we build a finite number of oracles to get rid of

most complexity in order to create a magical world in which many dreams come true almost on time?

One is led by instinct to believe that this is possible. A geometric picture may help us to understand why this is possible: Let us imagine that we have to find the maximal value of a function on one variable. There is no problem for a parabola. But if the number of peaks grows, this may become more tedious. And when the density of local peaks grows as an exponential function of $1/length$, then we get into troubles. Nevertheless one can imagine the following trick (known in the engineering literature as the extradimensional principle): immerse the graphic of your function, which is 2D, into 3D space. Next manage to extend your 2D graphic to a mountain with just one peak in such a way that all local maxima of your 2D graphic converge along an increasing tendency to the great peak. When can this be done? For a mild silhouette surely this is possible: this is the world of tools, in which one adds extra things, that need extra dimensions for their description, to help in the solution of problems. For extremely wild silhouettes this is not possible. But what does remain in the world of the almost? After all, this is the real world of very happy persons and of all dreamers that preach heavens on earth.....

4.2 Devising the tool

Let us attempt the development of a tool to control Xform parameters. It seems to be a challenging task. So, we will need various steps. The very idea is presented in the first, while in following versions some improvements will be added.

39 ***The tool, step 1, the very idea.*** *We devise a tool to interactively manage Xform properties of meshes in 3D space. We use trees to input the hand structure, listeners to select specific group of phalanges, sliders to modify the Xform parameters of that group: position after parallel displacement together with the axes and pivots for rotation. You can test our first version by running program* `P39 CylindricalMutations8`*.*

40 ***Exercise.*** *Run the cited program* `P39 CylindricalMutations8` *and play with the code. Play enough to detect aspects whose improvement is realizable.*

41 ***Exercise: The tool, step 2, fleshing.*** *Make the idea presented in program* `P39 CylindricalMutations8` *into something useful. You can compare your solution with our answer in program* `P41 CylindricalMutations9`*. Everything works fine with the exception of the controls for the change of the direction of the axis of rotation. Implemented operation is just symbolic.*

42 ***Exercise: The tool, step 3, more functionality.*** *Various untied ends have been left in the previous program* `P41 CylindricalMutations9`*. Continue your quest for full functionality. Compare your solution with ours in program P42 CylindricalMutationss10. The controls for the change of the direction of the axis of rotation has been implemented with rotations but the result is a fiasco.*

43 ***More unsuccessful tinkering can be seen in* `program P43 CylindricalMutationss11`*.***

4.3 Challenge

We failed in our attempt to endow our tool with the possibility to graphically define the direction of the axis of rotation of a mesh. Therefore, we have a task:

44 ***Open problem.*** *Fix the bug of the previous program that deceptively defines the direction of axis of rotations. Most possible, that cannot be done without an exhaustive lab on binding of complex Java objects, say of points3D to a set of xyz-sliders.*

4.4 Research

The design of tools for the management of meshes is very important and consequently has been widely developed.

45 ***Research on Internet: tools for the management of meshes. Orientation:***
Tools for the management of meshes are a whole unbounded world today that is an important asset of the industries of CAD, 3D games and of game engines (tools for game development). A good application must be free, open source, and famous. A light and powerful one is `Gmesh`*. Blender could be an interesting free-hands tool that is written in C.* `JMonkey` *is written in Java and is moreover very mature.* `Unity3D` *is proprietary but is nevertheless very famous and very popular. Some engines are proprietary but free for educational purposes such as the* `Unreal Development Kit`*. Other engines are addressed to the web. Example:* `Babylon.js` *that is a modern, complete JavaScript framework for building 3D games with HTML5, WebGL and Web Audio (Javascript is the most popular web language and its syntax is very similar to that of Java so, it will be easy to learn). And, what about* `JavaFX`*? It is free, open source, famous, but overall it is a* `customizable game engine` *on its own as one can verify by prompting Google with the phrase:* `game engine in javafx`*. Although one can include a JavaFX application into any web page, the truth is that this is not usual*

and, in spite promises, the situation will remain so for a long time. In any case, the better you know JavaFX, the easier it will be to learn any other 3D framework.

It would be good to keep in mind that the web can be used in order that other people could enjoy our work while JavaFX on a desktop is addressed for those special and rare persons that want just to get the fun and challenge of becoming creators. Now, Javascript have both at an excel level but nevertheless it does not opaque Java. So, what does Java has that it is so strong and popular? Experts mention many, many things (Spann, 2015). This is evidently a very complex theme but lay people must have concepts clear on how to organize the most important point of the discussion: the fundamental core of programming is functionality + evolution. How is that achieved? To be true, this is not achieved anyhow, rather with a great effort and training one can try to develop functional evolvable code. Now, functionality means ease to deal with bugs whose correction creates more bugs. On the other hand, evolvability implies the possibility to make functional changes on top of not perfectly functional changes. So, we have two questions. Frst: Does Java + Java IDEs help in these regards more than other languages? Second: Can Java successfully assimilate the virtues of other languages just as it did with functional programming in Java 8? These terrible questions surely lack a single answer, one that is accepted by everybody and so, many languages are continuously created and some are gaining favoritism, say, Kotlin, Ruby, Groovy, Swift, Scala, Hashkell, Go.

In conclusion, extremely wise remodeling does not grant eternal life to Java. At last, Java will look like a sort of Frankenstein. So, the time will come when we with our Java and JavaFX will be an obstacle for those that want to move into the future. Can we make something to make revolutions a bit less traumatic? Our directive in this regard might be summarized as follows:

Keep and eye on the future -look here and there- but at the meantime the best way to get ready for it is to work hard with Java and JavaFX.

```
Tim Spann (2015)
Yes, Java Has Flaws. But...
Apr. 15, 16 · Java Zone
https://dzone.com/articles/java-and-the-superfriends-stronger-than-one
```

4.5 Conclusion

We have had an encounter with the world of tools, specifically for the management of meshes. More than an encounter, it has been a crash. This is a fair illustration of simple facts of complexity: one can afford with half cooked products for most

problems. But to get at perfection it takes more than a serious effort. This operationally means that one needs many trials before success and that at every moment we are plenty of non top quality goods.

Chapter 5

JavaFX: sensitive meshes

`Meshes that feel`

46 ***Introduction.*** *We know how to produce cylindrical mutations. To simulate evolution we need to add (artificial) selection. In regard with the hand, its fitness is defined by its capability to hold fast a ball of appropriate radius. While we could assess how fit is a hand by direct inspection using our applications, the natural progress is towards automation. In that regard, we know how to close the hand but the bones offer no resistance when being traversed one by another. In consequence, we do not know how to detect that the ball has been grasped nor to stop the palm being traversed by the fingers. That is why, we need to endow our objects with intersection sensors.*

47 ***Purpose.*** *We explore some of the possibilities that JavaFX offers to endow meshes with sensitivity.*

5.1 Tetrahedrons from Internet

`F(x)yz` is a library to extend the power of JavaFX. One of its classes is of immediate interest for us: tetrahedrons. Let us play with it.

48 ***Exercise.*** *Let us run from `F(x)yz` a class that builds a tetrahedron with fancy texture:*

1. *`Go to its site of F(x)yz:`*

   `http://birdasaur.github.io/FXyz/`

and find the link to download the .zip version of a NetBeans project. Download it to an appropriate folder.

2. *On NetBeans, import this project: go to menu*

   File − > Import project − > From ZIP − > To your folder ...− > OK

3. *To correctly set up the platform look under the Project tab of NetBeans for the* `FXyzLib` *project. Right click over its name. Choose the menu:*

   Properties − > Run − > Runtime Platform − > Project Platform − > Manage − > Add platform − > Java Standard Edition − > (find folder..) − > Close − > OK

4. *Expand the tree of package* `org.fxyz.tests` *and select* `TetrahedraTest.Java`*. Run it with the menu*

   Run − > Run file.

   Observe the beautiful 3D pattern.

5. *Play with the many tests of this package. Pay attention to the style of coding that is proper of professionals and not of people that were forced to learn Java to solve very specific problems. The immediate difference points to a very high complexity level, next we see a difference in the architecture: it is not an addition of functions but a masterpiece: it is a great, coherent, multifunctional and well planned project. Besides, it uses naturally the full power of the language in its last version. Moreover, it was developed by a team and not by a single person so, delivery time shortens but the price to achieve coherence increases.*

6. *Return now to the study of* `TetrahedraTest.Java`*. JavaFX provides plane colors to color triangles. Try now to imagine how was engineered the capability to color the triangular faces of this tetrahedron in an almost pixel by pixel fashion. In particular, decide whether or not the grains of texture are tetrahedra.*

7. *Test your explanation: make suitable experiments, say, by silencing portions of code here and there, or by moving the value of suitable arguments. Guide yourself by the flow created by the semantics, by the meaning entailed in the names of methods and variables.* *Answer*

5.2 Our tetrahedron

Let us incorporate a tetrahedron into an Xform that moves around:

49 ***A planet with the form of a tetrahedron is presented by*** **`Program N49 Tetrahedron`**.

50 ***Exercise.*** *Run the previous program and play with the code.*

51 ***Exercise: The tetrahedron tool.*** *Extract from the previous program a class to synthesize a tetrahedron that could be used as a generic tool. Compare your answer with that provided in the package* `Tools`*, class* `tetrahedronTool.java`*. Rewrite the previous program using the new tool. Our answer is in* `Program P51 Tetrahedron2`*.*

5.3 Detecting picking by the mouse

Let us review in which form a JavaFX object can be endowed with the possibility to detect that it has been picked by the mouse. To that aim, we have the `.getIntersectedFace()` method.

52 ***Exercise.*** *A wonderful and delightful application of the* `.getIntersectedFace()` *method can be found in program* `ClothMeshTest.java` *of the package* `org.fxyz.test` *of the aforementioned library* `F(x)yz`*. Run this program and try to imagine the role of the* `.getIntersectedFace()` *method.*

It is our turn now.

53 ***A sensitive tetrahedron:*** *it is endowed with the capability to report that it has been picked up and on which face. This program is based on our previous work, program M89 MeshCubePickDemo of Vol XIII. The code can be found in* `Program P53 Tetrahedron3`*.*

54 ***Exercise.*** *Run the previous program and play with the code.*

5.4 Detecting collisions

To detect collisions, JavafX provides for 2D the operator `intersect(Shape shape1, Shape shape2)` that returns the shape that is subcontained in both shape1 and shape2. This operator is pressumibly too expensive because it is very precise. Next, we have the method `a.intersects(b.getBoundsInLocal())` to detect whether or not objects `a` and `b` have intersected one another.

55 ***Exercise.*** *Review and run* `program P55 CircleCollisionTester` *to see how the operator*

`intersect(Shape shape1, Shape shape2)` *is put to work for 2D.*

Let us try to implement the sensing of a collision on our own. Static collisions that originate after translapping creation of static objects already was tested in our previous work: intersection of 3D boxes was presented and used by `Program M63 FossilRecordd10`. Nevertheless, we do not know how this operates for moving objects.

56 ***The code in*** **`Program P56 Tetrahedron4`** ***tests the method*** **`a.intersects(b.getBounds`** ***to detect collisions of moving objects. In reality, programmed detection is addressed to detect collisions among the boxes that bound given objects.***

57 ***Exercise.*** *Run the previous program and play with the code. Play enough to agree else disagree with the Author: the program is a failure, it responds too late and possibly erratically.*

58 ***Exercise.*** *Is the failure of the previous program a result of bad programming? To begin with, the code of the previous program is redundant because planets are created by a duplication of methods. Simplify the code and include more sequential/parallel transitions to see what happens. Our answer can be seen in* `program P58 Tetrahedron5.`

Let us study some appropriate material from Internet.

59 ***Exercise.*** *Run program P59 CollisionsOfBalls and review its code to see in which form collisions are recurrently detected.*

60 ***Exercise.*** *Follow the strategy of the previous program to add collision detection to our tetrahedra planetary system. Our answer is in* `Program P60 Tetrahedron6`*. It is another failure.*

61 ***Exercise.*** *Given that we have been unable to fulfill our desire of satisfactorily detecting collisions among complex meshes, try it out with simple ones, i.e., with boxes. Our answer is in* `program P61 Tetrahedron7`*. We failed.*

Let us pass to another theme that will be needed below.

5.5 Polygons

We will need the know how to calculate the **area of a quadrilateral**. JavaFX has a method to do that.

62 ***The code in*** `Program P62 Polygone` ***shows how to calculate the area of a quadrilateral in pixel units.***

63 ***Exercise.*** *Run the previous program and play with the code.*

5.6 Conclusion

We have explored some JavaFX tools to detect collisions that are convenient to simulate the evolution of the hand. We were unable to have they working. We consider that this is a problem in JavaFX itself.

Chapter 6

Simulating the evolution of the hand

`Mutations + selection`

64 ***Purpose.*** *We know how to produce cylindrical mutations of the hand. Our purpose is to enroll resultant mutants in an ad hoc evolutionary process. So, we need to learn to simulate selection. But, since we failed in the important task of detecting collisions of JavaFX objects, we cannot simulate a hand holding an object so, we will rely on extremely simple models.*

65 ***Our selection criterion***

By observing own hand when fingers bent in toward the palm, one sees that fingers and the palm do not match planes but that they naturally make place for a ball although a place for a club also appears. If the hand gets closed completely, a fist is conformed.

A ball is an idealization of a pebble so, this gives rise to the evolutionary idea that the hand was evolved to fit the function of grasping stones that might be used as missiles. Chimps can use stones as missiles but cannot direct the shot while squirrels love to throw nuts and pine cones at human beings that, say, enjoy a picnic. Rats have human-like hands but use them not to throw things but to climb. The exercised function is equivalent in human beings to grip a club.

So and in agreement with most evolutionists, to grab stones and to grip clubs are possibly the two most important functions of the hand. By contrast, to use fists as weapons is an exclusive human property because animals prefer the use nails when these are strong. Bears, for instance, have blinded various excursionists with

a strike with their nails. Animals are masters in the management of nails so, they can play and they can also fight with them: cats are a well known example.

To measure fitness of a hand, we have the following proposal: we make the hand to grasp a ball. Next, we measure the redundant space in within the ball and the hand. Hands with the smallest vacuum will be selected for. The purpose of our optimization problem is to minimize this vacuum because we consider that the grabbing of the ball will be better the less is the redundant space left by fingers.

Let us underscore that our simulation is not framed by Darwinian Evolution. Rather, we are dealing with creationism because the selection criterion is given by us. To get into the realm of Darwinian Evolution we need to delete our imposed criterion and to directly tie mutants to differential reproduction without touching anywhere the anthropocentric concept of function. Our previous work, on hand made out of boxes, allows us to claim that deletion of selective criteria might result in great variability in the form and function of hands that mimic diverse weapons and tools. We find all this extremely complex so, we need more time to make JavaFX 3D works appropriately. At the meantime, we will rely on extremely simple models.

6.1 The simplest hand: a tail

The simplest biological boned model of a hand is a tail that next will be oversimplified by us into a polyline.

66 ***A mandatory retroprediction.***

A hand is a marvel. A tail also is but it is simpler. So, we stumble with a first retroprediction: the evolution of the hand must have had gone on top of the evolution of tailed hands. Our prediction is mandatory. What do we observe in nature or in the fossil record? Snakes and the tails of american monkeys are all in this line. Are these elements connected one with another and with the human hand by evolution?

67 ***Simulation of constrained evolution.***

A finger with 3 phalanges and its corresponding metacarpal is similar to a tail and will be idealized by a broken line with 4 strokes. The purpose of the next simulation is to determine the relative length of bones under the following selective criterion: The tail must cover a circle to allow maximal grasping. Our guess is that this is accomplished when the space in within the circle and the formed quadrilateral gets a minimum. We will observe a constraint: the number of bones is fixed.

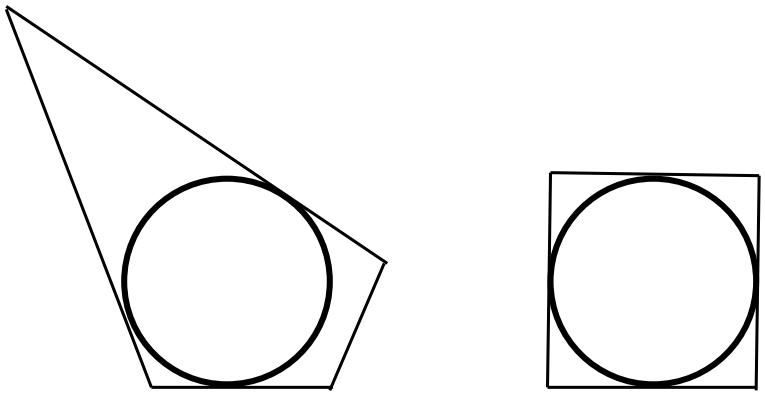

Figure 6.0. A tail with four bones can embrace a circle but the space in within the circle and the resultant quadrilateral might vary. Our purpose is to select the arrange with the minimal space in excess. We expect a square. Our guess is that this arrange will favor the maximal grasping of the circle by the tail that we idealize as a polyline with 4 strokes.

Our procedure to build quadrilaterals whose 4 edges are tangent to a circle is as follows:

1. We declare the first edge to be tangent to the circle at angle 0 (Cartesian coordinates).The corresponding tangent line is vertical. Next we throw and angle with value in the interval $(0, \pi)$. Next we throw and angle in the interval $(\pi, 2\pi)$. Next, an angle in the interval $(0, 2\pi)$. The three values are ordered counterclockwise.

2. With the aforementioned construction we have checked the condition for the tangents to conform to a quadrilateral: the three sampled points must lie not all on the same side of the circle, if one is in in the interval (0, π) at least one of the others must be in (π, 2π). If a set of 3 angles do not fill this condition, it is discarded. Besides, any two consecutive angles must not differ by an angle greater than π.

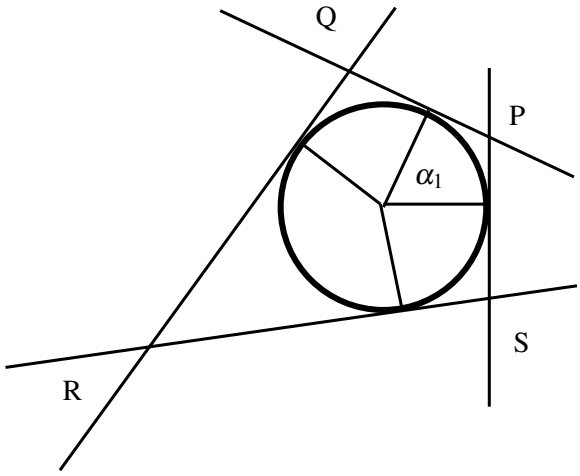

Figure 6.1. Four angles determine four tangent lines whose intersection generates the four vertexes of a quadrilateral. The first angle is zero and the associated tangent line is vertical, to the right. The center of the circle is the origin of coordinates.

To find the point of intersection of two lines we proceed as follows:
The equation of a line is $y = mx + b$
But if a line is tangent at the circle at a point whose angle is α, we get:
$y = sin\alpha = mcos\alpha + b$
$b = sin\alpha - mcos\alpha$
Besides, the slope of the tangent line is $m = \frac{-1}{tan\alpha}$
Now, we can get the point of intersection of two lines:
$y = m_1x + b_1$
$y = m_2x + b_2$
$m_1x + b_1 = m_2x + b_2$
$(m_1 - m_2)x = b_2 - b_1$
$x = \frac{b_2 - b_1}{m_1 - m_2}$
and $y = m_1x + b_1$.
The code can be found in `Program P67 Polygone2`.

68 ***Exercise.*** *Run the previous program and play with the code.*

69 ***Exercise.*** *Reorganize the previous program for reuse: a call to a single method must produce the area of a quadrilateral whose edges are tangent to a circle. Our answer is in* `Program P69 Polygone3`.

70 ***Evolution as a test: let us compose a genetic algorithm, a simulation of evolution, to test the idea that our model predicts a tail whose vertebrae have all the same length.*** *The code can be found in* `program P70 EvolutionOfTail`.

71 ***Exercise.*** *Run the previous program and play with the code. Play enough to agree else disagree with the Author: the algorithm does not converge perfectly to a square but dances around. Moreover, excess area with respect to the square seems to be overestimated.*

72 ***Exercise.*** *Deal with the aforementioned troubles and solve them: the algorithm must converge to a square and also graphical and numerical outputs must agree. Our answer is in* `program P72 EvolutionOfTail2.`

73 ***Exercie :*** *Show that the type of evolution simulated in the previous exercise is not Darwinian. Remake the code into a simulation of a truly Darwinian Evolution.* *Partial answer*

74 ***Challenge :*** *Use the code that was developed in the last exercise to measure the effectiveness of a hint. You will need to compare the code of this program with its Lamarckian version. To that aim, you can use a facility of NetBeans: select the names of the two files in the* `Projects` *window, go to the menu* `Tools` *and select* DIFF. *You can select two files that are one after the other in the* `Project` *window with* `CAPS + arrowUp`.

75 ***Challenge :*** *We have simulated the evolution of a tail with an invariable number of vertebrae. Reuse the code to simulate a tail with variable number of vertebrae. Prove that, with the same objective function, evolution tends to augment the number of vertebrae towards infinite. Somewhere along the fossil record a transition from too many vertebrae to some few but curved ones shall be observed.*

6.2 Conclusion

We have begun our quest for a full fledge simulation of the evolution of the hand and of any boned structure. Our starting achievement is very humble: we studied the evolution of the simplest model of a hand, a prehensile tail. Assuming that the function of the tail is to grasp the arms of trees as strongly as possible, and that this is achieved when the bones of the tail circumvent a circle creating extra space as small as possible we were able to prove that all vertebrae have the same length. Because we used a constant number of vertebrae, the final pattern fits a square. But, it was given as a challenge to prove that if this condition is released, the number

of vertebrae will increase towards infinite. We used a genetic algorithm that was subsequently cleansed from Lamarckian hints. Nevertheless, genetic algorithms are not Darwinian: they include the selection criteria in the form of an objective function and that is not Darwinian. A simulation to be purely Darwinian must be free from hints and from the concept of function and of any other anthropocentric concept altogether.

Chapter 7

The hand and the chimp-man gap

The hand as tracer of evolution

76 ***Introduction and purpose.***

The great and abundant similarities among chimps and human beings make of the Evolutionary Theory of the origin of species by descent the natural null hypothesis of science. Nevertheless, it is also natural to see that this hypothesis is obviously false:

It is not easy to reuse a chimp to make a human. It is very hard. What does it mean? It means that if such reuse is successful, a lot of trials must had been made and if that was done by evolution, a lot of imperfections in the fossil record must be found along with too many malfunctions in extant populations. Such a prediction is important and mandatory but is not fulfilled.

The mission of our community is to pass from wording to simulations that shall be specific, well posed, clear cutting and determinant to decide whether or not our null hypothesis is to be rejected. To that aim, it seems that any complex trait must be a good terrain for experimentation. So, the hand seems to be good and challenging enough for this purpose.

We also shall explain why International Science accepts our null hypothesis while it is obviously false. In fact, the error of International Science is that it pays attention to the possibilities of evolution but not to the scientific method: the possibilities of evolution must not be filtered to explain observed events. Instead, those possibilities must be calculated to the full, important and mandatory predictions must be formulated to be next contrasted with all data. This is difficult and our purpose here is to put ourselves at the start of the road to that magnificent project:

we will show what we mean by random mutation, the fundamental creator of variability in Darwinian Evolution.

7.1 Differences between hands

There is a simple trait thanks to which we can readily imagine some important differences between the human and chimp hands.

77 ***A discriminant trait.***

Human beings can write but chimps not. That is due to the fact that we have some muscles associated to the thumb that the chimp lacks. In principle, four muscles are necessary and sufficient to write: one that pulls towards the left, other to the right, a third towards front and the last towards the rear. Now, chimps can took hold of arms of trees much as we, although their muscles of their thumbs are weaker. So, the chimp have two of the four muscles that would be needed to write, one that pulls towards the front and the other towards the rear. The hand of the chimp lacks two additional muscles to move the thumb to the left and to the right. Curiously enough, anatomists speak of three additional muscles instead of two (Young [5], 2003). Those muscles are: flexor pollicis longus muscle (Wikipedia [2], 2016), the deep head of the flexor pollicis brevis (Wikipedia [3], 2016) and the first volar or palmar interosseous muscle (Wikipedia [4], 2016). The Author does not understand the role of that redundancy: if more than two extra muscles would be desired, four or eight would be a better figure. In general, it seems to me that our Community will need some 30 years to mature its understanding of the hand.

78 ***International Science explains the trait.***

We can invent a simple explanation of the trait under discussion according to used Modern Evolutionary Thought: the capability to write that humans have is just the product of duplication of extant muscles in the chimp that was accompanied with adaptive relocation of insertion points that co-evolved with the corresponding neural connections and brain structures. This changes caused a general reformulation of the architecture of the hand.

As we see, there is no problem in enjoying evolution as a dogma. But we are a bit different: we test important dogmas in only that is possible. Now, to test the Evolutionary Theory with Java is perfectly possible and no matter how difficult it could be, we will do it to any desired level of demanded perfection and clear cut falsification power.

79 ***The ingredients of our null hypothesis***

To pursue our own ideas, let us focus on muscles, whose duplication plays a fundamental role in the discussion about the origin of the capability of humans to write. Duplication of fingers is a common mutation and so, most people know that there are persons with six fingers in their hands whose neural connections work perfectly but without independence. The author has no idea about the rate of duplication of separate muscles but no theory can convince him that such separate events cannot exist. Therefore, we will rely on the possibility of duplication of muscles together with a change in their insertion points. So, our null hypothesis reads as follows:

Dogma: Muscles can duplicate and their insertion points can migrate. Moreover, all these changes happen at random.

Null Hypothesis: Random mutations that cause duplication of muscles and migration of their insertion points are the sufficient cause of the appearing of the human hand beginning from the hand of the chimp or some related common ancestor. Natural selection of mutants for sophistication of achievable tasks, say, throwing, clubbing, manufacturing and dexter use of tools, filtered mutants to end with the type of hand we enjoy now.

It is a sacred duty for our community to calculate this null hypothesis. This is the reason of our existence. As our future work will show, this is extremely complicated. The plan is to begin with whatever we can to fire a quest for complexity and perfection. Thus, our aim for this chapter is extremely humble: it is to illustrate just what is the output of random mutation and what mandatory predictions of evolution could be envisaged. So, let us begin.

7.2 Random duplication of muscles

To visualize what we mean by random duplication of muscles with migration of insertion points all we need is to represent a muscle by a line in 3D.

80 ***What is mutation at random?*** *Let us represent muscles by lines to next develop the code that illustrates what we shall understand by mutation at random in regard with duplication of muscles and of insertion points. The code can be seen in* `program P80 MuscleMutation`*.*

81 ***Exercise.*** *Run the previous program and play with the code. Play enough to agree else disagree with the Author: The asked muscles that will enable writing surely will appear but its probability is orders of magnitudes less than the probability of appearance of muscles that join other bones not related with the thumb. Thus, we got a mandatory falsifying prediction of the Evolutionary Theory. Because that prediction is not fulfilled, the Evolutionary Theory is false.*

82 ***Definition.*** *A* `mandatory falsifying prediction` *is defined on the image of the tale of the fake sportsman that is a man that says:* `I have run 20Km to come here. We answer: Did you? OK, you are here. That is a good point. But, where is your perspiration? where is you sweat? There is neither sweat nor perspiration so, you are here just to cheat`*. More officially, a mandatory falsifying prediction of a theory is one that backs the theory with a not nil probability but makes additionally the prediction of other many accompanying events and assigns to them huge probabilities making them unavoidable. So, if unavoidable accompanying events are not observed, the theory is rejected because the absence of unavoidable accompanying events denies the right to exist of the claimed event that we want to explain.* ***Mutations*** *are called* ***good*** *when they fit the claimed pattern. Otherwise they are called* ***falsifying****. Thus, the ratio of good mutations to falsifying ones is 1 to large N:*

MANDATORY FALSIFYING PRDICTIONS		
FAKE SPORTSMAN		*THE EVOLUTIONARY THEORY*
PRETENSIONS		
I am here after running 20Kms.		The chimp-man gap was filled by evolution.
GOOD POINTS		
You are here.		The chimp, the man and evolution all exist.
MANDATORY ACCOMPANYING PREDICTIONS		
Where is you sweat?		Where are the non functional mutations?
Where is your perspiration?		Where are the bugs of evolution?

83 ***Exercise.*** *Add some code to the previous program to report the proportion of muscles that connect the metacarpal associated to the thumb to neighboring bones. Use this result to produce a crude estimate of the disproportion of probabilities of mutations that enable writing to that of accompanying falsifying mutations. Our answer can be seen in program* `P83 MuscleMutation2`*. As expected, the achieved ratio for large numbers of mutations was in the vicinity of 1 to 20. For small numbers sampling effects create large variability.*

84 ***Exercise.*** *Answer to the following criticism: Mutation and Evolution have too many plausible forms of operation. In the previous program we have depicted one mutational process that is one among myriads. Many more processes are important. For instance and to say the least, envisaged events involve the migration of two insertion points. But, mutations that involve the migration of just one point would be definitely more probable than those that involve two. So, the previous calculation deals with irrelevant improbable facts not with the most probable. Besides, the program assumes that in relation with writing only the hand is involved but the truth is that all big muscles that animate the hand are connected to the arm. Important for our discussion is the flexor pollicis longus muscle, absent in chimpanzees, that empowers the thumb. It is connected to the radius. So, considered mutations are an extremely oversimplified cartoon.* Answer

85 ***Interlude.*** *The Evolutionary Theory is obviously false. Everyone knows this by instinct by just looking at Heavens and Earth. So, everything we do is to drop water over the sea. Anyway, this is very nice. And extremely challenging. Anyway, we hope to pass someday from concocting falsifications to try to understand what a hand is. This means that we shall input a program with a function to optimize and the program must output a human hand. The demanded level of sophistication is so high that to replace a muscle by a line will not do it. So, let us develop a code for a mesh that will resemble a muscle a bit more accurately than a line.*

86 ***A basic mesh for a muscle.*** *We reuse Program M132 HandChimpHuman6 of Vol XIII. The code can be seen in Program* `P86 Muscle2`.

87 ***Exercise.*** *Run the previous program and play with the code. Run the diverse versions of the method* `radius(float z)` *and choose the one you like the most. You might prefer to make additional tries. The Author chose the radius generated by the sinus function.*

88 ***Exercise.*** *Our purpose is to devise a* `Class Muscle` *that shall be built on meshes on the image of* `Class Line3D`. *In that way, it will straightforward to replace lines by meshes in our programs that simulate the duplication of muscles. To begin with, encapsulate the mesh in program* `P86 Muscle2` *in an inner class of a new version of this program. Our answer can be seen in program* `P88 Muscle3`.

We can exhibit now our meshes into our simulation of duplication of muscles.

89 ***Meshes in place of lines.*** *In the following program we considered duplication of muscles that is accompanied with migration of just one insertion point. Lines have been replaced by meshes in program* `P83 MuscleMutation2`*. This can be seen in program* `P89 MuscleMutation3`*. Muscles are instances of the* `Class Muscle` *that has been added to package* `Tools`*.*

90 ***Exercise.*** *Run the previous program and play with the code.*

7.3 Conclusion

We have made the following definition: A `mandatory falsifying prediction` is defined on the image of the tale of the fake sportsman that is a man that says: *I have run 20Km to come here. We answer: Did you? OK, you are here. That is a good point. But, where is your perspiration? where is you sweat? There is neither sweat nor perspiration so, you are here just to cheat*. More officially, a mandatory falsifying prediction of a theory is one that backs the theory with a not nil probability but makes additionally the prediction of other many accompanying events and assigns to them huge probabilities making them unavoidable. So, if unavoidable accompanying events are not observed, the theory is rejected because the absence of unavoidable events denies to the claimed event the right to exist. **Mutations** are called **good** when they fit the claimed pattern. Otherwise they are called **falsifying**. In relation with the chimp-man gap, The Evolutionary Theory needs the appearance of 3 muscles to enable human beings to write. The natural way to explain this is by duplication of extant muscles ensued by the migration of insertion points. If we suppose that duplication happened at random, as Darwin demands, then the demanded mutations possibly happened but were accompanied with many falsifying duplications. Our estimation rendered a ratio of 1 good mutation to 20 falsifying ones. This is a toy model of a refutation of the Evolutionary Theory and must be taken seriously as a promise for future work but not as an actual refutation.

Chapter 8

Conclusion

This is a challenge for you

Answers

48, page 19. To begin with, we silenced the `AnimationTimer timerEffect` around line 170 in program `Tetrahedra.test`. To do that we silenced the instruction to start:

```
//          timerEffect.start();
```

Silencing produces no change. Next, we silenced in line 107 a call that adds a tetrahedron to the scene:

```
//sceneRoot.getChildren().addAll(group);
```

The program runs but no tetrahedron appeared. The line is activated anew to go further. So, the mystery of the texture must be looked in the program that creates the tetrahedron. To find it, we right click on the constructor with 3 arguments in line 88 and select `Navigate -> Go to source`. A new tab is open with the content of the `public class TetrahedraMesh extends TexturedMesh`. We find the constructor with 3 arguments in line 66 and determine that the important instruction is `updateMesh()` in line 71. This leads us to line 148 and from there to line 159:

```
private TriangleMesh createTetrahedra(float height, int level)
```

The specifications of a mesh are found. By inspecting the code we arrive to the conclusion that the variable level is responsible for the number and size of coloring grains of texture. To test this idea, in line 88 of program `Tetrahedra.test` we change the second argument that takes the value 7 to 2, 3, 4, 5. Running the program with these settings shows us that we are correct. But, What do grains consist in? Are they triangle or tetrahedra? The answer is read from the instruction of line 162 in program `TetrahedraMesh`:

```
if(level>0){
            m0= createTetrahedra(height, level-1);
        }
```

The diverse levels invoke a recursion to the same constructor of tetrahedra. So, our grains seem to be tetrahedra. Anyway, to be sure we must go to the coloring procedure: where is it? It can be researched beginning with line 99 of program `Tetrahedra.test`.

```
tetra.setTextureModeVertices3D(1530,p->p.magnitude());
```

By the way, we observe that one can pass functions as arguments, such as `p->p.magnitude()` . This was made possible since Java 8. Now, the aforementioned method and its whole parent class `TetrahedraMesh` are by themselves enslaved to previous work, i.e., to programs `TetrahedraMesh` and `TetrahedraTest`. This means that we must return to the constructor method. To see the entrails of the pyramid, in line 91 we change the instruction

```
 tetra.setCullFace(CullFace.BACK);
```

by

```
tetra.setCullFace(CullFace.NONE);
```

Next, we give to `level` in line 88 of class `TetrahedraTest` the value of 0: this is a pyramid. For value 1, it looks like some pyramids inside a mother one. For value 2, it becomes clear that subtriangulation refers only to exterior faces of the mother tetrahedron but not to inner sheets. So, our conclusion is that the grains of texture correspond with the triangles of a mesh and the recursion in the constructor just divides a triangle into 3 subtriangles that is achieved by joining middle points of the edges of triangle. So, we look inside program `TetrahedraMesh` for a method to get middle points, which might be taken from JavaFX else be built by the developer. That method is found in two versions, one in line 351 and the other in line 363. These versions keep track as of points as of their indexes in the list of vertexes. One of those versions is used in lines 291, where the vertexes of a triangle are called to return middle points. However, middle points are also necessary were the world filled in tetrahedra. So, how do we know for certain that our case is with division of triangles but not of tetrahedra? The reason is that middle points are found for three points without any loop above. By contast, for tetrahedra we would demand a loop with four rounds, one for each face.

73, page 29.

Darwinian Evolution has no incorporated hints into its code. Previous code has the following hints: the triad of angles that with angle zero will generate 4 tangent lines to a circle whose intersection will generate the vertexes of a quadrilateral is

made mathematically. Moreover, mutants are produced by tireless repetitive calls on the same procedure until a good mutant is produced (a well formed quadrilateral). Additionally, the fittest individual is cloned intact to the next generation, it is saved from mutation, recombination and drift, i.e., sampling forgiveness. Deletion of these Lamarckian hints will be ensued by disappearance of the population. Else, one must be ready to provide for a huge number of individuals together with unlimited time. To check this prognostics, we can run program P73 EvolutionOfTail3. (Alternate path to hide hints: devise a genetic algorithm to replace them. In this case, the algorithm itself will be the hint.)

On the other hand, the Evolutionary Theory relies on natural selection but the evolutionary process has no such anthropomorphic concept. The evolutionary process relies on differential surviving . And that is all to evolution. On top on it, we, human beings, are perfectly free to impose the concept of function if only we prove that such a concept can be disentangled from the game of *live and let die*. This was not done. That is why our new algorithm is near to Darwinian Evolution but matching has not been achieved.

84, page 35. Questions of this sort are tremendously important. So, everyone in our community shall be ready to create a better answer. Ours is as follows: Program `P83 MuscleMutation2` must not be understood as a simulation of some process but as a projection over time of the last move of all possible processes that end with the creation of a duplication of a muscle and with migration of its insertion points. Mutation certainly has too many more mechanisms to create variability than a direct duplication of muscles with migration of both insertion points. Nevertheless, we shall notice that not listed products must be added to the accompanying falsifying mutations. Our guess is that they add to myriads. That is why we claim that the probability of appearing of accompanying falsifying mutations is orders of magnitude larger than that of the appearing of the anatomical arrangement that enables writing.

8.1 Biblio

Bibliography

[1] JAMA (2012) JAMA : A Java Matrix Package
http://math.nist.gov/javanumerics/jama/
Verified 7/XI/2015 v

[2] WIKIPEDIA CONTRIBUTORS 'Flexor pollicis longus muscle', Wikipedia, The Free Encyclopedia, 14 February 2016, 08:19 UTC, https://en.wikipedia.org/w/index.php?title=Flexor_pollicis_longus_muscle&ol accessed 11 May 2016 32

[3] WIKIPEDIA CONTRIBUTORS Wikipedia contributors, 'Flexor pollicis brevis muscle', Wikipedia, The Free Encyclopedia, 4 April 2016, 16:32 UTC, https://en.wikipedia.org/w/index.php?title=Flexor_pollicis_brevis_muscle&ol accessed 11 May 2016 32

[4] WIKIPEDIA CONTRIBUTORS Wikipedia contributors, 'Palmar interossei muscles', Wikipedia, The Free Encyclopedia, 17 March 2016, 06:39 UTC, https://en.wikipedia.org/w/index.php?title=Palmar_interossei_muscles&oldid= accessed 11 May 2016 32

[5] YOUNG RW (2003). Evolution of the human hand: the role of throwing and clubbing. Journal of Anatomy 202(1):165-174. doi:10.1046/j.1469-7580.2003.00144.x.
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1571064/
Verified 26/III/2015 32