**UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI** FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENŢĂ

**Vegetation Sim**

Propusă de

**Ilieș Norbert-Mario**

Sesiunea: **Iulie, 2018**

Coordonator știinţific

**Lect. Dr. Ghirvu Lucian**

UNIVERSITATEA „ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

**LUCRARE DE LICENȚĂ**

Vegetation Sim

**Ilieș Norbert-Mario**

Sesiunea: **Iulie 2018**

Coordonator știinţific

**Lect. Dr. Ghirvu Lucian**

DECLARAŢIE PRIVIND ORIGINALITATEA ŞI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licenţă cu titlul „Vegetation Sim” este scrisă de mine şi nu a mai fost prezentată niciodată la o altă facultate sau instituţie de învăţământ superior din ţară sau străinătate.

De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

* toate fragmentele de text reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi deţin referinţa precisă a sursei;
* reformularea în cuvinte proprii a textelor scrise de către alţi autori deţine referinţa

precisă;

* + codul sursă, imaginile etc. preluate din proiecte opensource sau alte surse sunt utilizate

cu respectarea drepturilor de autor şi deţin referinţe precise;

* rezumarea ideilor altor autori precizează referinţa precisă la textul original.

Iași, Absolvent *Ilieș Norbret-Mario*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „Vegetation Sim”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” Iași să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programelecalculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iași, Absolvent *Ilieș Norbert-Mario*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Introducere**
   1. **Motivație**

Vegetation Sim este o aplicație care vizează, după cum sugerează și numele, o modalitate de a simula creșterea unor structuri vegetative pentru a putea observa în detaliu cum acestea se dezvoltă, cum cresc, după o structură bazată pe anumite reguli și/sau probabilități.

Am ales această aplicație pentru frumusețea estetică pe care putem să o observăm în momentul in care vedem, pas cu pas, cum natura își arată modurile, considerate până nu de mult, misterioase de a dezvolta plante, pentru scopul creativ prin care ne putem construi propriile structuri vegetative, iar nu in ultimul rand, cel educativ, anume că misterele despre care am scris mai sus sunt dezvăluite, putând întelege, astfel, mai bine natura.

Aplicația este implementată cu ajutorul limbajului de programare Java. Am ales acest limbaj deoarece conține toate elementele necesare pentru dezvoltarea acestei aplicații, iar aici ma refer in deosebi la librăria JavaFX cu ajutorul careia am putut implementa partea grafică. Un alt motiv pentru alegerea acestui limbaj este că Java este cel mai popular limbaj de programare in momentul de față, ceea ce permite o mentenabilitate a codului facută posibilă de un număr potențial cât mai mare de oameni.

[[1]](#footnote-1)

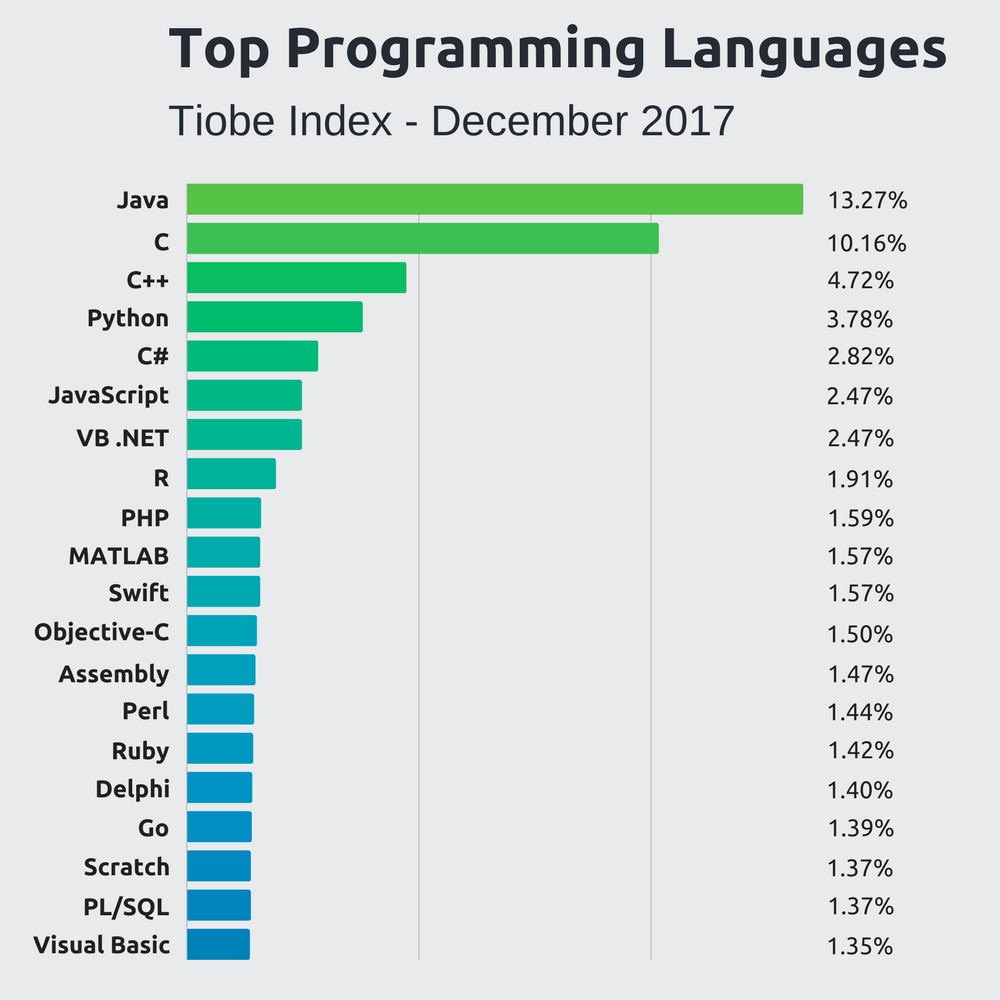


Figura . Topul limbajelor de programare la finalul anului 2017

* 1. **Context**

E-learning-ul[[2]](#footnote-2) este o metodologie a viitorului care presupune utilizarea de media electronică și tehnologii informaționale și de comunicație în educație. Pentru a putea pune în practică acest concept, este nevoie de diferite resurse cu ajutorul cărora oamenii să poată participa în mod activ la educație.

Vegetation Sim este o aplicație care ar putea avea aplicabilitate în metodologia e-learning, putând servi drept resursă pentru științe ale naturii, pentru a întelege creșterea vegetației.

O alta aplicabilitate a aplicației ar putea fi generarea procedurală a mediului înconjurător a unui joc 2D, dacă am avea o funcționalitate de exportare a plantelor într-un mod interpretant de către un motor de joc.

* 1. **Cerințe funcționale**
* Vizualizarea unor plante in format 2D
* Adăugarea/ștergerea/modificarea unui set de reguli care poate fi interpretat de către aplicație în mod graphic
* Generarea unei “grădini” pe baza regulilor existente
* Modificarea culorilor folosite
  1. **Sistemele L**

Un **sistem L**[[3]](#footnote-3) sau un sistem Lindenmayer este un sistem paralel de rescriere și un tip de gramatică formală. Un sistem L constă dintr-un alfabet de simboluri care poate fi folosit pentru a face șiruri de caractere, o colecție de reguli de producție care extind fiecare simbol într-un șir mai mare de simboluri, un șir inițial "axiomă" din care să înceapă construcția și un mecanism pentru traducerea șirurilor generate în structuri geometrice.

Acest model ar fi unul determinist - adică, dat fiind orice simbol în alfabetul gramaticii, a existat exact o regulă de producție, care este întotdeauna aleasă și întotdeauna efectuează aceeași conversie. O alternativă este să specificați mai mult de o regulă de producție pentru un simbol, oferindu-i fiecăruia o probabilitate de apariție. De exemplu, am putea schimba o regulă fixă în două reguli cu o anumită probabilitate:

Regulă simplă:

0 → 1 [0] 0

Reguli probabilistice:

0 (0,5) → 1 [0] 0

0 (0,5) → 0

* 1. **Tehnologii utilizate**

**Java**[[4]](#footnote-4) este un [limbaj de programare](https://ro.wikipedia.org/wiki/Limbaj_de_programare) orientat-[obiect](https://ro.wikipedia.org/wiki/Programare_orientat%C4%83_pe_obiecte) conceput de către James Gosling la [Sun Microsystems](https://ro.wikipedia.org/wiki/Sun_Microsystems) (acum filială [Oracle](https://ro.wikipedia.org/wiki/Oracle)) la începutul [anilor ʼ90](https://ro.wikipedia.org/wiki/Anii_1990), fiind lansat în 1995. Limbajul împrumută o mare parte din sintaxă de la [C](https://ro.wikipedia.org/wiki/C) și [C++](https://ro.wikipedia.org/wiki/C%2B%2B), dar are un model al obiectelor mai simplu și prezintă mai puține facilități de nivel jos. Un program Java compilat, corect scris, poate fi rulat fără modificări pe orice platformă care e instalată o mașină virtuală Java. Mașina virtuală Java este mediul în care se execută programele Java. În prezent, există mai mulți furnizori de JVM, printre care [Oracle](https://ro.wikipedia.org/wiki/Oracle), [IBM](https://ro.wikipedia.org/wiki/IBM), Bea, [FSF](https://ro.wikipedia.org/wiki/Free_Software_Foundation). În 2006, Sun [a anunțat](http://www.sun.com/2006-1113/feature/) că face disponibilă varianta sa de JVM ca [open-source](https://ro.wikipedia.org/wiki/Open-source).

[[5]](#footnote-5)

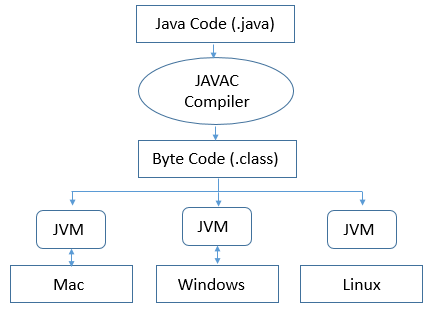


Figura . Compilatorul Java

**JavaFX**[[6]](#footnote-6) este o platformă software pentru crearea și livrarea aplicațiilor desktop, precum si a aplicațiilor pe internet care funcționează pe o varietate de dispositive. JavaFX a fost conceput pentru a inlocui Swing-ul, care a fost libraria standard de Java pentru a implementa o interfață grafică. JavaFX are support pentru computere desktop și browsere web pe Windows, Linux si macOS.

Dupa versiunea 2.0 a JavaFX, aceasta a aparut și ca librarie nativa de Java, putând astfel să se lucreze cu JavaFX folosind limbajul standard Java, nu cel declaravit, numit **JavaFX Script**.

Un exemplu de cod Java, utilizând librăria JavaFX pentru a face o aplicație de tipul “Hello, World!”:

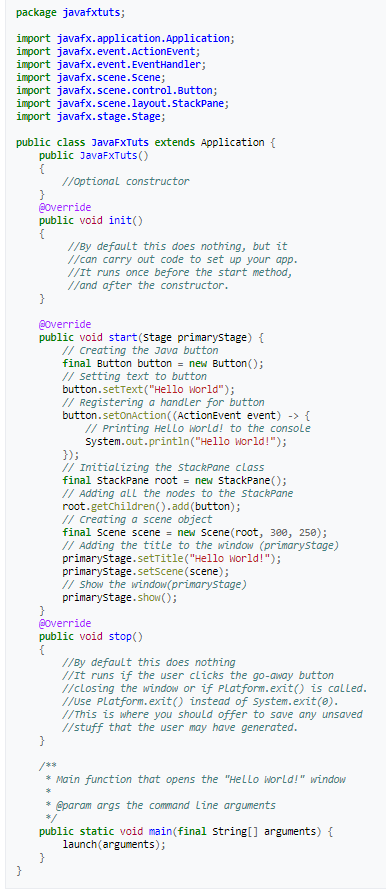
[[7]](#footnote-7)

Figura . Exemplu cod JavaFX

**IntelliJ IDEA**[[8]](#footnote-8) este un mediu integrat de dezvoltare Java (IDE) pentru dezvoltarea de software de calculator. Acesta este dezvoltat de JetBrains (cunoscut anterior ca IntelliJ) și este disponibil ca o editie comunitară licențiată Apache 2 și într-o ediție comercială proprietară. Ambele pot fi utilizate pentru dezvoltarea comercială.

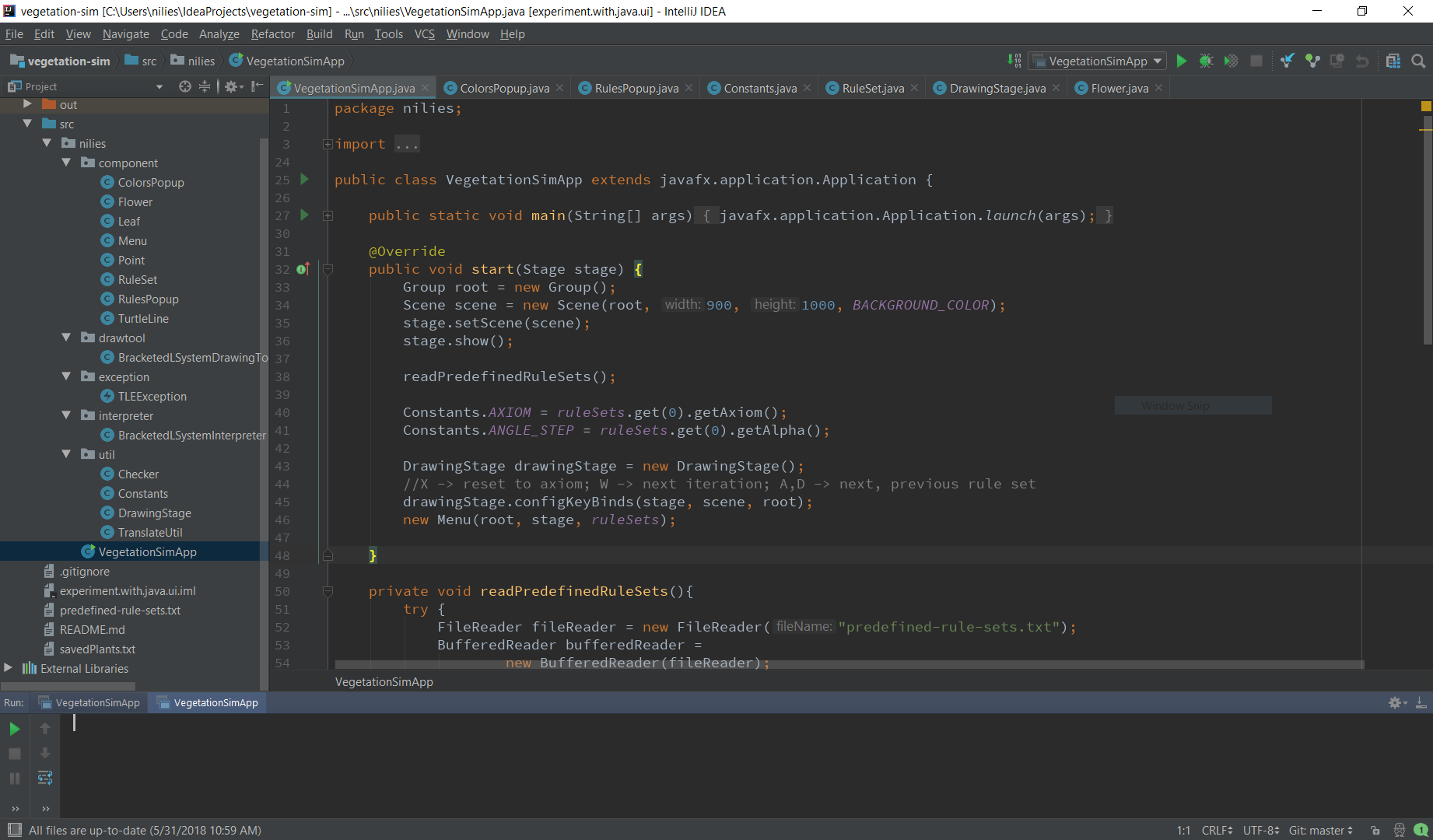


Figura . Mediul de dezvoltare IntelliJ IDEA

1. **State of the Art**

Sursa principală de inspirație și din care am cules informațiile teoretice necesare realizării acestui proiect este cartea *“The algorithmic beauty of plants”,* scrisă de Przemyslaw Prusinkiewicz și Aristid Lindenmayer. În aceasta este prezentată în detaliu structura sistemelor L (Lindenmayer, după numele autorului) și a liniilor de tip turtle de care m-am folosit.

În această carte găsim și exemple pe care le-am folosit:

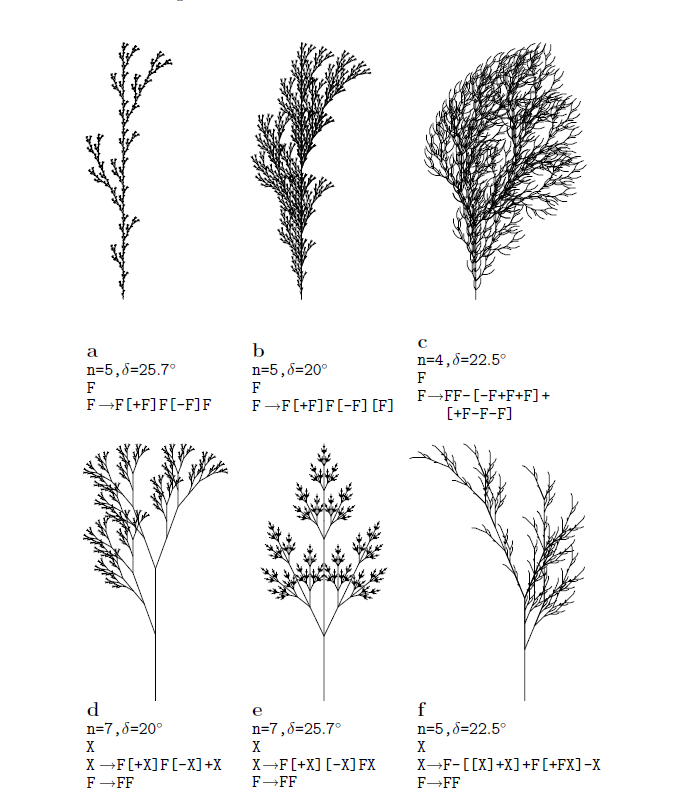


Figura . Exemple din cartea "The algorithmic beauty of plants"

Tot de aici m-am insipirat spre a face o grădină botanică generată pseudoaleator, folosind mai multe tipuri de plante.

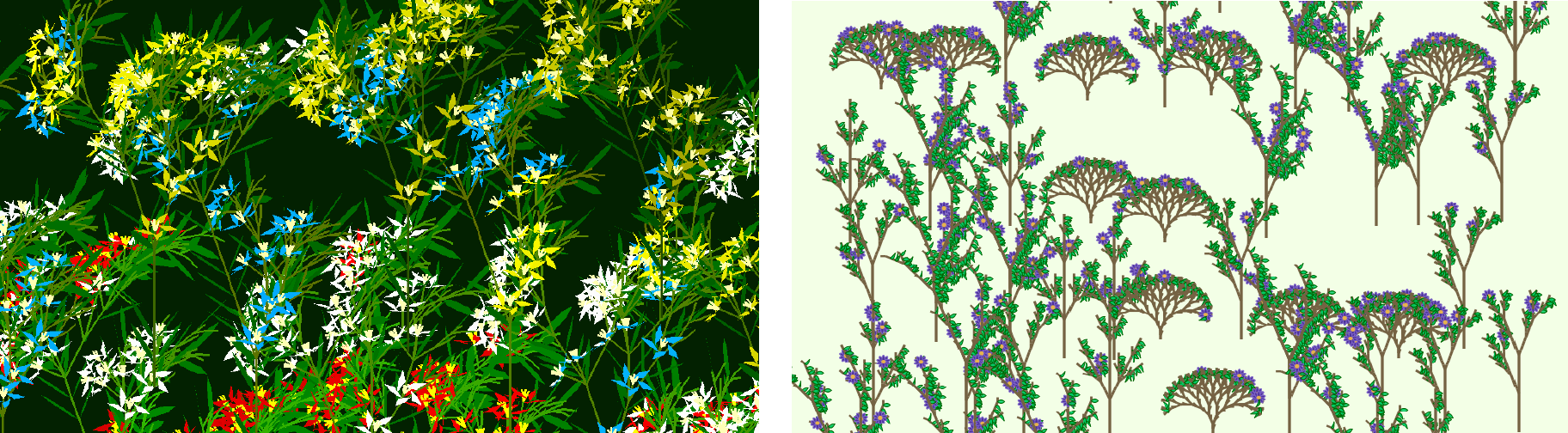
****

Figura . Comparație între grădina botanică din carte și cea din aplicație

Aici găsim multe detalii despre generarea copacilor și generarea 3D despre care aș dori să aprofundez pe viitor. Cu multe exemple vizuale, cartea este o adevărată inspirație pentru cei ce doresc să exploreze acest domeniu, oferind explicații mai mult decât pertinente pentru fiecare în parte.

Consider că *“The algorithmic beauty of plants”* a fost o fundație foarte bună pentru ceea ce am vrut să dezvolt.

1. **Structura aplicației**
   1. **Arhitectură**

Din punct de vedere arhitectural, aplicația este standalone, momentan, neavând o bază de date și nedepinzând de vreo altă aplicație sau third party.

Codul aplicației este impărțită în pachete sugestive care au rolul îmbunătățirii organizaționale și a mentenabilității codului.

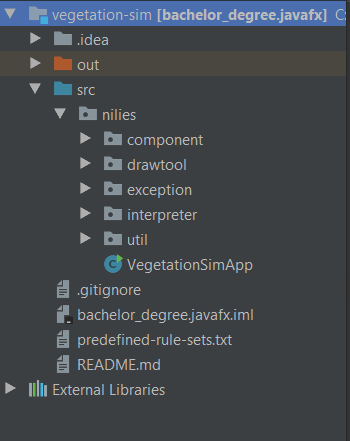


Figura . Structura pachetelor din aplicație

* **component** – acest pachet conține clasele care sunt destinate să fie obiecte “afișabile” (exemplu: flori, frunze, meniu etc.) sau obiecte generale care sunt folosite in orice componentă (exemplu: punct, linie etc.)
* **drawtool** – aici sunt destinate clasele ce au rolul de a desena obiectele in funcție de o propoziție care este interpretată în maniera sistemelor L; momentan, exista doar o singura implementare, dar se pot face mai multe cu anumite variații
* **exception** – în acest pachet găsim excepțiile personalizate pentru acest proiect
* **interpreter** – acest pachet conține clasele necesare dezvoltării unei propoziții in funcție de un set de reguli sau, mai concret, trecerea de la o iterație la alta
* **util** – în acest pachet găsim clase strict utilitare (exemplu: constante, verificatori etc.)
* **clasa VegetationSimApp** – acesta este “motorul” de unde pornim aplicația

Pe lângă acestea, mai există fișierul “predefined-rule-sets.txt” unde se află seturi de reguli pentru câteva plante care se incarcă în aplicație la fiecare rularea a acesteia. Un exemplu cu ce ar putea conține acest fișier se poate vedea in imagine:

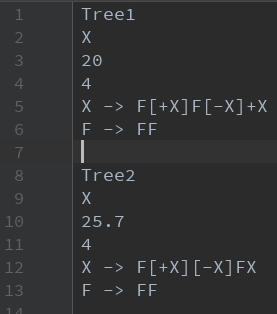


Figura . Exemplu de set de reguli

Unde:

* **Tree1** **→** nume
* **X** **→** axiomă
* **4** **→** număr optim de operații
* **X -> F[+X]F[-X]+X și F -> FF** **→** reguli de iterare

După ce intră in aplicație, utilizatorul are următoarele opțiuni:

* iterarea prin structurile deja existente
* observarea dezvoltării unei astfel de iterare
* modificarea culorilor
* adăugarea/editarea/stergerea unui set de reguli
* generarea aleatorie a unei “grădini” cu plantele existente

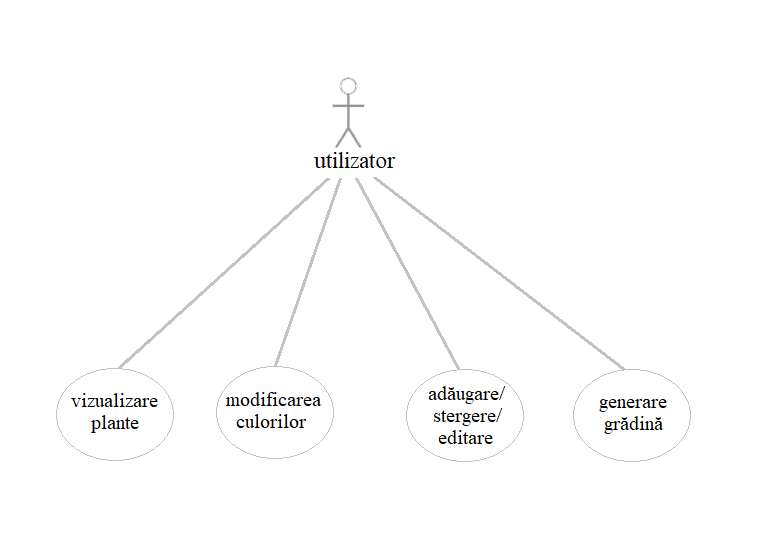


Figura . Diagramă de cazuri

* 1. **Versionare**

Aplicația a dispus de la bun început de un sistem de versionare, anume **git**, un sistem care îndeplinește acest scop de control al versiunii. Scopul principal este de a putea reveni la una din versiunile anterioare în caz că este necesar și de a ușura munca mai multor contribuitori la același proiect.

Folosirea **git**-ului a ajutat și la portabilitatea proiectului, putând lucra la el de pe orice mediu, atât timp cât dispuneam de conexiune la internet.

După ce am terminat de implementat proiectul, am putut trage diferite concluzii cu ajutorul acestui sistem:

* primul **commit** a fost făcut pe 29 martie
* proiectul constituie 100% cod Java
* în momentul de față sunt 27 de **commit**-uri

Folosirea unui sistem de versionare nu era o necesitate într-ucât a existat doar un singur contribuitor, dar celelalte avantaje m-au încurajat să folosesc un astfel de sistem, iar beneficiile au fost vizibile, putând să lucrez, după cum am spus mai sus, de pe mai multe mașini.

Putem vedea în imaginea de mai jos cum arată acest proiect, vizualizat online:

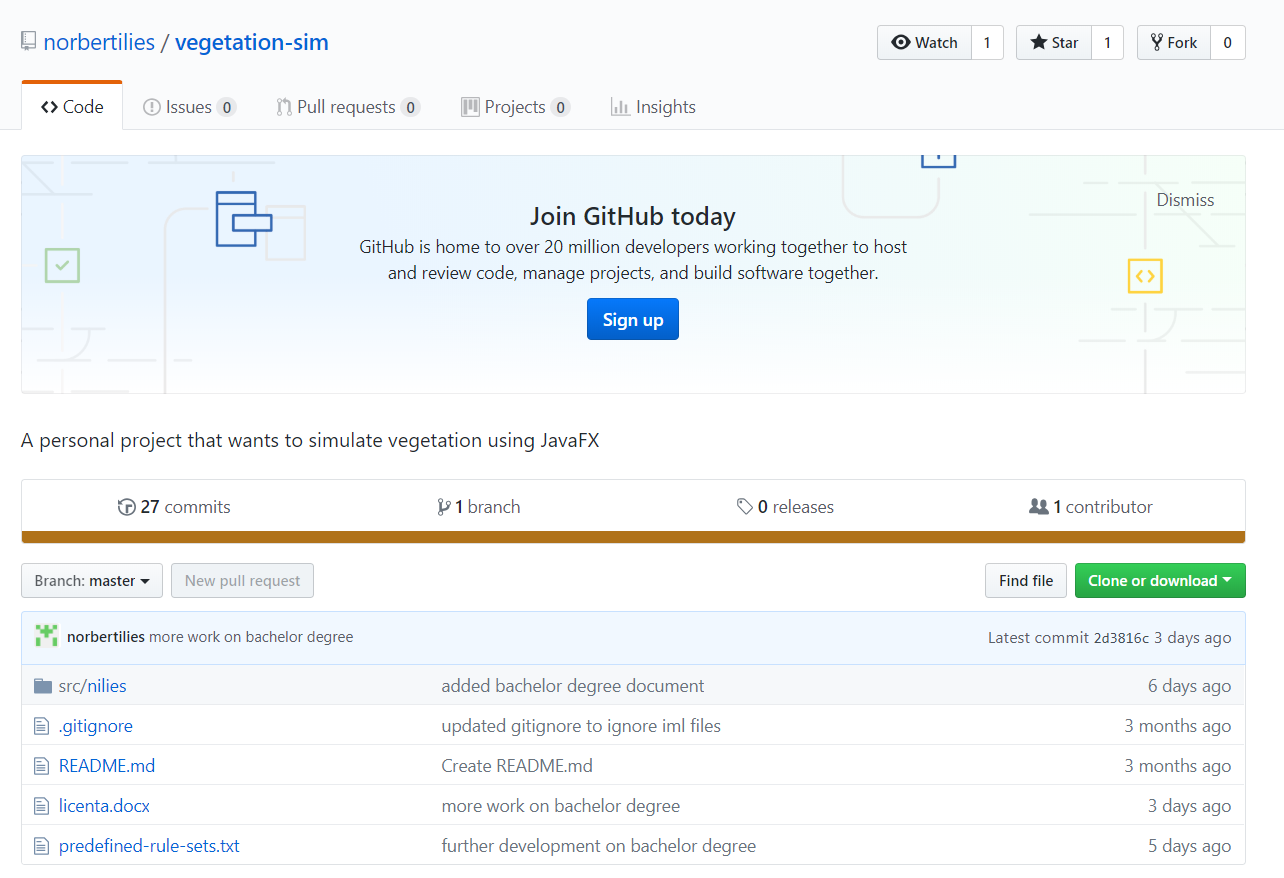


Figura . Aplicația pe siteul github

1. **Progresul vizual al aplicației**
   1. **Implementarea liniei țestoasă**

În primă fază am avut nevoie să implementez logica de “linie țestoasă”, concept ce simulează traseul pe care l-ar parcurge o țestoasă care poate primi 3 comenzi:

* Mergi înainte
* Rotește-te cu α grade la stânga
* Rotește-te cu α grade la dreapta

După ce avem acest concept implementat, am putut trece la următorul pas, anume construirea fractalilor, după câteva reguli cunoscute. În imagini se pot observa perechi de (set\_reguli, reprezentare):

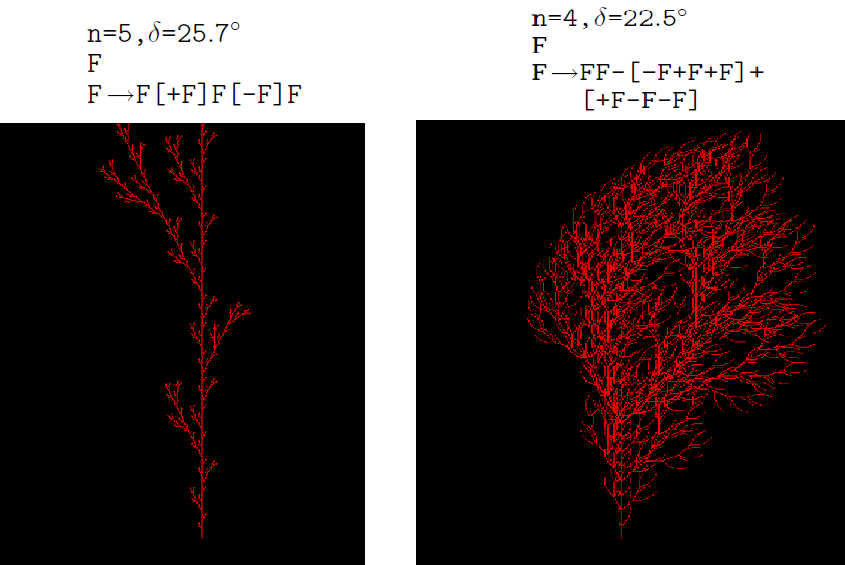


Figura . Progresul aplicației după exemplele din carte

O primă provocare a fost generarea probabilisă. Aceasta constă în a avea mai multe reguli pentru aceeasi cheie(de exemplu, F -> FF cu probabilitatea 0.5 si F -> F[+F][-F] cu probabilitatea 0.5). Primele iterații de forme generate procedural se pot vedea mai jos:

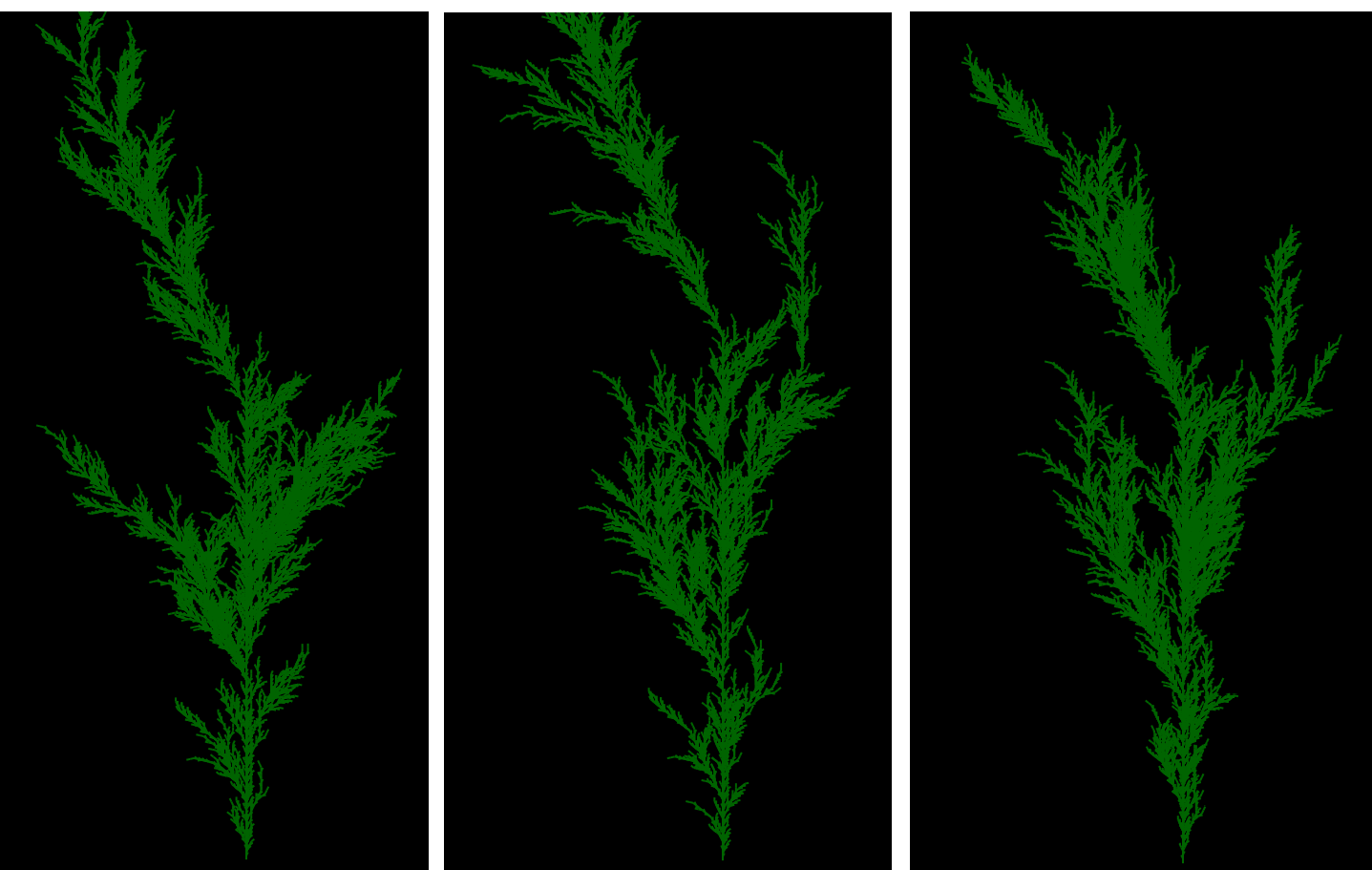


Figura . Progresul aplicației după seturi de reguli proprii

Aceste 3 reprezentări sunt pe baza aceluiași set de reguli. Se poate observa o similitudine destul de mare, dar si diferențe evidente intre cele 3 forme. Densitatea plantelor este din cauza faptului că regulile erau considerabil mai lungi, motivul fiind pentru a se vedea diferentele mai ușor.

* 1. **Primele frunze și flori**

Acum că avem copaci, am vrut să aibă și flori și frunze. Găsirea unor reguli care să facă apariția frunzelor si florilor deterministă s-a dovedit a fi una foarte dificilă, așa că am ales să adopt o metodologie aleatorie, anume câteva din nodurile terminale să fie reprezentate de flori sau frunze.

Am adăugat de asemenea și o secțiune în colțul din stânga, jos, având câteva informații utile, precum numărul setului de reguli la care ne aflăm, timpul necesar ultimei iterații, numărul de iterații executat etc.

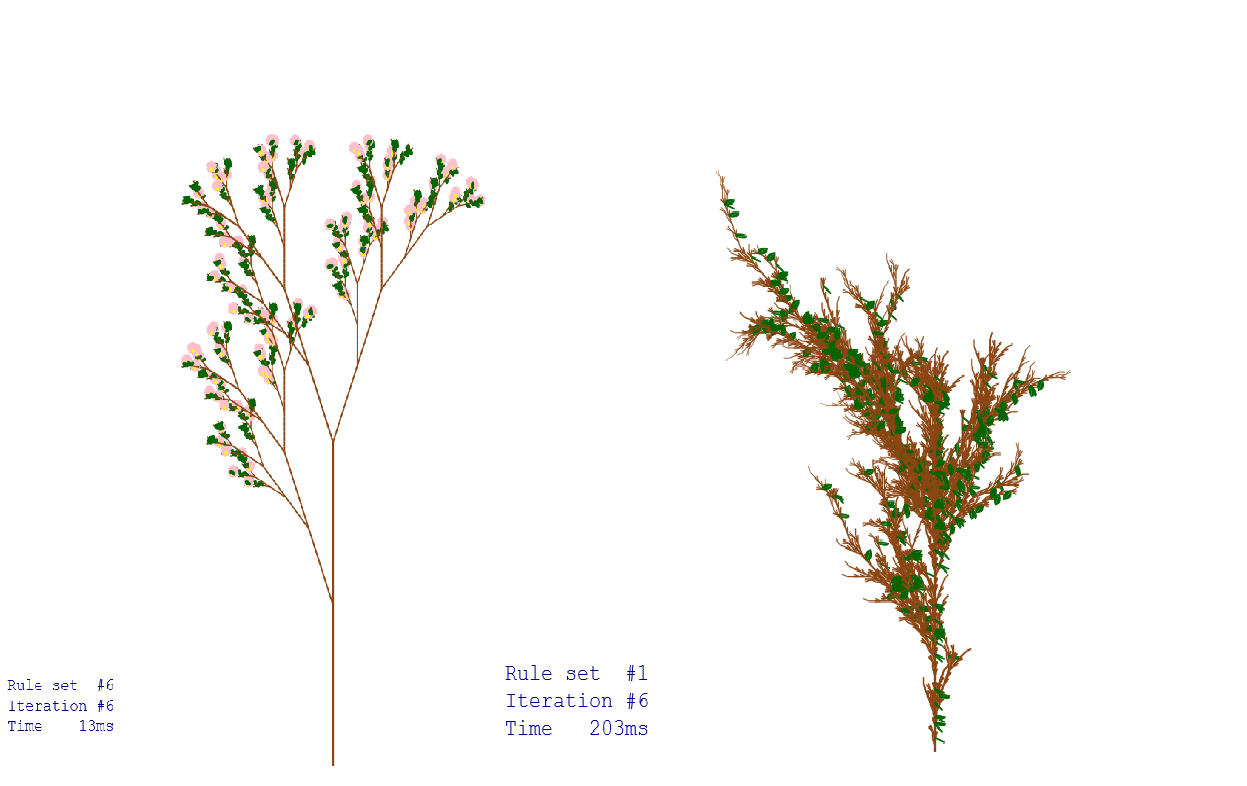


Figura . Progresul aplicației după adăugarea unor frunze și flori

După cum se poate observa, florile nu se pot distinge prea bine, iar frunzele nu au o formă plăcută. Poza din dreapta este făcută înainte să fie implementată logica de floare, motiv pentru care apar doar frunze.

* 1. **Grădina**

Pentru a pune mai multe seturi de reguli în același plan, am implementat simularea unei grădini care așează în mod pseudoaleator forme de vegetație într-un tablou.

Printre provocări, se pot numi:

* găsirea unui echilibru între flori si frunze
* așezarea plantelor
* densitatea plantelor
* găsirea unei palete de culori plăcut vizuală

În continuare, va urma un set de ilustrații prin care se poate vedea progresul și trecerea peste provocările menționate mai sus:

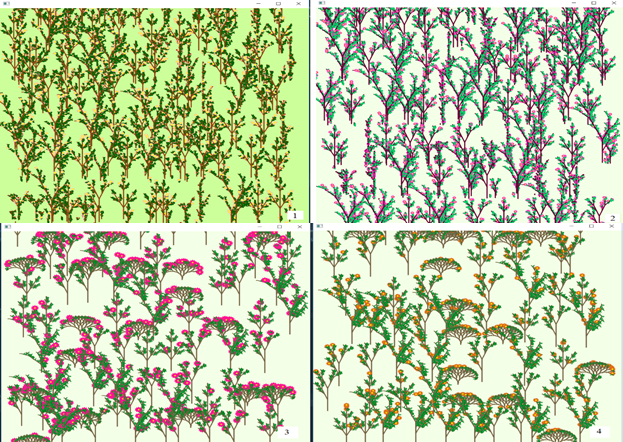


Figura . Progresul grădinii botanice

1. **Interfața grafică**
   1. **Butoane cu scurtături**

După ce am terminat partea vizuală, am implementat și o interfață grafică pentru a le fi de ajutor utilizatorilor.

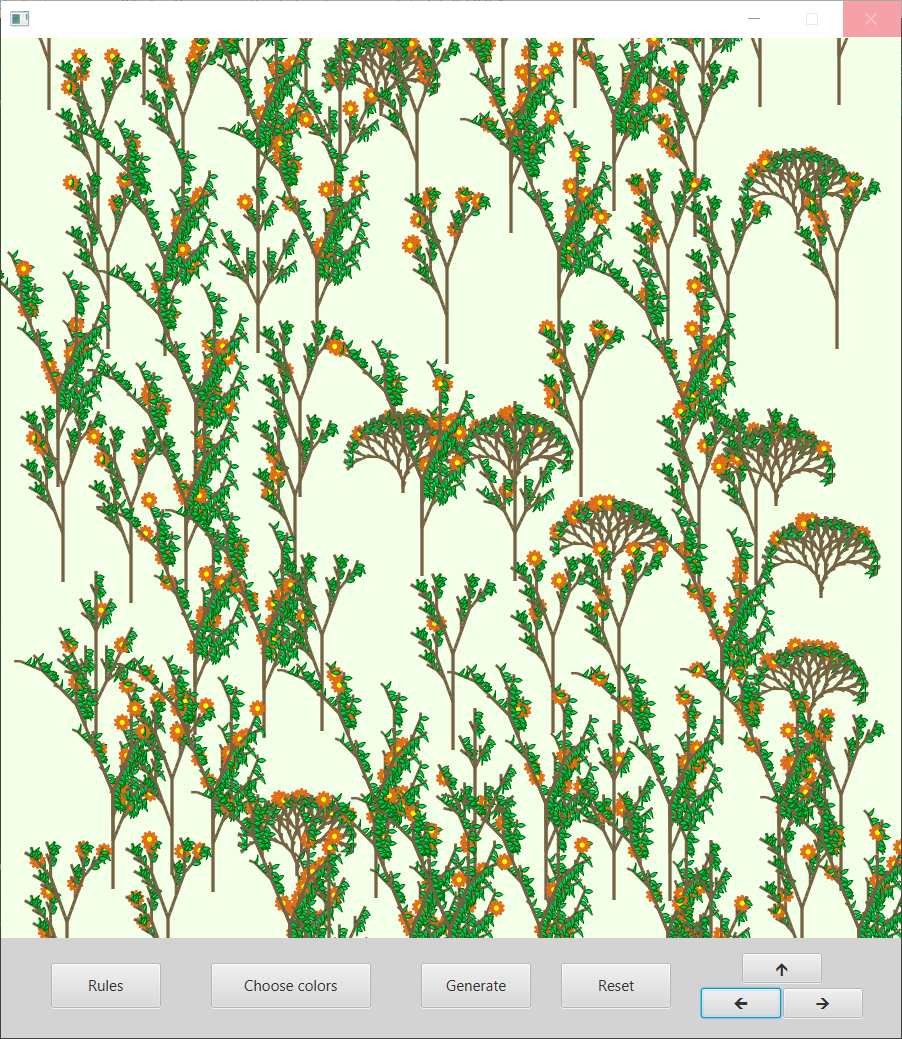


Figura . Progresul după adăugarea interfeței grafice

Nu toate butoanele sunt necesare acestei interfețe grafice, unele comenzi fiind accesibile și prin apăsarea unor butoane de pe tastatură, aceste scurtături fiind vizibile pe eticheta butoanelor respective (dacă se ține cursorul mouseului pe ele).

* Butoanele care conțin săgețile stânga și dreapta (⬅ și ⮕) au rolul de a naviga prin seturile de reguli. Scurtăturile pe tastatură sunt ***A*** și ***D***.
* Butonul cu săgeată in sus (⬆) are rolul de a trece la următoarea iterație a unui set de reguli. Scurtătura este pe tasta ***W***.
* Butonul “Reset” are rolul de a reveni la iterația cu numărul 0, adică axioma setului curent. Scurtătura este pe tasta ***X***.
* Butonul “Generate” este folosit pentru a genera grădina, folosind seturile existente. Scurtătura este pe tasta ***G***.
  1. **Butoane fără scurtături**

Celelalte două butoane nu dispun de scurtături, fiind prea complicat de folosit doar cu ajutorul tastaturii.

Butonul “Choose colors” deschide o fereastră adițională cu ajutorul căreia putem selecta paleta de culori folosită pentru desenare. Putem schimba culorile trunchiului, a florilor, a conturului florilor, a pistilului, a frunzelor și a conturului frunzelor. Aceste culori sunt folosite de la următoarea acțiune, cea curentă rămânând neschimbată.

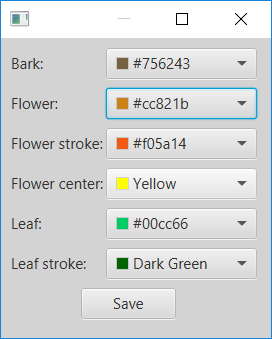


Figura . Fereastra de gestionare a culorilor

Butonul “Rules” deschide și el la rândul lui o fereastră adițională folosită pentru a modifica seturile de reguli.

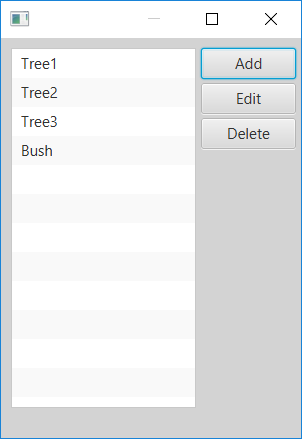


Figura . Fereastra deschisă de butonul **Rules**

Aici dispunem de trei opțiuni:

* **Add –** prin care adăugăm un set nou de reguli după un formular

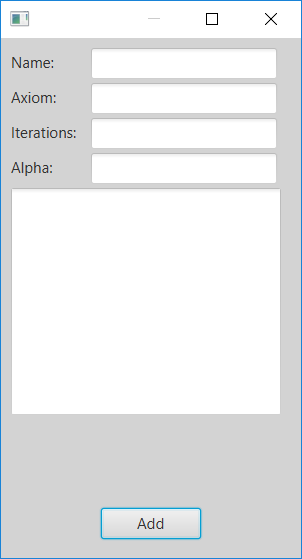


Figura . Fereastra deschisă de butonul **Add**

* **Edit** – în care ni se incarcă un set de reguli selectat din meniu, putând ulterior să îl modificăm după bunul plac

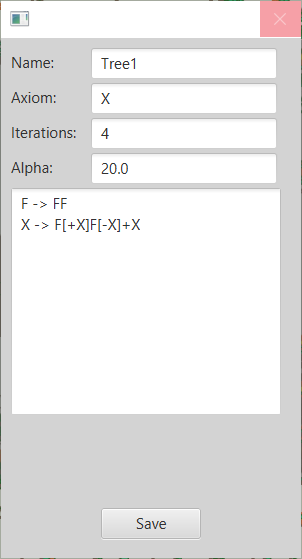


Figura . Fereastra deschisă de butonul **Edit**

* **Delete** – cu ajutorul căruia putem șterge setul de reguli selectat din meniu.

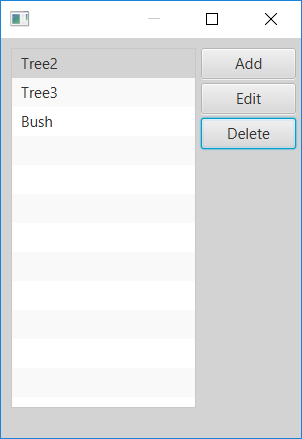
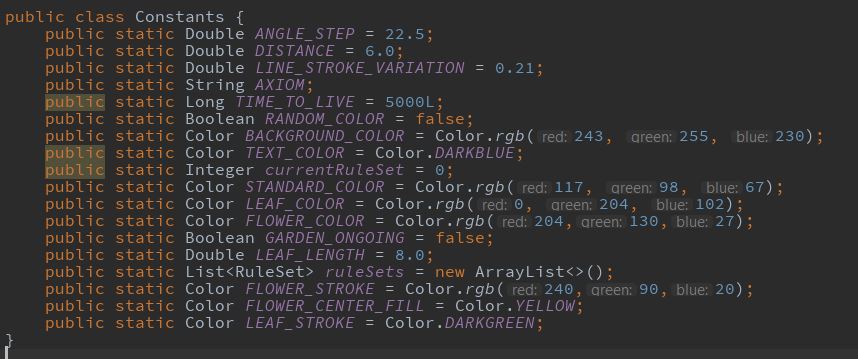


Figura . Fereastra butonului **Rules** după ștergerea elementului **Tree1**

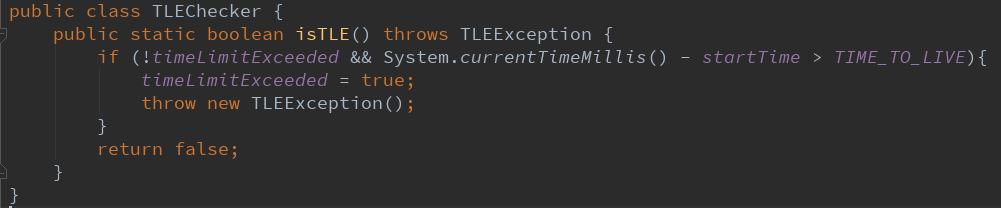
1. **Implementare**
   1. **Metode și clase utilitare**

Despre clasele si metodele utilitare voi menționa trei aspecte pe care le consider mai importante.

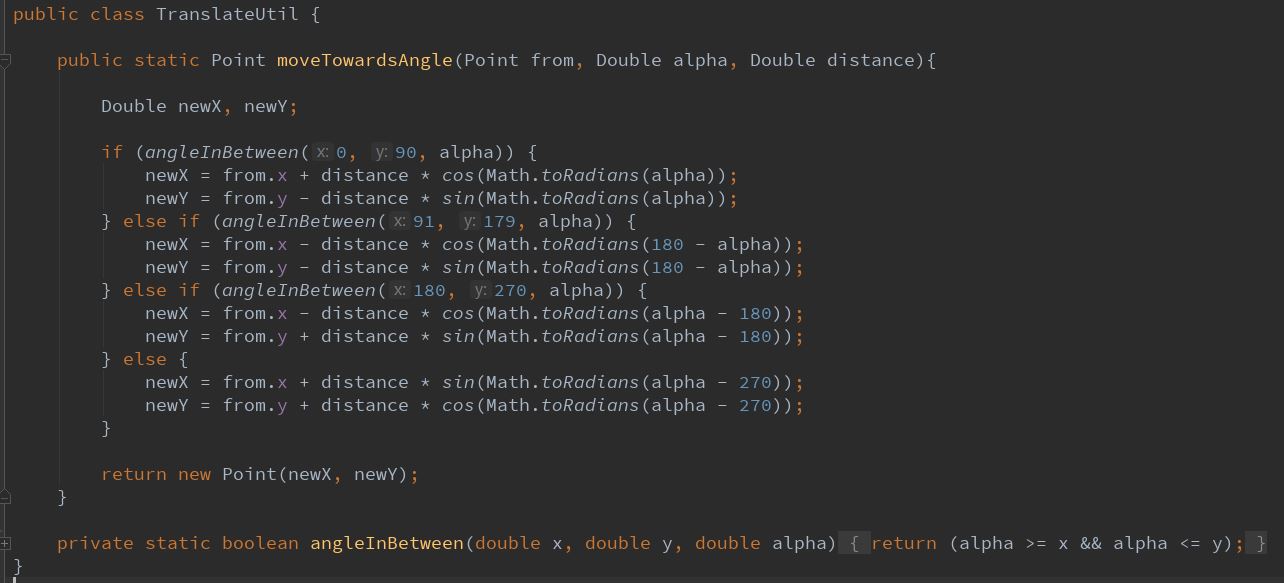
În primul rând, avem clasa *Constants* în care sunt declarate în mod public atributele folosite global. Deși numele clasei este *Constants*, aceste variabile se pot schimba în funcție de setul de reguli folosit sau paleta de culori aleasă. Aceste variabile sunt constante pentru operațiile executate într-un anumit moment.



Pentru a evita generări care ar putea avea durate de ordinul ridicolului, am implementat o excepție personalizată în vederea evitării acestui fapt. Verificarea se face într-o clasă utilitară *TLEChecker* care este apelată în mai multe puncta din aplicație, verificând dacă un timp *TIME\_TO\_LIVE* (pe care îl puteți vedea mai sus declarant, de ordinul milisecundelor) a fost depășit.



Clasa *TranslateUtil* are o singură metodă publică, aceasta deservind rolul de a returna un punct în spațiu dacă se dorește deplasarea unui alt punct cu o anumită distanță, într-o anumită direcție dată de un unghi (alpha), acesta fiind raportat la cercul trigonometric (de exemplu, vertical în sus reprezintă o mișcare cu un unghi de 90 de grade).

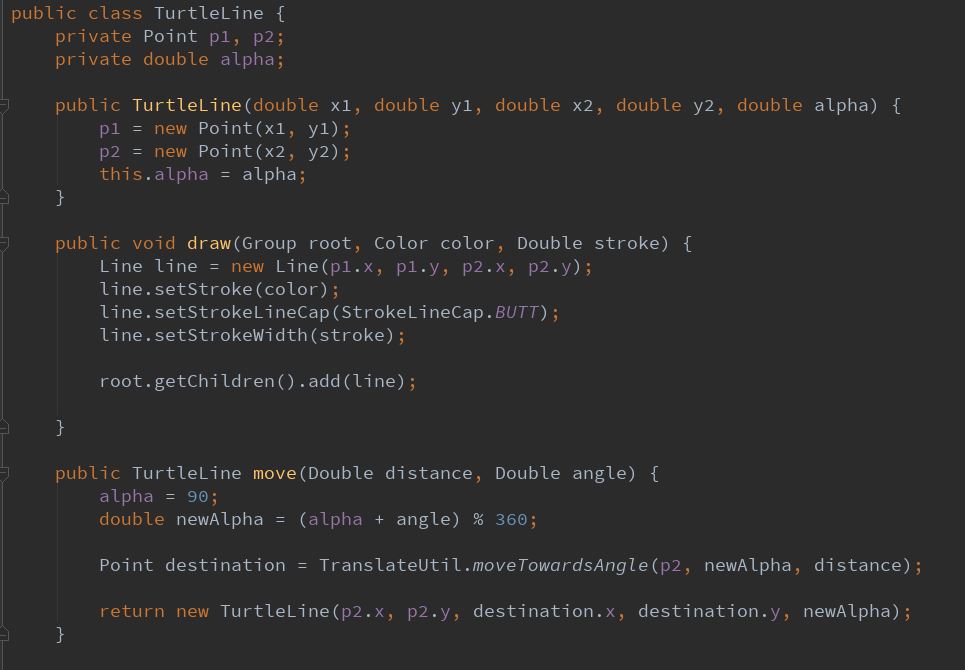


Am tratat fiecare caz pentru fiecare din cele patru cadrane ale cercului trigonometric. Noul punct este calculate cu ajutorul funcțiilor sinus si cosinus, dupa caz, în funcție de cadran, distanță și unghiul respectiv.

Mai dispune de o metodă, privată, care verifică dacă unghiul este într-un anumit interval. Această metodă este folosită strict pentru a îmbunătăți lizibilitatea codului.

* 1. **Componente de bază** 
     1. **Clasa TurtleLine**

Clasa *TurtleLine* este o componentă care are rolul efectiv de a simula mersul unei țestoase, descris în introducere, prin desenarea unei linii care pornește dintr-un punct și se deplasează într-o anumită direcție. Aici este folosită clasa *TranslateUtil* descrisă mai sus. Clasa are două puncte care delimitează un segment și un unghi care reprezintă unghiul format de dreaptă in raport cu cercul trigonometric. Metoda *draw* desenează efectiv linia, cu o anumită culoare, iar metoda *move* simulează mișcarea țestoasei din punctul curent într-o direcție.



* + 1. **Clasa Leaf**

Pentru generarea de frunze, avem clasa *Leaf* care dispune și ea de o metodă *draw* similară cu cea din clasa *TurtleLine*, dar metoda pe care o vom discuta este *generateLeaf* care returnează o listă cu două linii curbe deschise care, unite, ne dau frunza dorită de noi, asemănător cu imaginea de mai jos. Această clasă folosește si ea, la rândul ei, clasa *TranslateUtil*.

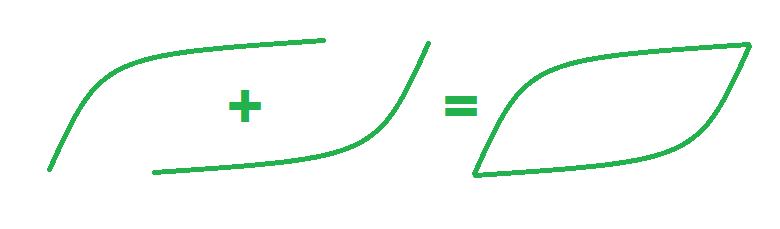
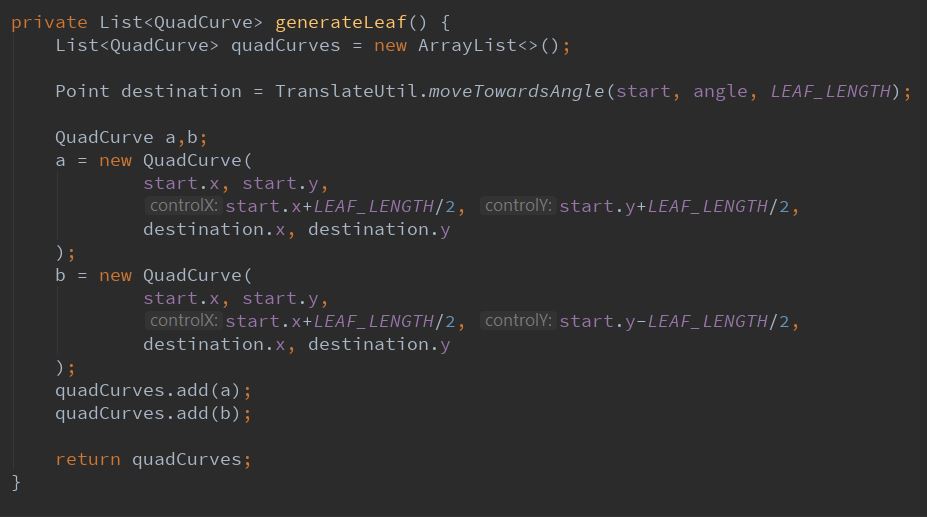


Figura . Construirea unei frunze

Aceasta este o frunză ușor curbată (un unghi de aproximativ 30°).



Se poate spune că desenăm o frunză în jurul unui segment de dreaptă, folosindu-ne de unghiul dreptei în raport cu cercul trigonometric.

* + 1. **Clasa Flower**

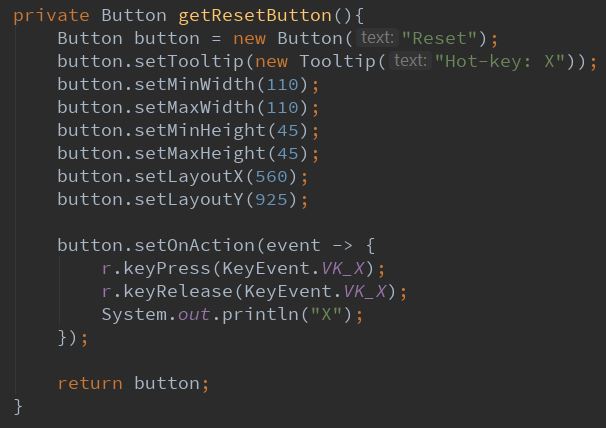
Această clasă se ocupă cu desenarea florilor. Desenarea petalelor este asemănătoare cu cea descrisă pentru o frunză. Logica este că se alege un punct care va fi centrul florii și se desenează N petale egal distanțate în jurul centrului, acesta fiind punctul de start pentru fiecare. În final, se adaugă un cerc colorat în mijloc, simulând stamina acesteia. Desenarea se realizează, din nou, într-o metodă numită *draw* (asemănătoare cu cea din clasa *TurtleLine*), iar generarea petalelor are loc în metoda *generateFlower*, returnând o listă de elipse care, desenate, vor simula petalele dorite.



* + 1. **Generarea unui buton**

Un buton din JavaFX are multe metode ajutătoare, dar pentru noi, contează doar dimensiunile (înălțimea și lățimea), poziția, textul, eticheta (textul ce apare atunci când se ține cursorul deasupra sa pentru câteva secunde) și funcționalitatea.

Butonul **Reset** prezentat mai jos are ca acțiune apăsarea tastei **X**, fiind interpretată de un alt eveniment care tratează acest caz.

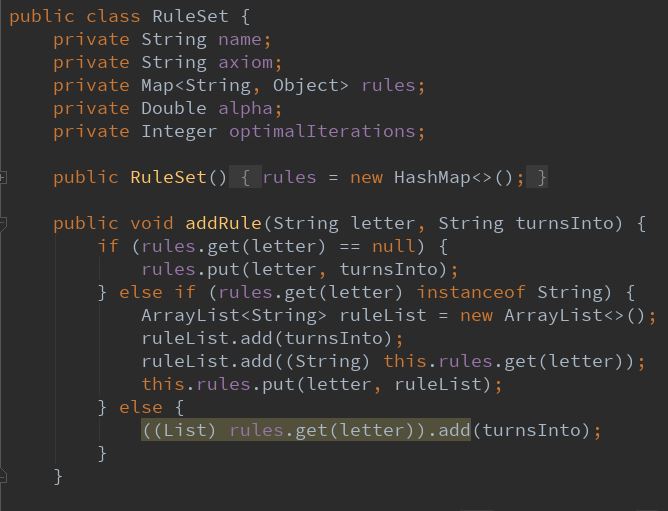


* 1. **Funcționalitatea regulilor**
     1. **Clasa RuleSet**

Această clasă conține variabilele necesare pentru a memora un set de reguli:

* **name** – numele unui set de reguli, necesar pentru menajarea mai ușoară de către utilizator
* **axiom –** punctul de start pentru setul de reguli respective
* **alpha –** unghiul cu care își schimbă direcția “țestoasa” atunci când această comandă i se adresează
* **optimalIterations –** un număr care reprezintă numărul de iterații cu care se dorește a fi afișată planta reprezentată de setul de reguli respectiv când se dorește generarea grădinii
* **rules –** o listă de tip <cheie, valoare> de reguli care definesc setul

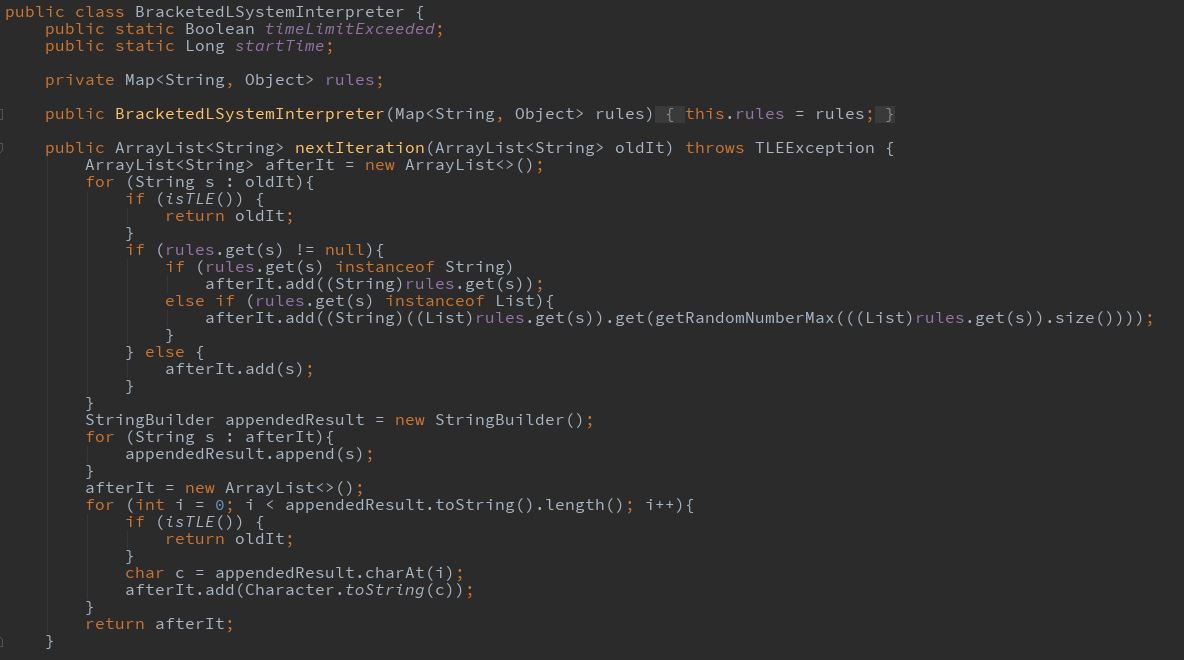
De asemenea, o metodă importantă este cea de a adăuga o regulă nouă în listă, deoarece pot fi mai multe chei. Am ales ca perechile să fie de tipul <cheie, listă\_de\_valori> în aceste cazuri.



* + 1. **Incrementarea unei iterații**

Aceasta se realizează în clasa *BracketedLSystemInterpreter* care are o metodă principală *nextIteration* care primește ca parametru o listă de caractere, constituind iterația precedentă (inițial, evident, aceasta este axioma). Următoarea iterație se obține astfel:

* se parcurge, caracter cu caracter, iterația precedentă
* în cazul în care avem o regulă pentru caracterul respectiv, acesta este înlocuit într-un nou șir cu regula găsită
* în caz contrar, caracterul se copiază în noul șir, acesta putând fi, de exemplu, o paranteză sau un caracter pentru care nu există regulă



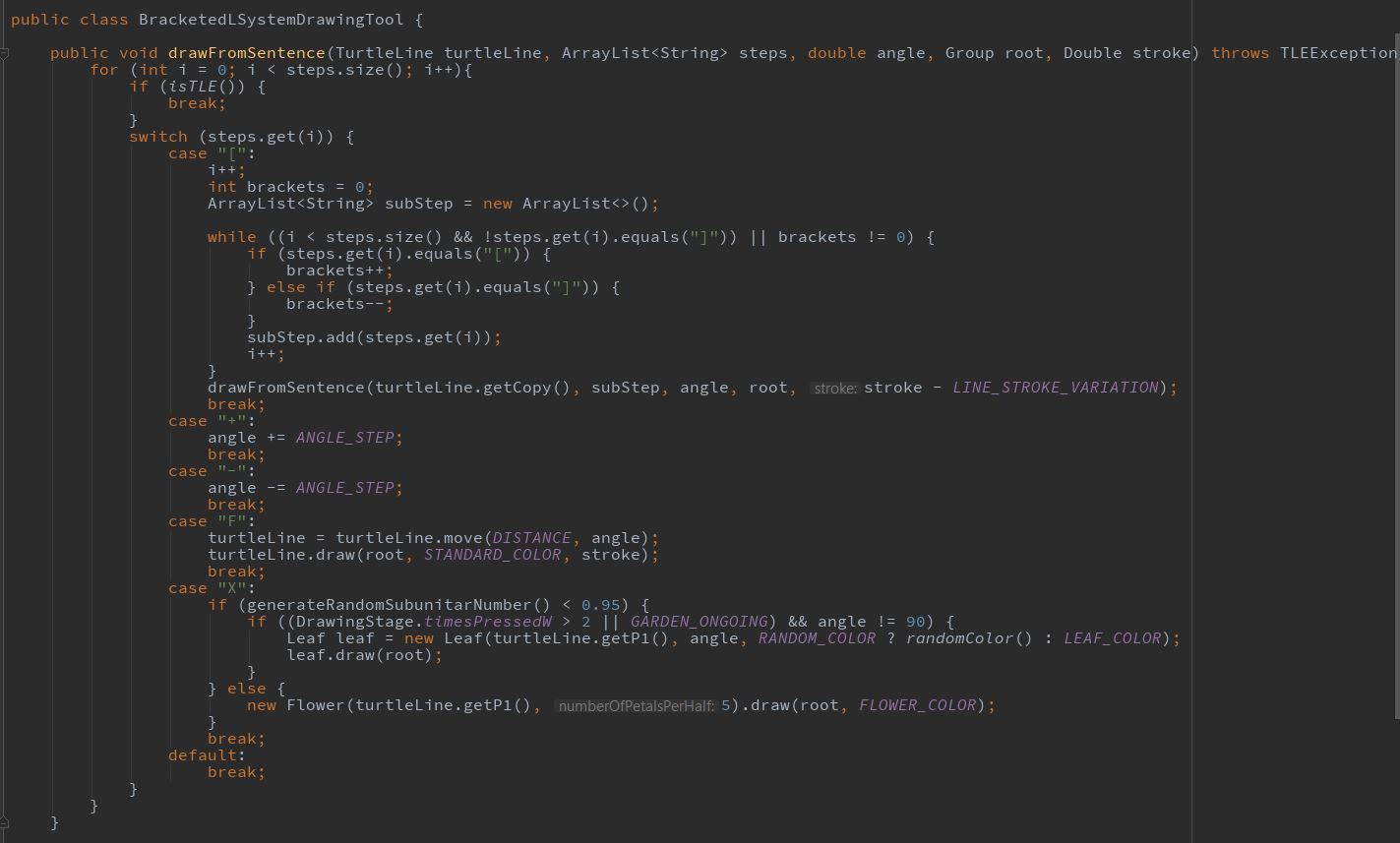
* + 1. **Desenarea unei iterații**

Desenarea se face în clasa cu numele *BracketedLSystemDrawingTool*. Aceasta se face secvențial. Se parcurge caracter cu caracter șirul aflat sub forma unei liste, iar caracterele speciale sunt tratate. Caracterele speciale sunt:

* **[ (paranteză pătrată deschisă)** - acest caracter denotă faptul că, după ce se ajunge la un caracter detipul **] (paranteză pătrată închisă)**, *unghiul* liniei țestoasă revine la valoarea dinaintea **parantezei pătrate deschise**. Acesta s-a dovedit a fi cel mai complicat caracter de interpretat, deoarece a implicat un apel recursiv a funcției până se ajunge la un caracter **]**, moment in care se poate reveni la valoarea inițială a *unghiului*.
* **+ (plus)** – reprezintă incrementarea *unghiului* cu valoarea dorită (valoare ce se găsește în orice set de reguli sub denumirea de **alpha**)
* **- (minus)** – reprezintă decrementarea *unghiului*, similar cu simbolul **+ (plus)**
* **F** – acest caracter denotă comanda dată “țestoasei” pentru merge înainte
* **X –** cazul în care se dorește a fi afișată o frunză (probabilitate 95%) sau o floare (probabilitate 5%)
* **Niciunul din aceste caractere –** nu se întâmplă nimic, iar caracterul respectiv este ignorat

Clasa mai dispune de o metodă *generateRandomSubunitarNumber()* care generează un număr cuprins în intervalul [0, 1) necesar deciderii dacă un caracter de tipul **X** va fi floare sau frunză.

Se poate vedea cum au fost tratate aceste cazuri în poza următoare:

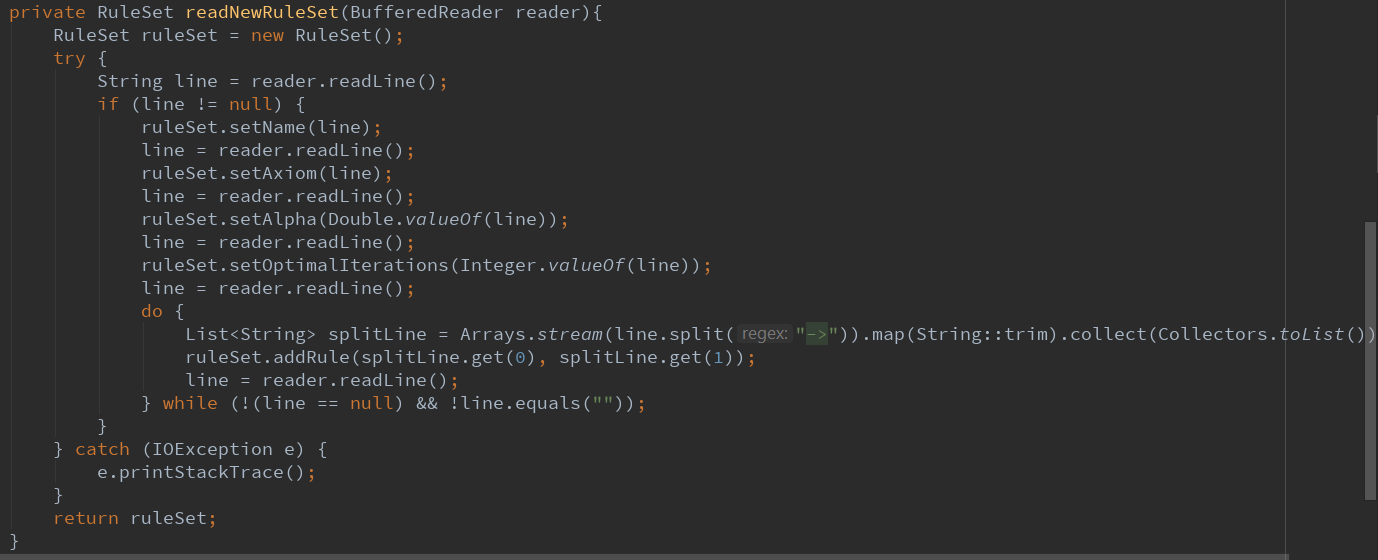


* 1. **Decursul aplicației**
     1. **Citirea seturilor predefinite de reguli**

Aceste reguli sunt citite dintr-un fișier text denumit *predefined-rule-sets.txt,* despre care am menționat puțin în introducere, unde se află, structurate, informații necesare incărcării seturilor de reguli. Această citire se face la începutul rulării cu ajutorul metodei *readPredefinedRuleSets* prezentată mai jos.

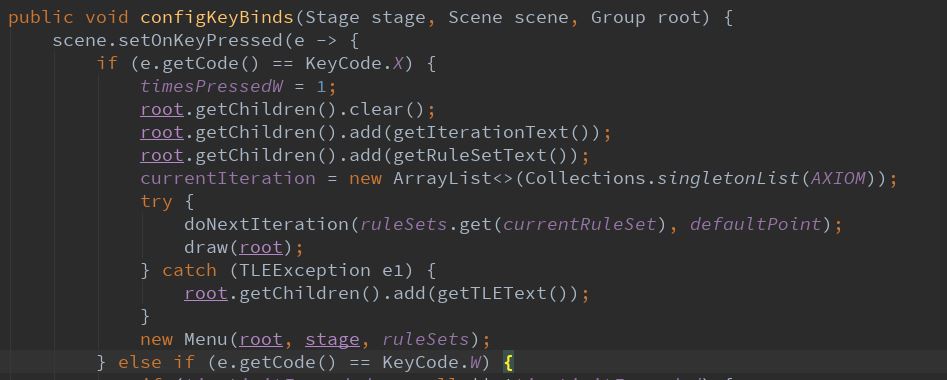


Această metodă încarcă pe rând câte un set de reguli cu ajutorul metodei *readNewRuleSet* care citește linie cu linie din fișierul de reguli, așteptându-se la un anumit format. Numărul de reguli, putând fi nelimitat, este marcat de o linie care nu conține nimic, făcându-se astfel separarea între seturi.



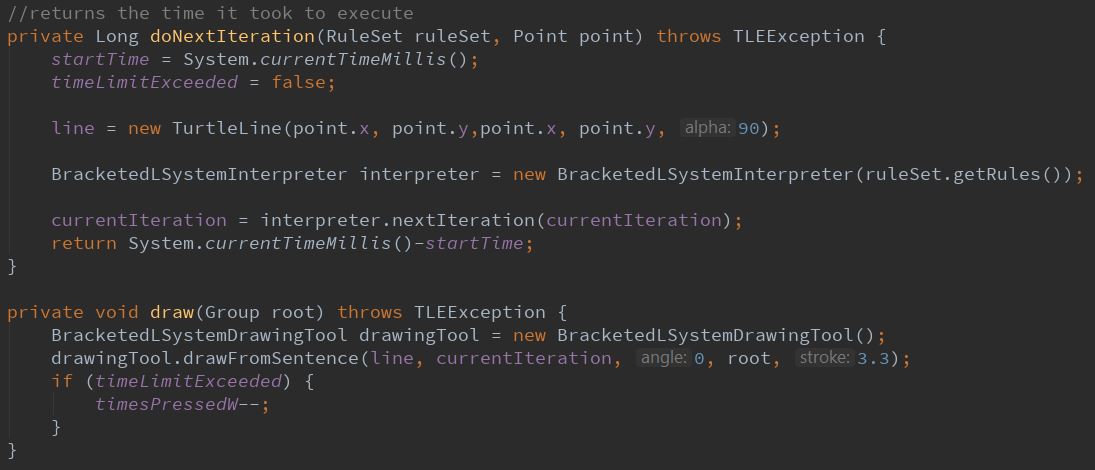
* + 1. **Configurarea butoanelor**

Sunt unele butoane cărora li s-au atașat anumite acțiuni atunci când sunt apăsate. Este vorba de butoanele având rol de scurtătură pentru o altă modalitate de a folosi anumite funcționalități. Metoda *configKeyBinds* aflată în clasa *DrawingStage* este responsabilă pentru aceste scurtături. Putem vedea implementarea pentru scurtătura atașată butonului **Reset**, anume tasta **X**, în imaginea de mai jos.



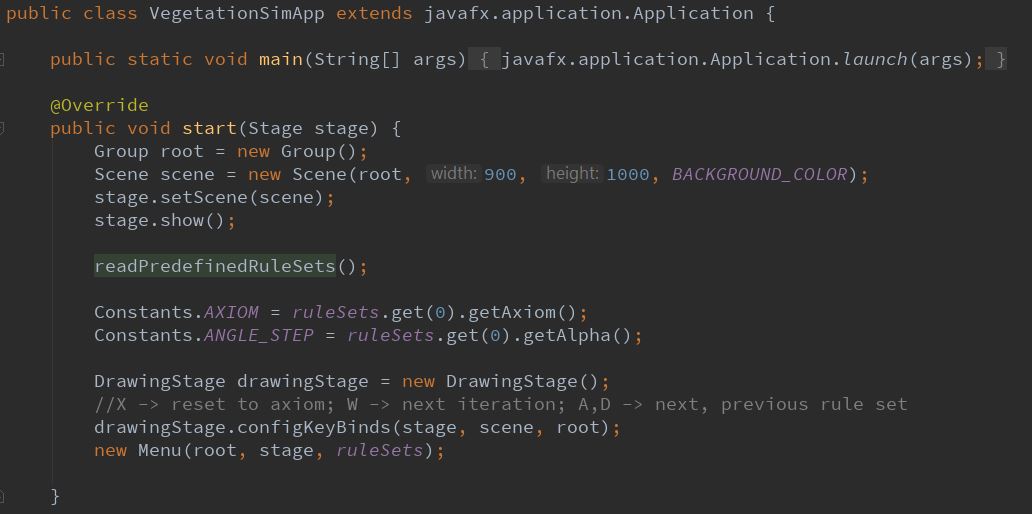
Această metodă șterge totul din lista de obiecte afișate, urmând după să adauge noile texte pentru iterația curentă (care este setată cu axioma), iar după se desenează această axiomă și se adaugă meniul aflat în partea de jos a aplicației.

Metoda *doNextIteration* folosește clasa *BracketedLSystemInterpreter* pentru a obține următoarea iterație a șirului nostru. Aceasta returnează un număr care reprezintă timpul necesar obținerii următoarei iterații exprimat in milisecunde. Metoda *draw* folosește clasa *BracketedLSystemDrawingTool* pentru a desena un anumit șir, exprimând o iterație.



* + 1. **Funcția main**

Aplicația se rulează apelând funcția *launch* din JavaFX în funcția statică *main*. Această funcție apelează, printer altele, și metoda *start* pe care trebuie să o implementăm obliatoriu. În metoda *start* se creează prima fereastră cu aplicația în sine. Odată cu închiderea acestei ferestre, se termină și rularea aplicației. De asemenea, aici citim seturile predefinite de reguli, setăm setul de reguli aflat în focus, configurăm butoanele cu scurtături și adăugăm meniul.



1. **Exemple proprii**

După terminarea implementării, am decis să experimentez cu regulile pentru a obține propriile forme vegetative. După nu foarte mult timp, am ajuns la câteva exemple pe care le puteți vedea in continuare:

1. Nume - Copac

Axiomă - X

Alpha - 22.5

Reguli:

* F -> F[+F[-X]]F[-F][+F[X][+X]]F
* F -> F[-F]F[X][-X]F[F[-X]]
* F -> F[+F][-F[-X]][+F]F
* X -> F



Figura . Setul de reguli **Copac** generat în 3 reprize separate

1. Nume – Arbust

Axioma – X

Alpha – 25.7

Reguli:

* F -> F[+F]F[-F][F]
* F -> F[+F]F[-F]F
* F -> [X][F][X][F][X]
* X -> F[+X]F[-X]+X

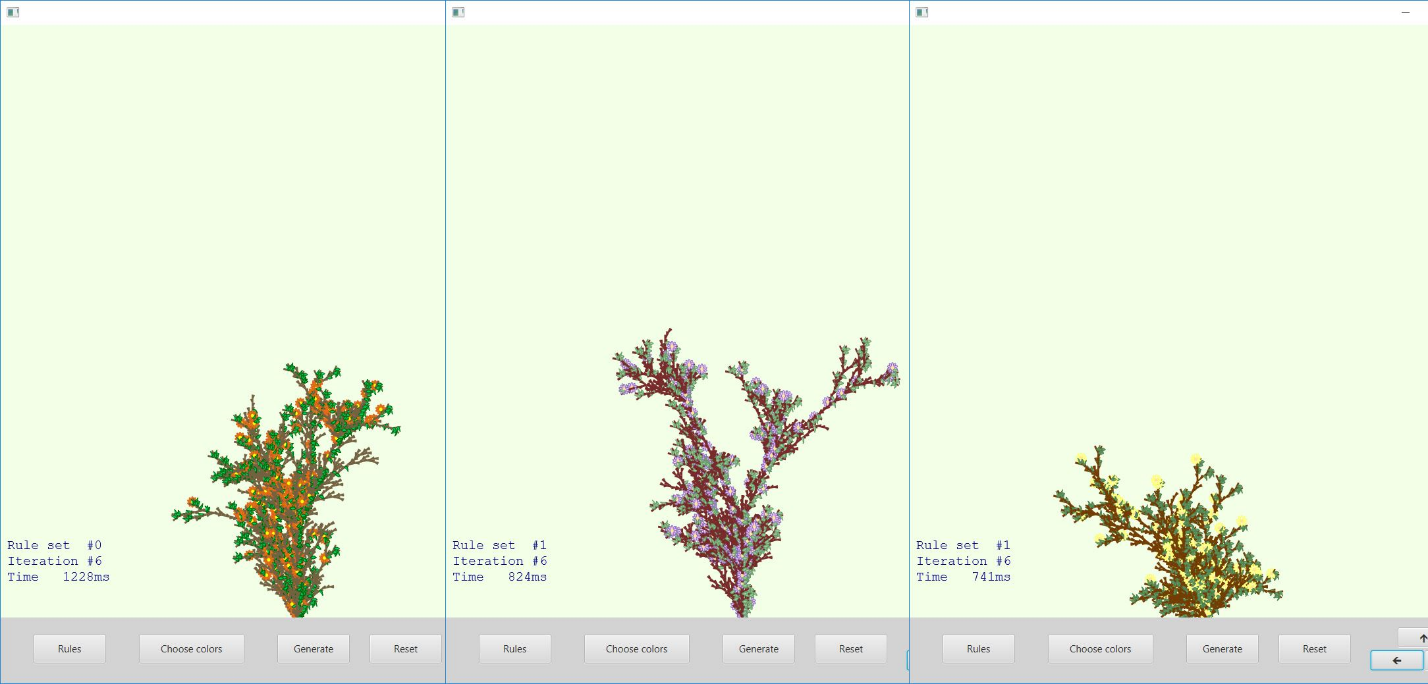


Figura . Setul de reguli **Arbust** generat în 3 reprize separate

Se poate observa o diferență destul de mare dată de schimbarea culorilor folosite.

1. Nume – Tufiș

Axiomă – X

Alpha – 22.5

Reguli:

* F -> F[+F]F[-F][F]
* F -> FF-[-FXF+FXF]+[+FXF-FXF]
* F -> [X][F][X][F][X]
* X -> F[+X]F[-X]+X
* X -> F

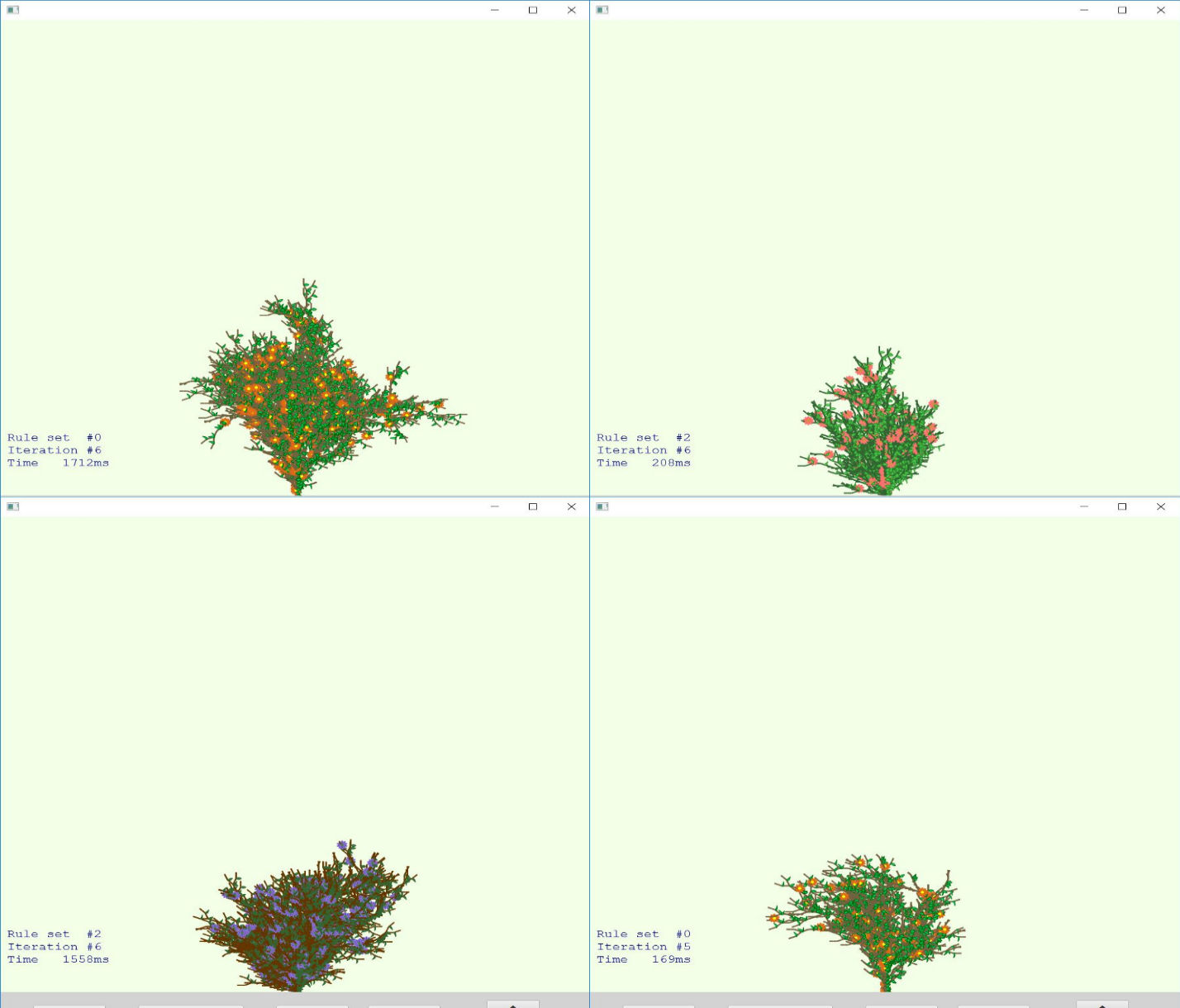


Figura . Setul de reguli Tufiș generat în 4 reprize separate

1. **Concluzii**

Aplicația Vegetation-Sim, prezentată în această lucrare, este o aplicație care îndeplinește mai multe scopuri cum ar fi educativ sau de divertisment. Ideea aplicației a originat din mediul înconjurător împreună cu studiul fractalilor în cadrul universității.

Printre provocări se pot număra reprezentarea unei linii turtle, gestionarea parantezelor imbricate și desenarea florilor.

Ceea ce am adăugat pe parcurs și nu era plănuit este gestionarea culorilor folosite și informațiile utile din coltul stânga jos atunci când se iterează prin plantele existente.

Consider că am reușit să îndeplinesc o bună parte din ceea ce îmi doream în legătură cu acest proiect, reușind să simulez cu succes forme vegetative având varietate vizibilă.

1. **Direcții de dezvoltare**

Aplicația poate fi imbunătățită pe mai multe planuri, cum ar fi introducerea unei baze de date centralizate și conectarea aplicației la un server. Acest procedeu ar implica și un procedeu de logare, iar datele utilizatorului ar fi păstrate într-o bază de date, putând fi refolosite cu ușurință.

O altă funcționalitate pe care mi-am dorit să o implementez inițial dar nu am avut timp este o variantă 3D a acestui proiect, având posibilitatea de a roti plantele și a le privi din orice direcție.

Alte posibile îmbunătățiri, mai minore, ar fi:

* Schimbarea culorii fundalului aplicației
* Adăugarea mai multor tipuri de frunze și flori
* Posibilitatea schimbării probabilității de generare între flori și frunze din interfață
* Îmbunătățiri pe partea așezării plantelor din grădina botanică
* Grosimea tulpinei/trunchiului să se reducă progresiv de la bază spre vârf

1. **Bibliografie**

[1] Przemyslaw Prusinkiewicz, Aristid Lindenmayer. *The Algorithmic Beauty*

*of Plants*, 2004

[2] “Curs practic de Java”, Cristian Frăsinaru de pe: <https://profs.info.uaic.ro/~acf/java/Cristian_Frasinaru-Curs_practic_de_Java.pdf>

[3] “JavaFX Tutorial”, de pe: <https://www.tutorialspoint.com/javafx/index.htm>

[4] “L Systems”, de pe: <https://en.wikipedia.org/wiki/L-system>

[5] “List of trigonometric identities”, de pe: [https://en.wikipedia.org/wiki/List\_of\_trigonometric\_identities](https://en.wikipedia.org/wiki/List_of_trigonometric_identities%20)

1. https://stackify.com/popular-programming-languages-2018/ [↑](#footnote-ref-1)
2. https://ro.wikipedia.org/wiki/E-learning [↑](#footnote-ref-2)
3. https://en.wikipedia.org/wiki/L-system [↑](#footnote-ref-3)
4. https://ro.wikipedia.org/wiki/Java\_(limbaj\_de\_programare) [↑](#footnote-ref-4)
5. http://net-informations.com/java/intro/jvm.htm [↑](#footnote-ref-5)
6. https://en.wikipedia.org/wiki/JavaFX [↑](#footnote-ref-6)
7. https://en.wikipedia.org/wiki/JavaFX#Example\_code [↑](#footnote-ref-7)
8. https://en.wikipedia.org/wiki/IntelliJ\_IDEA [↑](#footnote-ref-8)