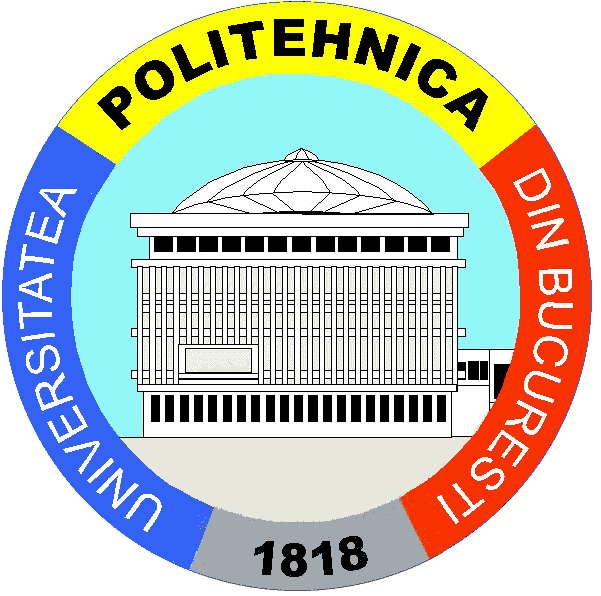
Universitatea POLITEHNICA din Bucuresti, Romania

Facultatea de Automatica si Calculatoare



VAN GOGU’

-Software Design Document -

Echipa :

Balutoiu Diana 342 C5

Mihalache Andreea Codrina 342C5

Iacob Mihai Alexandru 342C5

Stoica Eugen 342C5

Marinescu Robert 341C5

Ungureanu Cristian 342C4

**CUPRINS**

**1.Introducere**

1.1.Scopul documentului

1.2.Obiective

1.3.Organizarea documentului

1.4.Definitii

1.5.Descrierea proiectului

**2.Arhitectura propusa**

2.1.Tehnologii folosite in dezvoltare

2.2.Descrierea algoritmului ce realizeaza imaginea

2.2.1. Descrierea modului de realizare a unui element

2.2.2. Descrierea modelului fractalic implementat

2.2.3. Factori aleatori utilizati

2.3 Conditii limita

**3.Avantaje ale arhitecturii**

**4.Elemente de testare**

**5.Dezvoltare ulterioara**

1. **INTRODUCERE**
   1. **Scopul documentului**

Scopul acestui document este de a prezenta modul de implementare al acestui joc, dar si modalitatea de functionare. De asemenea, descrie detaliat structura componentelor folosite.

* 1. **Obiective**

Descrierea arhitecturala prezentata in acest document are urmatoarele roluri:

* Descrierea limbajelor ce vor fi folosite in implementare
* Descrierea structurii functionale si a algoritmilor implementati
* Faciliteaza definirea si indeplinirea cerintelor
* Ofera suport in posibile activitati de mentenanta
  1. **Organizarea documentului**

Documentul este structurat in trei capitole, dupa cum urmeaza:

* Capitolul 1: ofera o introducere si defineste conceptele folosite in implementare
* Capitolul 2: prezinta arhitectura propusa si modul in care sunt generate imaginile
* Capitolul 3: prezinta avantaje ale folosirii acestei arhitecturi
* Capitolul 4: prezinta modalitatea de testare
* Capitolul 5 : prezinta planurile de viitor ale echipei CrispyTeam cu privire la “Van Gogu”
  1. **Definitii**
* **Scheme/Racket** – limbaj de programare functional tipat dinamic si ce permite folosirea de “side-effects”. In centrul limbajului se afla evaluarea de functii. In scheme, orice constructie este de fapt o functie, inclusiv expresiile conditionale (if). Nu este un limbaj de programare functional “pur”, permitand folosirea si mai ales modificarea valorilor variabilelor globale (side-effect).
* **Recursivitate** - În matematică și informatică, recursivitatea sau recursia este un mod de a defini unele funcții. Funcția este recursivă, dacă definiția ei folosește o referire la ea însăși, creând la prima vedere un cerc vicios, care însă este numai aparent, nu și real.
* **Programare functionala** – paradigmă de programare care tratează calculul ca evaluare de funcții matematice și evită starea și datele muabile. Se pune accent pe aplicarea de funcții, spre deosebire de programarea imperativă, care folosește în principal de schimbările de stare. Modelul matematic al programării funcționale îl reprezintă calculul lambda. Limbajele funcționale moderne pot fi considerate extensii ale calculului lambda.[1] Noțiunea de bază în această paradigmă este cea de funcțională sau funcție de nivel înalt, o funcție care poate accepta ca argument sau returna ca valoare o altă funcție.
* **Fractal** - figură geometrică fragmentată sau frântă care poate fi divizată în părți, astfel încât fiecare dintre acestea să fie (cel puțin aproximativ) o copie miniaturală a întregului. Are o structură fină la scări arbitrar de mici. Are o structură fină la scări arbitrar de mici.

**1.5.Descrierea proiectului**

Scopul proiectului a fost de a crea o aplicatie care sa genereze o imagine cu o valoare artistica cat mai ridicata, utilizand doar geometrie fractalica si algoritmica simpla. De asemenea, la fiecare rulare a aplicatiei, ar trebui sa se genereze o imagine diferita, fiind folosit un factor aleator.

Pentru realizarea proiectului am ales sa generam o imagine abstracta, formata din cercuri si linii. Aceasta se genereaza recursiv – 3 niveluri de recursivitate. Factorul aleator consta in numarul de linii, distanta dintre ele, dimensiune precum si culorile utilizate. Imaginile generate seamana cu niste fulgi de nea multicolori.

**2.Arhitectura propusa**

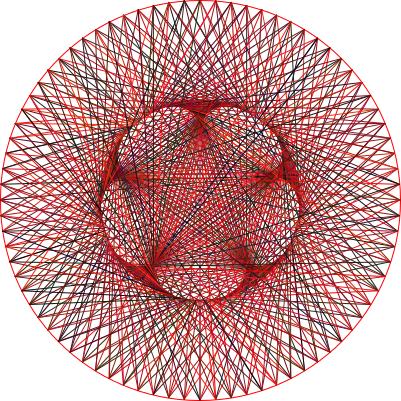
**2.1.Tehnologii folosite**

Deoarce pentru a crea fractali cel mai usor se foloseste recursivitate, iar functiile recursive se definesc foarte usor si rapid folosind programare functionala, am ales sa implementam proiectul in Scheme. Acesta este un limbaj de programare functional, in care fiecare expresie definita este practic o functie. Algoritmii recursivi se scriu foarte usor si rapid in acest limbaj. De asemenea are functii de desenat foarte usor de folosit, accesibile doar incluzand o librarie, spre deosebire de alte libaje de programare.

**2.2.Descrierea algoritmului ce realizeaza imaginea**

**2.2.1. Descrierea modului de realizare a unui element**

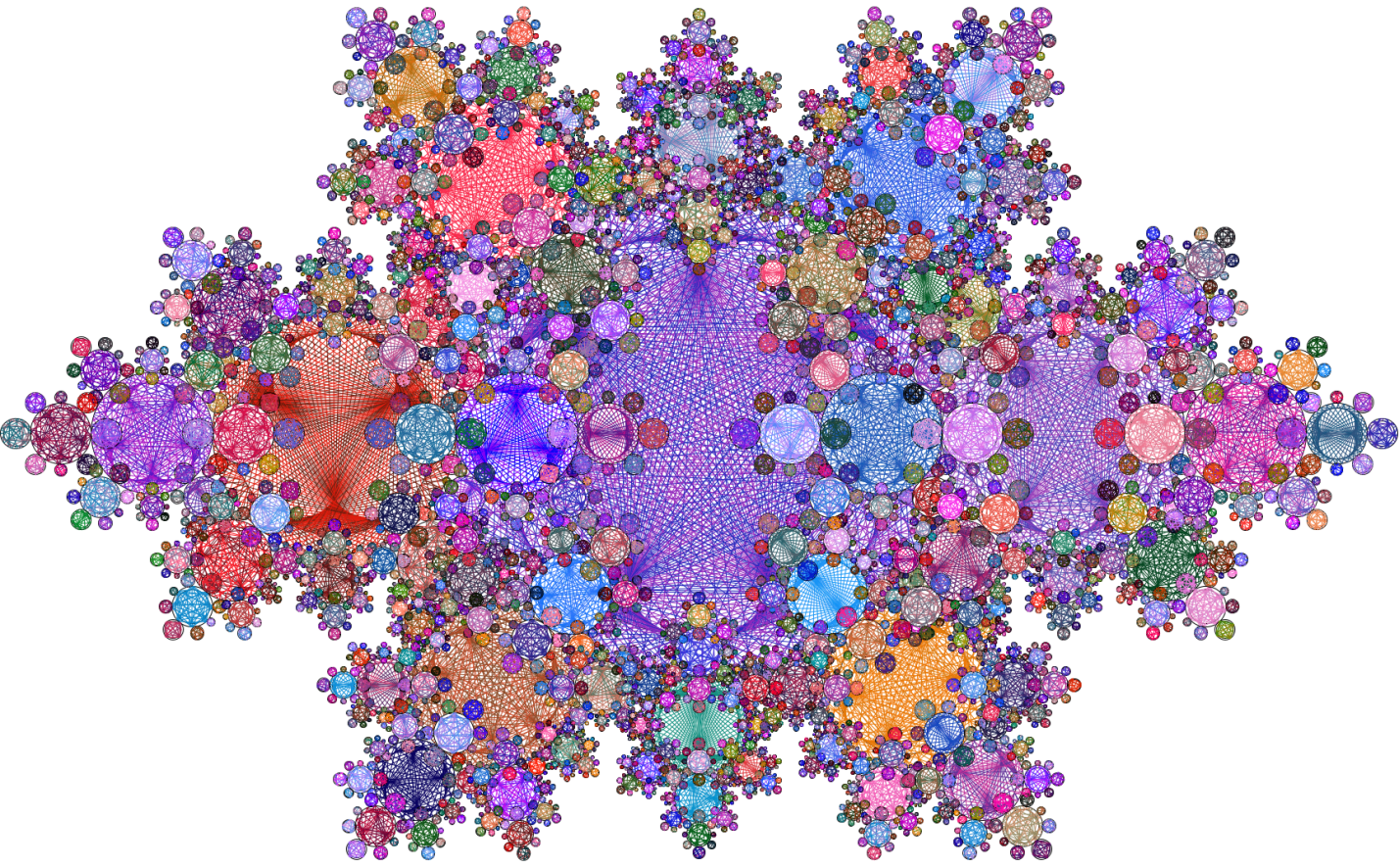
Elementul de baza utilizat pentru a genera imaginea este format din doua cercuri concentrice. Intre punctele de pe aceste doua cercuri sunt trasate linii, in functie de un unghi prestabilit, formandu-se diverse imagini abstracte ce se aseamana cu niste flori. Acest model este cel de baza, utilizat in realizarea imaginii finale, reprezentand elementul initial al modelului fractalic, ce va fi multiplicata si scalata, formand rezultatul final.



O sa denumim acest tip de element geometric in continuare cerc de baza.

**2.2.2. Descrierea modelului fractalic implementat**

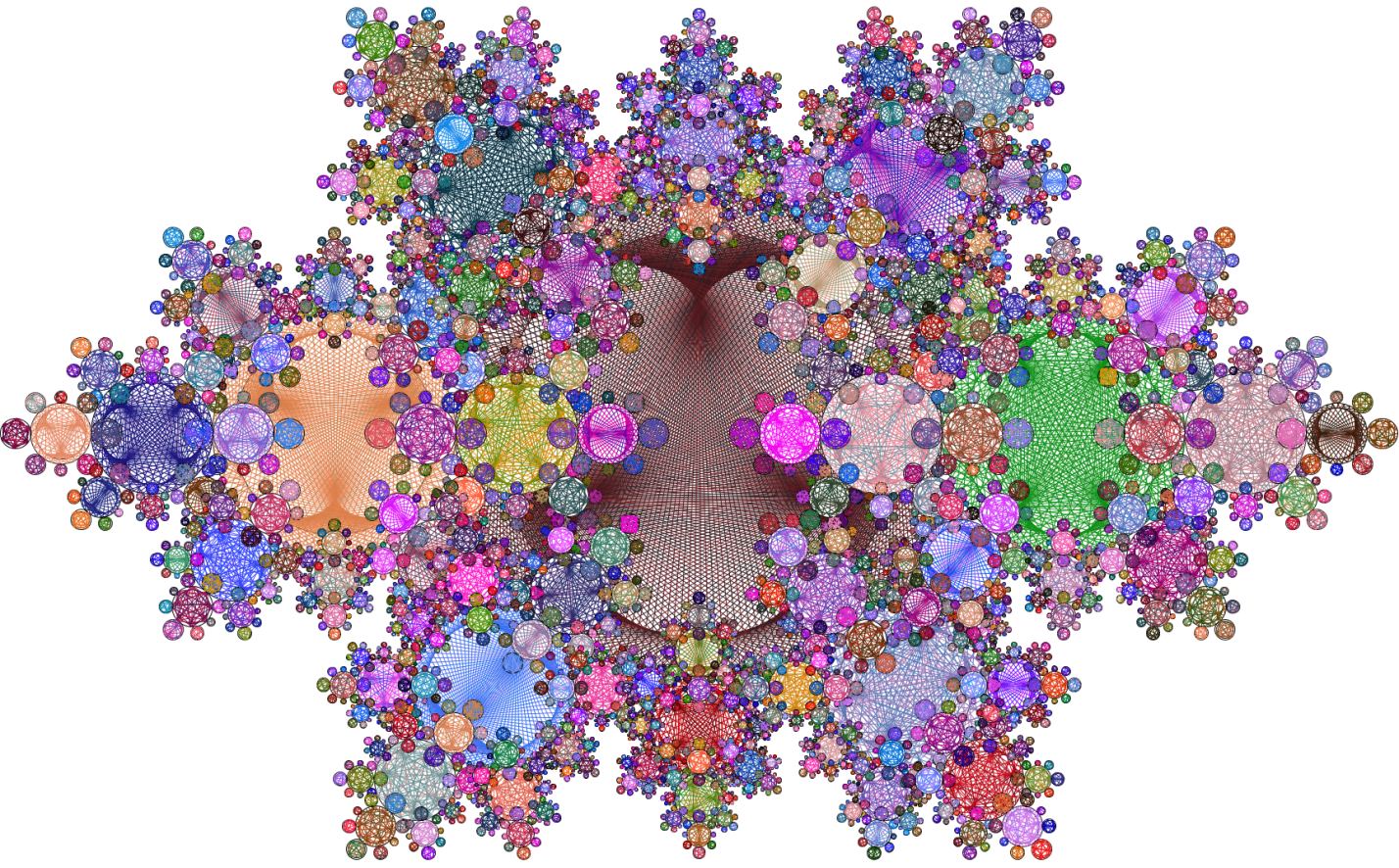
Definitia unui model geometric fractalic este accea ca este formata din aceeasi structura geometrica complexa, repetata. La baza imaginii generate de aceasta aplicatie se afla cercul de baza definit mai sus. In centrul imaginii se afla un cerc de baza mare. Pe marginile acestuia, la unghiuri de 0, 50, 90, 130, 180, 230, 270 si 310 (privind cercul de baza ca pe un cerc trigonometric) se deseneaza alte cercuri. Procedeul se repeta recursiv pe aceste cercuri, recursivitatea avand nivel 4. Pentru un nivel mai mare, timpul de executie creste exponential, si nu se mai incadreaza in limita de 1 minut impusa de cerinta

****

**2.2.3. Factori aleatori utilizati**

Pentru a genera imagini unice la fiecare rulare, am introdus anumiti factori aleatori:

* Culorile interiorului cercurilor, dar si ale exteriorului acestora se genereaza aleator, dintr-o anumita paleta cromatica bazata in principal pe culori reci (albastru, mov).
* Unghiul dintre liniile trasate in interiorul cercurilor de baza este aleator pentru fiecare cerc in parte.



**2.3.Conditii limita**

Nu exista conditii limita. La fiecare rulare se va genera o imagine.

**3.Avantaje ale arhitecturii**

- Adaptabilitate - codul este foarte usor de customizat si de modificat. Se pot modifica usor dimensiunile cercurilor, culorile si pozitiile, precum si nivelul de recursivitate – daca acesta este crescut, se produc imagini mai spetaculoase, timpul de rulare fiind insa mai mare.

- Originalitate – ideea este proprie acestei echipe, nu este “imprumutata”, fiind doar produsul imaginatiei noastre.

- Simplitate – codul rezultat este foarte scurt si simplu, avand doar 75 de linii de cod.

**4. Elemente de testare**

Aplicatia nu a putut fi testata in mod traditional, deoarece nu exista usecase-uri sau un rezultat final “corect” ce trebuie obtinut. Procesul de testare a constat efectiv in rularea aplicatiei de mai multe ori, analizarea imaginilor rezultate, si efectuarea mai multor modificari asupra codului pentru a genera imagini mai frumoase si mai spectaculoase, pana s-a obtinut produsul final. Insa acest produs final este subiectiv.

**5.Dezvoltare ulterioara**

Pe viitor echipa CrispyTeam doreste sa se joace in continuare cu acesta aplicatie, inlaturand insa limita de timp. Am observat ca cu cat creste gradul de recursivitate mai mult, cu atat se obtin imagini mai spectaculoase. De asemenea am dori sa mai lucram la partea de cromatica, sa definim mai multe palete de culori ce se potrivesc din care sa fie alese culorile folosite pentru imagine.