Colegiul Național “Vasile Alecsandri” Galați

Proiect pentru susţinerea atestatului profesional la informatică

**Rubik's cube**

Filieră: teoretică

Profil: real

Specializare: matematică-informatică

Autor: Cojocaru Teodora Laura

Clasa: 12E

Coordonator: Stan Mihaela

**mai 2023**

**Cuprins**

[1. Descrierea temei și motivația alegerii ei 2](#_Toc125397766)

[**2.** **Cerințe de sistem** 2](#_Toc125397767)

[**2.1** **Cerințe hardware** 2](#_Toc125397768)

[2.2. **Cerințe software** 2](#_Toc125397769)

[**3.** **Limbajul Java** 2](#_Toc125397770)

[**3.1**  **Prezentare generala** 2](#_Toc125397771)

[**3.2** **Programul utilivat si relatia dintre acesta si limbajul de programare** 3](#_Toc125397772)

[**4** **Grafica in aceasta aplicatie** 3](#_Toc125397773)

[5 Prezentarea în ansamblu a aplicației 3](#_Toc125397774)

[6. Detalii de implementare 5](#_Toc125397775)

[7. Surse de documentare online 11](#_Toc125397776)

8[. Anexe 13](#_Toc125397777)

# Descrierea temei și motivația alegerii ei

Cubul rubik este un puzzle 3D cu care fiecare copil a incercat sa se joace macar o data. Inventat in 1974 de un sculptor din Ungaria, a castigat premiul pentru „jocul german al anului”. Cu toate ca a fost inventat cu mult timp in urma, rezolvarea sa aduce problema oamenilor pana in prezent. Odata cu trecerea timpului au fost construite diferite variatiuni ale cubului ce au la baza diferiti algoritmi de rezolvare.

Cubul original are 6 fete impartite in cate 9 patrate, iar fiecare fata are o culoare diferita dintre: alb, rosu, albastru, portocaliu, verde si galben. Dupa ce cobul e amestecat, scopul e revenirea la pozitia initiala, fiecare fata avand o singura ciloare, intr-un timp cat mai scurt cu putinta.

Proiectul meu se adreseaza oricui isi doreste sa isi puna la incercare rabdarea, atentia, spiritul de observatie, dar si gandirea logica, algoritmica in rezolvarea unui puzzle captivant, potrivit pentru toate varstele.

1. **Cerințe de sistem**
   1. **Cerințe hardware**

- Procesor la 1.6GHz sau mai rapid

- 1 GB (32-biti) sau 2 GB (64-biti) memorie RAM (se adauga 512 MB daca se ruleaza dintr-o masina virtuala)

- 3GB spatiu disponibil pe hard disk

- Hard disk cu o viteza de rotatie a platanelor de minim 5400 RPM

- Placa video capabila DirectX 9, ruland la o rezolutie de 1024 x 768 pixeli sau mai ridicata

- Unitate optica DVD-ROM

## 2.2. **Cerințe software**

- Sistem de operare : cel putin Windows XP/7/8

- Microsoft .NET Framework 4.5

1. **Limbajul Java**

Pentru implementarea aplicatiei am utilizat limbajul Java.

## **3.1 Prezentare generala**

Java este unul dintre cele mai populare limbaje de programare la nivel mondial. A fost creat de James Gosling și Patrick Naughton, angajați ai Sun Microsystems, cu sprijinul lui Bill Joy, co-fondatorul Sun Microsystems.

Java este descris ca fiind un limbaj multifuncțional de programare orientată pe obiecte (OOP). Prin proiectare, are cât mai puține dependențe de implementare. Cu acest limbaj de programare, putem crea aplicații pe mai multe dispozitive. Domeniul său de aplicare este foarte larg, permițându-ne să creăm software pentru dispozitive mobile, terminale de puncte de vânzare, bancomate, IoT (Internet of Things), precum și pagini web.

Deși Kotlin este limba preferată pentru dezvoltarea Android în 2021, Java este încă limba implicită pentru dezvoltarea de aplicații pentru Android. Principalul motiv pentru aceasta este integrarea strânsă a Android cu Java. De fapt, părți din acel sistem de operare, cum ar fi interfața de utilizare și bibliotecile de bază specifice, sunt scrise în Java.

## **Programul utilizat si relatia dintre acesta si limbajul de programare**

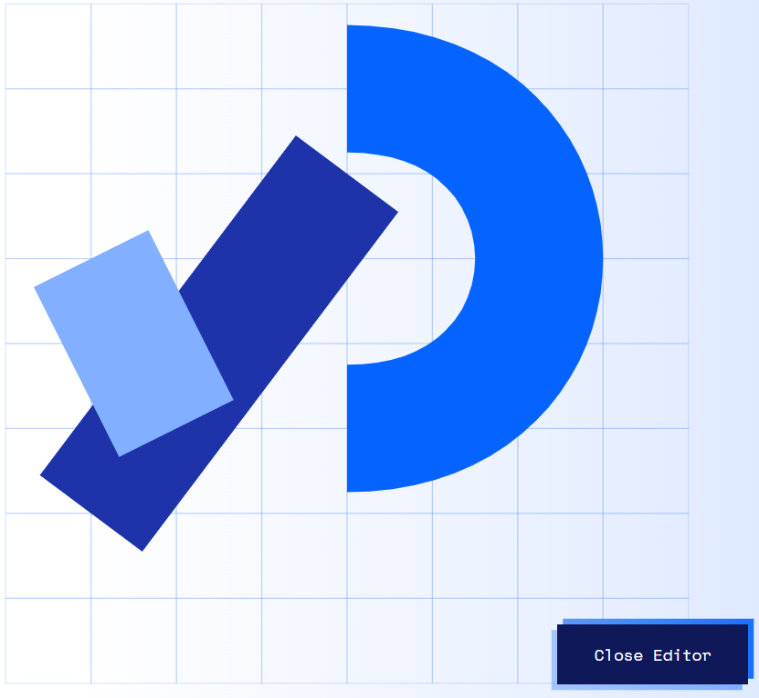
Processing este un mediu de programare simplu care a fost creat pentru a facilita dezvoltarea aplicațiilor orientate vizual, cu accent pe animație și oferind utilizatorilor feedback instantaneu prin interacțiune. Dezvoltatorii au vrut un mijloc de a „schița” ideile în cod. Pe măsură ce capacitățile sale s-au extins în ultimul deceniu, Processing a ajuns să fie folosit pentru lucrări mai avansate la nivel de producție, pe lângă rolul său de schiță. Construit inițial ca o extensie specifică domeniului pentru Java, destinată artiștilor și designerilor, Processing a evoluat într-un instrument complet de proiectare și prototipare utilizat pentru lucrări de instalare la scară largă, grafică în mișcare și vizualizare complexă a datelor.

Processing se bazează pe Java, dar pentru că elementele programului din Processing sunt destul de simple, puteți învăța să-l utilizați chiar dacă nu cunoașteți Java. Dacă sunteți familiarizat cu Java, cel mai bine este să uitați că Processing are ceva de-a face cu Java pentru o perioadă, până când înțelegeți cum funcționează API-ul.

Processing constă în:

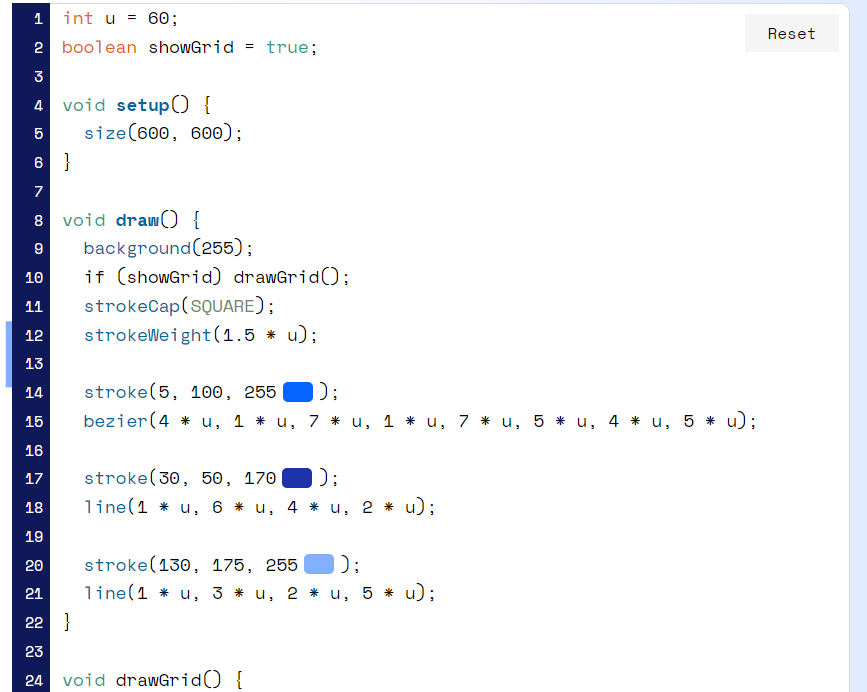
* Mediul de dezvoltare a procesării (PDE). Acesta este software-ul care rulează când faceți dublu clic pe pictograma Processig. PDE este un mediu de dezvoltare integrat (IDE) cu un set minimalist de caracteristici concepute ca o simplă introducere în programare sau pentru testarea ideilor unice.
* colecție de funcții (numite și comenzi sau metode) care alcătuiesc interfața de programare „de bază” sau API, precum și mai multe biblioteci care acceptă funcții mai avansate, cum ar fi trimiterea de date printr-o rețea, citirea imaginilor live de la o cameră web și salvarea imaginilor complexe în format PDF.
* sintaxă a limbajului, identică cu Java, dar cu câteva modificări.

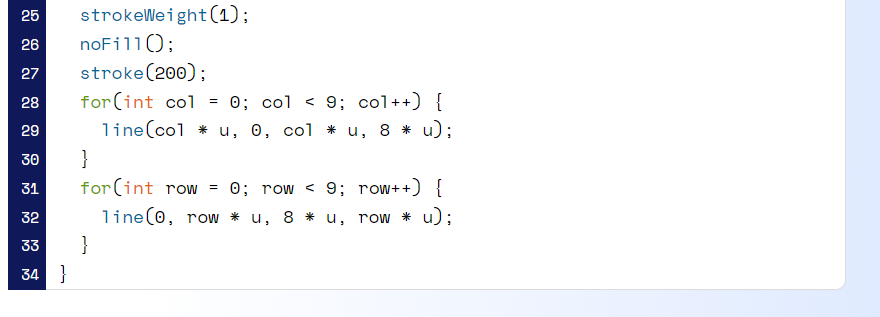
1. **Grafica in aceasta aplicatie**



Processing este un mediu de programare flexibil si usor de utilizat care incurajeaza invatarea programarii, folosind elemente preponderent vizuale, imbinand in acelasi program arta si tehnologia.

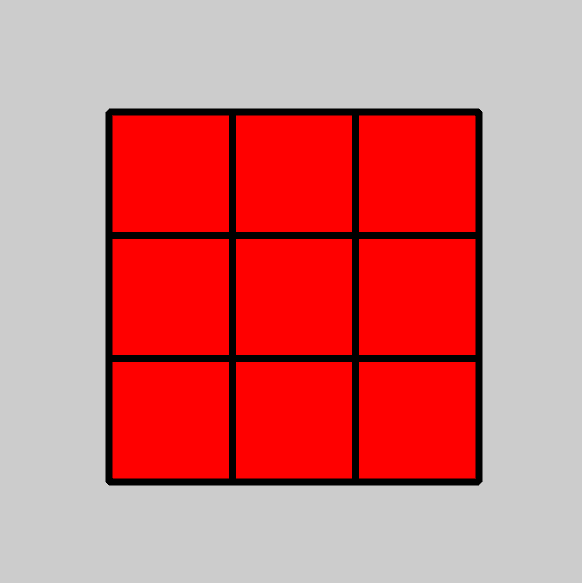
In partea dreapta este atasat logoul aplicicatiei, uramat, pe pagina urmatoare, de codul din spatele acestuia.



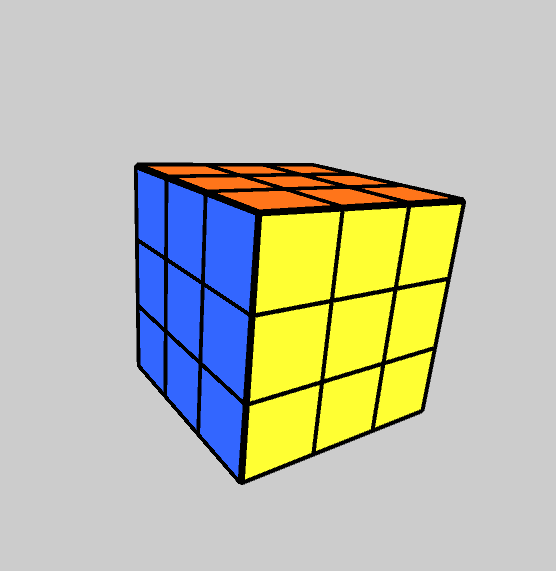
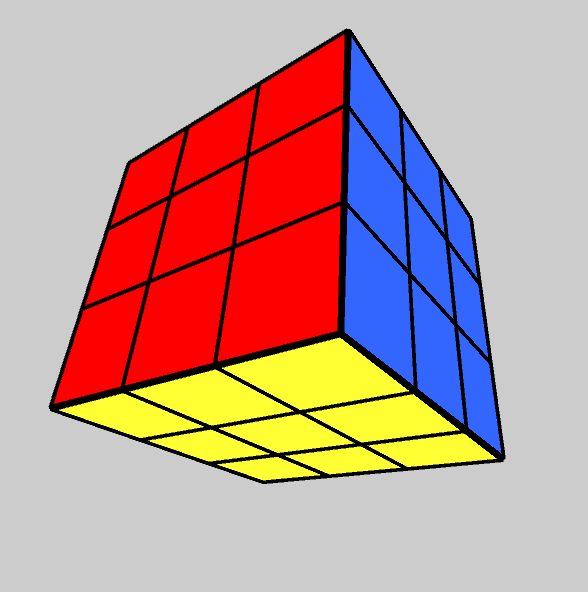


# Prezentarea în ansamblu a aplicației

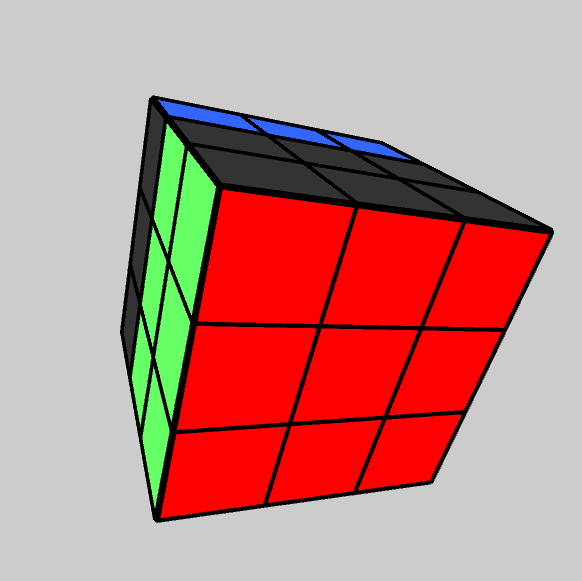
Jocul pe care l-am realizat in Java cu ajutorul Processing se intitulează Rubik’s Cube. Acesta contine un simulator digital al unui cub rubik, ce se comporta la fel ca puzzle-ul real, putand fi amestecat si rezolvat.



Aplicatia se deschide cu ceea ce pare sa fie imaginea unui patrat rosu impartit in 9 fete. Totusi, cu o simpla miscare a mouse-ului se poate repede oberva ca patratul este doar o fata a unui obiect 3D, un cub. Acesta are 6 fete colorate diferit si fiecare este impartita la randul ei in alte 9 patrate de dimensiuni mai mici.



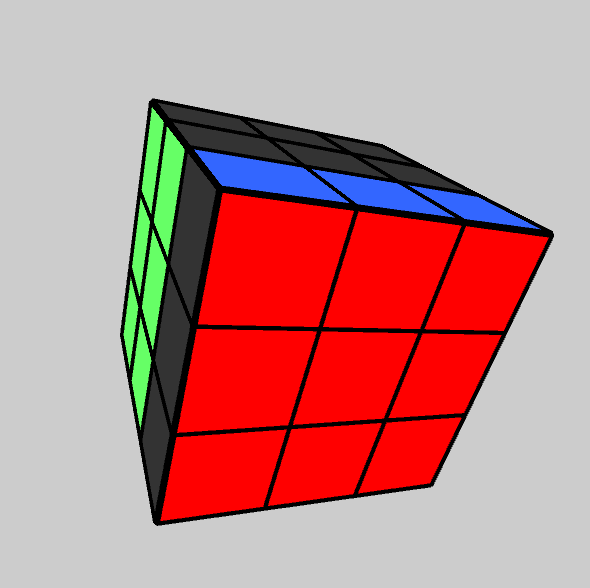
Pentru controlarea punctului de vedere se foloseste mouse ul, iar pentru miscarea efiectiva a felelor cubului se folosesc testele w,e,s,d,x,c sau aceleasi taste impreuna cu tasta shift pentru a obtine miscarea inversa. Astfel, cubul se folseste dupa cum urmeaza:



Tasta w

Muta spre stanga fata din spate a cubului

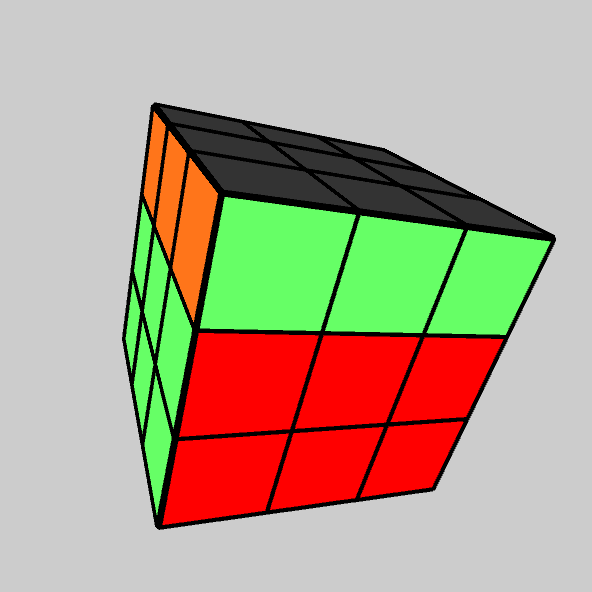
(Tasta w+ shift va readuce cubul in pozitia initiala)



Tasta e

Muta spre stanga partea din fata a cubului

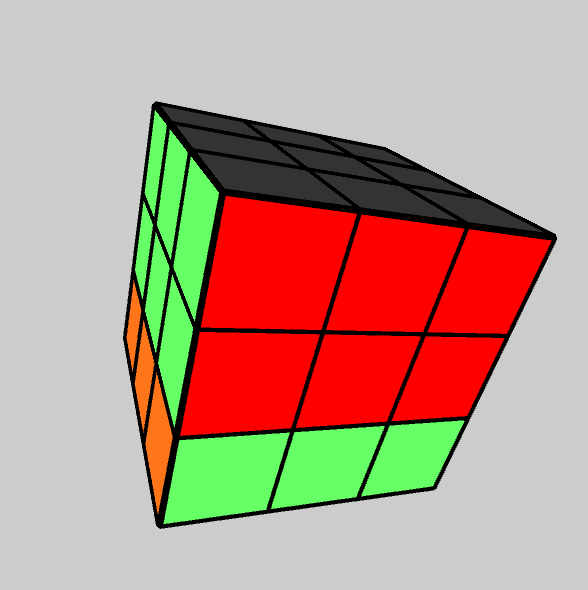
(Tasta e+ shift va readuce cubul in pozitia initiala)



Tasta s

Muta spre dreapta fata de sus a cubului

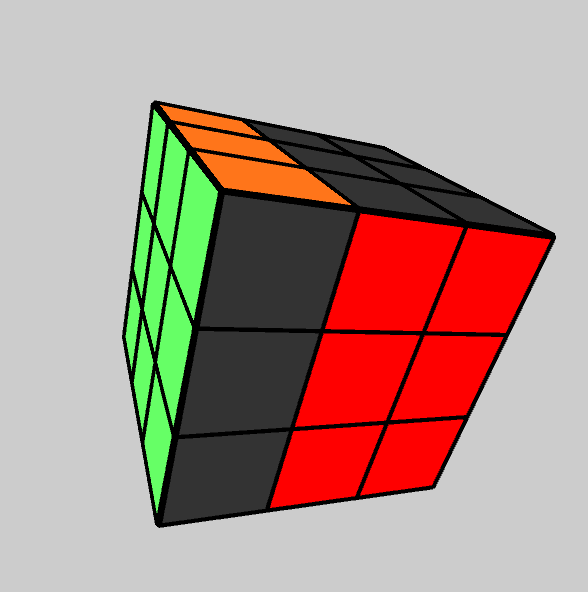
(Tasta s+ shift va readuce cubul in pozitia initiala)



Tasta d

Muta spre dreapta fata de jos a cubului

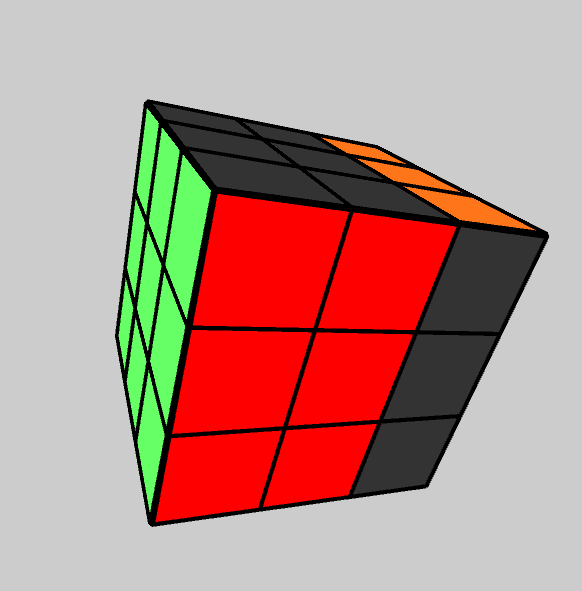
(Tasta d+ shift va readuce cubul in pozitia initiala)



Tasta x

Muta in jos fata din stanga a cubului

(Tasta x+ shift va readuce cubul in pozitia initiala)



Tasta c

Muta in jos fata din dreapta a cubului

(Tasta c+ shift va readuce cubul in pozitia initiala)

# 

Odata ce utilizatorul se obisnuieste cu tastele poate ajunge foarte usor la modele unice precum cel din stanga. Structura din stanga se foarmeaza apasand tastele: w w e e s s d d x x c c

# 6. Detalii de implementare

1. Pentru colorarea fetelor s a folosit urmatorul vector de valori ce repreznta culorile fetelor traduse

color[] colors ={

#333333 , #ffff33 ,

#3366ff , #66ff66 ,

#ff0000 , #ff751a

};

1. Cubul este reprezentat intr un sistem de coordonate cu 3 directii: x, y, z. Intregul cub rubik este gandit ca si cum ar fi fost alcatuit din 3x3x3 cuburi mai mici, cu toate ca cel din mijloc nu este folosit. Astfel, fiecarui cubulet ii corepund 3 coordonate, notate x, y, z. (exemplu: cubuletul din centrul cubului are coordonatele 0, 0, 0, iar cel din fata lui 0, 0, 1)
2. Pentru a roti o fata a cubului, o privim ca pe o forma 2D. Valorile sale sunt retinute intr-o matrice care arata astfel:

|  |  |  |
| --- | --- | --- |
| -1 -1 | 0 -1 | 1 -1 |
| -1 0 | 0 0 | 1 0 |
| -1 1 | 0 1 | 1 1 |

Dupa rotirea fetei, matricea va arata astfel:

|  |  |  |
| --- | --- | --- |
| -1 1 | -1 0 | -1 -1 |
| 0 1 | 0 0 | 0 -1 |
| 1 1 | 1 0 | 1 -1 |

Aceasta rotire este efectuata de functia Turn Z.

void turnZ(int index, int dir){

for(int i=0;i<cube.length;i++){

Box qb=cube[i];

if(qb.z==index){

PMatrix2D matrix= new PMatrix2D();

matrix.rotate(dir\*HALF\_PI);

matrix.translate(qb.x,qb.y);

qb.turnFacesZ(dir);

qb.update(round(matrix.m02),round(matrix.m12),round(qb.z));

}

}

}

Functia e repetata de 3 ori pentru fie care dintre cele 3 directii.

Fiecarei fete ii corespunde pe langa vectorul de pozitie si o culuare cu care e initializata in structura:

faces[0]=new Face(new PVector(0,0,-1), color(#ff751a));//spate

faces[1]=new Face(new PVector(0,0,1), color(#ff0000 ));//fata

faces[2]=new Face(new PVector(0,1,0), color(#ffff33));//sus

faces[3]=new Face(new PVector(0,-1,0), color(#333333));//jos

faces[4]=new Face(new PVector(1,0,0), color(#3366ff));//dr

faces[5]=new Face(new PVector(-1,0,0), color(#66ff66));//st

Toate miscarile posibile sunt puse intr un vector, cu scopul de a fi mai usor de accesat. Pentru fiecare directie din spatiu exista 4 miscari facute pentru a roti cubul. Directiei Ox ii corespund fetele stanga si drepata, fiecare putand fi miscata in sus sau in jos. Directiei Oy ii corespund fetele sus si jos care se pot misca fiecare in stanga sau in dreapta. Directiei Oz ii corespund fetele stape si fata care se pot misca in sensul acelor de ceasornic sau invers. Astfel, vectorul de mutari contine 16 elemente.

Move[] allMoves = new Move[]{

new Move (0, 1, 0, 1),

new Move (0, 1, 0,-1),

new Move (0, -1, 0, 1),

new Move (0, -1, 0, -1),

new Move(1, 0, 0, 1),

new Move (1, 0, 0, -1),

new Move(-1,0, 0, 1),

new Move (-1, 0, 0, -1),

new Move (0,0 , 1, 1),

new Move (0, 0, 1, -1),

new Move (0, 0, -1, 1),

new Move (0, 0, -1,-1)

};

**7 Surse de documentare on-line:**

https://www.youtube.com/watch?v=o9sgjuh-CBM&list=PLiYf1keoXdGz4-GV1ed8lvJQMN\_2gdv1j&index=4

https://processing.org/tutorials/

https://www.geeksforgeeks.org/introduction-to-processing-java/

https://www.youtube.com/watch?v=9PGfL4t-uqE&list=PLiYf1keoXdGz4-GV1ed8lvJQMN\_2gdv1j&index=1

https://www.youtube.com/watch?v=EGmVulED\_4M&list=PLiYf1keoXdGz4-GV1ed8lvJQMN\_2gdv1j&index=2&t=2649s

# 8. Anexe

8.1 Main program

import peasy.\*;

PeasyCam cam;

//sus jos dr st fata spate

color[] colors ={

#333333 , #ffff33 ,

#3366ff , #66ff66 ,

#ff0000 , #ff751a

};

Move[] allMoves = new Move[]{

new Move (0, 1, 0, 1),

new Move (0, 1, 0,-1),

new Move (0, -1, 0, 1),

new Move (0, -1, 0, -1),

new Move(1, 0, 0, 1),

new Move (1, 0, 0, -1),

new Move(-1,0, 0, 1),

new Move (-1, 0, 0, -1),

new Move (0,0 , 1, 1),

new Move (0, 0, 1, -1),

new Move (0, 0, -1, 1),

new Move (0, 0, -1,-1)

};

int dim = 3;

Box[] cube = new Box[dim\*dim\*dim];

Move move= allMoves[4];

void setup(){

//size(600,600,P3D);

fullScreen(P3D);

cam = new PeasyCam(this, 400);

int index=0;

for(int x=-1;x<=1;x++) {

for(int y=-1;y<=1;y++){

for(int z=-1;z<=1;z++){

PMatrix3D matrix = new PMatrix3D();

matrix.translate(x,y,z);

cube[index] = new Box(matrix,x,y,z);

index++;

}

}

}

cube[2].c=color(255,0,0);

cube[0].c=color(0,0,255);

//move=new Move(0,1 ,0,1);

}

int index=0;

void turnZ(int index, int dir){

for(int i=0;i<cube.length;i++){

Box qb=cube[i];

if(qb.z==index){

PMatrix2D matrix= new PMatrix2D();

matrix.rotate(dir\*HALF\_PI);

matrix.translate(qb.x,qb.y);

qb.turnFacesZ(dir);

qb.update(round(matrix.m02),round(matrix.m12),round(qb.z));

}

}

}

void turnX(int index,int dir){

for(int i=0;i<cube.length;i++){

Box qb=cube[i];

if(qb.x==index){

PMatrix2D matrix= new PMatrix2D();

matrix.rotate(dir\*HALF\_PI);

matrix.translate(qb.y,qb.z);

qb.turnFacesX(dir);

qb.update(round(qb.x),round(matrix.m02),round(matrix.m12));

}

}

}

void turnY(int index,int dir){

for(int i=0;i<cube.length;i++){

Box qb=cube[i];

if(qb.y==index){

PMatrix2D matrix= new PMatrix2D();

matrix.rotate(dir\*HALF\_PI);

matrix.translate(qb.x,qb.z);

qb.turnFacesY(dir);

qb.update(round(matrix.m02),round(qb.y),round(matrix.m12));

}

}

}

void keyPressed(){

switch(key){

case 'w':

//turnZ(-1,-1);

move=allMoves[11];

allMoves[11].start();

break;

case 'e':

//turnZ(1,-1);

move=allMoves[9];

allMoves[9].start();

break;

case 'W':

//turnZ(-1,1);

move=allMoves[10];

allMoves[10].start();

break;

case 'E':

//turnZ(1,1);

move=allMoves[8];

allMoves[8].start();

break;

case 's':

//turnY(-1,-1);

move=allMoves[3];

allMoves[3].start();

break;

case 'd':

//turnY(1,-1);

move=allMoves[1];

allMoves[1].start();

break;

case 'S':

//turnY(-1,1);

move=allMoves[2];

allMoves[2].start();

break;

case 'D':

//turnY(1,1);

move=allMoves[0];

allMoves[0].start();

break;

case 'x':

//turnX(-1,-1);

move=allMoves[7];

allMoves[7].start();

break;

case 'c':

move=allMoves[5];

allMoves[5].start();

break;

case 'X':

//turnX(-1,1);

move=allMoves[6];

allMoves[6].start();

break;

case 'C':

move=allMoves[4];

allMoves[4].start();

break;

}

if(key==' ')

move.start();

}

void draw(){

background(#cccccc);

scale(70);

for(int i=0;i<cube.length;i++){

push();

if(abs(cube[i].z)>0&&cube[i].z==move.z){

rotateZ(move.angle);

}

else if(abs(cube[i].x)>0&&cube[i].x==move.x){

rotateX(move.angle);

}

else if(abs(cube[i].y)>0&&cube[i].y==move.y){

rotateY(-move.angle);

}

cube[i].show();

pop();

}

move.update();

}

8.1 Class Box

class Box{

PMatrix3D matrix;

int x=0;

int y=0;

int z=0;

color c;

Face[]faces= new Face[6];

Box(PMatrix3D m, int x, int y, int z){

this.matrix =m;

this.x=x;

this.y=y;

this.z=z;

c=color(255);

faces[0]=new Face(new PVector(0,0,-1), color(#ff751a));//spate

faces[1]=new Face(new PVector(0,0,1), color(#ff0000 ));//fata

faces[2]=new Face(new PVector(0,1,0), color(#ffff33));//sus

faces[3]=new Face(new PVector(0,-1,0), color(#333333));//jos

faces[4]=new Face(new PVector(1,0,0), color(#3366ff));//dr

faces[5]=new Face(new PVector(-1,0,0), color(#66ff66));//st

}

void turnFacesZ(int dir){

for(Face f: faces)

f.turnZ(dir\*HALF\_PI);

}

void turnFacesY(int dir){

for(Face f: faces)

f.turnY(dir\*HALF\_PI);

}

void turnFacesX(int dir){

for(Face f: faces)

f.turnX(dir\*HALF\_PI);

}

void update(int x, int y, int z){

matrix.reset();

matrix.translate(x,y,z);

this.x=x;

this.y=y;

this.z=z;

}

void show(){

noFill();

stroke(0);

strokeWeight(0.1);

pushMatrix();

applyMatrix(matrix);

box(1);

for(Face f : faces){

f.show();

}

popMatrix();

}

}

8.2 Class Face

class Face{

PVector normal;

color c;

Face(PVector normal, color c){

this.normal=normal;

this.c=c;

}

void turnZ(float angle){

PVector v2 = new PVector();

v2.x= round(normal.x \*cos(angle) - normal.y\*sin(angle) ) ;

v2.y= round (normal.x \* sin(angle) + normal.y \* cos (angle)) ;

v2.z = round (normal.z);

normal= v2;

}

void turnY(float angle){

PVector v2 = new PVector();

v2.x= round(normal.x \*cos(angle) - normal.z\*sin(angle) ) ;

v2.z= round (normal.x \* sin(angle) + normal.z \* cos (angle)) ;

v2.y = round (normal.y);

normal= v2;

}

void turnX(float angle){

PVector v2 = new PVector();

v2.y= round(normal.y \*cos(angle) - normal.z\*sin(angle) ) ;

v2.z= round (normal.y\* sin(angle) + normal.z \* cos (angle)) ;

v2.x = round (normal.x);

normal= v2;

}

void show(){

pushMatrix();

fill(c);

noStroke();

rectMode(CENTER);

translate(0.5\*normal.x,0.5\*normal.y,0.5\*normal.z);

//rotate(HALF\_PI,normal.x,normal.y,normal.z);

if(abs(normal.z) > 0){

rotateZ(HALF\_PI);

}else if(abs(normal.x)>0){

rotateY(HALF\_PI);

}else if(abs(normal.y)>0){

rotateX(HALF\_PI);

}

square(0,0,1);

popMatrix();

}

}

8.3 Class Move

class Move{

float angle=0;

int x=0;

int y=0;

int z=0;

int dir;

boolean animatie = false;

boolean finished = false;

Move(int x, int y, int z, int dir){

this.x=x;

this.y=y;

this.z=z;

this.dir=dir;

}

void start(){

animatie=true;

}

boolean finished(){

return finished;

}

void update(){

if(animatie){

angle+=dir\*0.1;

if(abs(angle)>HALF\_PI){

angle=0;

if(abs(z)>0){

turnZ(z,dir);

}else if(abs(x)>0){

turnX(x,dir);

}else if(abs(y)>0){

turnY(y,dir);

}

animatie=false;

}

}

}

}