|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | |  | | |
| IBS-2 PROJECT REPORT | | |  | | |
|  | | |  | | |
|  | | | 27th June, 202019BIO113 | | |
|  | | | Abijith Pradeep | | |
|  |  |  | |  |

# Acknowledgement

I would like to thank Mr. K.P Soman to grant me an opportunity to make this project report analysis and helping throughout the course. I would also like to thank Mr. Premjith and Mr. Paul for guiding and helping me throughout course.

# Table of Contents

Cover page ……………………………………………………………………………………………………….. i

Acknowledgement ………….…………………………………………………………………………………. ii

Part 1 ……………………………………………………………………………………………………………….. 1

Code ……....……………………………………………………………………………………………………….. 2

Explanation …..…..…………………………………………………………………………………………….. 9

Output ……….……..…………………………………………………………………………………………….. 10

Part 2 ……………..……………………………………………………………………………………………….. 11

Table ……………………………………………………………………………………………………………….. 12

Justification ……………………………………………………………………………………………………… 13

References ……………………………………………………………………………………………………….. 14

# Part 1

# Processing Implementation of a Rubik’s Cube

# Code

## Cube

import peasy.\*;

PeasyCam cam;

int dim = 3;

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

String[] allMoves = {"f", "b", "u", "d", "l", "r"};

String sequence = "";

int counter = 0;

boolean started = false;

void setup() {

size(600, 600, 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 Cubie(matrix, x, y, z);

index++;

}

}

}

for (int i = 0; i < 20; i++) {

int r = int(random(allMoves.length));

if (random(1) < 0.5) {

sequence += allMoves[r];

} else {

sequence += allMoves[r].toUpperCase();

}

}

for (int i = sequence.length()-1; i >= 0; i--) {

String nextMove = flipCase(sequence.charAt(i));

sequence += nextMove;

}

}

String flipCase(char c) {

String s = "" + c;

if (s.equals(s.toLowerCase())) {

return s.toUpperCase();

} else {

return s.toLowerCase();

}

}

void turnZ(int index, int dir) {

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

Cubie qb = cube[i];

if (qb.z == index) {

PMatrix2D matrix = new PMatrix2D();

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

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

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

qb.turnFacesZ(dir);

}

}

}

void turnY(int index, int dir) {

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

Cubie qb = cube[i];

if (qb.y == index) {

PMatrix2D matrix = new PMatrix2D();

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

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

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

qb.turnFacesY(dir);

}

}

}

void turnX(int index, int dir) {

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

Cubie qb = cube[i];

if (qb.x == index) {

PMatrix2D matrix = new PMatrix2D();

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

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

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

qb.turnFacesX(dir);

}

}

}

void draw() {

background(51);

if (started) {

if (frameCount % 10 == 0) {

if (counter < sequence.length()) {

char move = sequence.charAt(counter);

applyMove(move);

counter++;

}

}

}

scale(50);

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

cube[i].show();

}

}

## Cubie

class Cubie {

PMatrix3D matrix;

int x = 0;

int y = 0;

int z = 0;

color c;

Face[] faces = new Face[6];

Cubie(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(0, 0, 255));

faces[1] = new Face(new PVector(0, 0, 1), color(0, 255, 0));

faces[2] = new Face(new PVector(0, 1, 0), color(255, 255, 255));

faces[3] = new Face(new PVector(0, -1, 0), color(255, 255, 0));

faces[4] = new Face(new PVector(1, 0, 0), color(255, 150, 0));

faces[5] = new Face(new PVector(-1, 0, 0), color(255, 0, 0));

}

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() {

//fill(c);

noFill();

stroke(0);

strokeWeight(0.1);

pushMatrix();

applyMatrix(matrix);

box(1);

for (Face f : faces) {

f.show();

}

popMatrix();

}

}

## Controls

void keyPressed() {

if (key == ' ') {

started = true;

}

}

void applyMove(char move) {

switch (move) {

case 'f':

turnZ(1, 1);

break;

case 'F':

turnZ(1, -1);

break;

case 'b':

turnZ(-1, 1);

break;

case 'B':

turnZ(-1, -1);

break;

case 'u':

turnY(1, 1);

break;

case 'U':

turnY(1, -1);

break;

case 'd':

turnY(-1, 1);

break;

case 'D':

turnY(-1, -1);

break;

case 'l':

turnX(-1, 1);

break;

case 'L':

turnX(-1, -1);

break;

case 'r':

turnX(1, 1);

break;

case 'R':

turnX(1, -1);

break;

}

}

## 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);

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

rotateY(HALF\_PI);

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

rotateX(HALF\_PI);

}

square(0, 0, 1);

popMatrix();

}

# Explanation

The Cube tab is the main class where all the background processes are running. The void setup has all the setup required and the usage of Peasy cam is done to ensure the total viewing of cube. Using the left mouse button and moving the mouse we can view all the sides of the cube. Double-click anywhere on the screen will reset its position. Then we have a string array of the list of moves which when pressed on the keyboard will do that particular function. Then we have the turning of a side of the cube, it rotates by pi/2 degrees about X or Y or Z direction. There is also the void draw where the frame rate of the cube movement and does the rotation to shuffle the cube and solve it back. All this is done in a random manner.

The Cubie tab has the 3DMatrix type which is a single unit in the Main cube. Like a corner piece or a middle piece etc. Here the definition of colors occurs. Which side will have what color and which side color should be opposite to the color given? These colors are implemented using array of object type Face which will be discussed below. Then we have the turn face X, Y and Z functions which rotates the normal of the faces for all six sides. These functions are a for loop of the function turnX, turnY, turnZ of the type Face. Then there is a void update which updates the orientation of the Cubies once the rotations are done. The void show shows the total cube with the faces and their colors.

The Face tab has the 2DMatrix type which resembles the face of each side of the whole cube. It has the PVector normal and color c parameter for telling which orientation it must be in and what color it must have. The are turnX, turnY and turnZ functions to turn the faces of the cube in that manner. There is a void show now which draws the face all overlapped, so the faces to avoid the overlapping we’re doing the rotateX and rotateY based on what the normal’s co-ordinates are. The built-in square function is used.

The Controls tab is used for mapping the keyboard keys for the respective turns and in what direction. We use the built-in keypressed function for this. The applymove function has the switch case command of the keys and what to do when encountered with that key.

### Control set

|  |  |
| --- | --- |
| Case | Rotation type |
| f | Clockwise turnZ of positive axis |
| F | Counter Clockwise turnZ of positive axis |
| b | Clockwise turnZ of negative axis |
| B | Counter Clockwise turnZ of negative axis |
| u | Clockwise turnY of positive axis |
| U | Counter Clockwise turnY of positive axis |
| d | Clockwise turnY of negative axis |
| D | Counter Clockwise turnY of negative axis |
| r | Clockwise turnX of positive axis |
| R | Counter Clockwise turnX of positive axis |
| l | Clockwise turnX of negative axis |
| L | Counter Clockwise turnX of negative axis |

# Output

[C:\Users\321ni\Videos\Captures\Cube 2020-06-27 22-07-37.mp4](file:///C:\Users\321ni\Videos\Captures\Cube%202020-06-27%2022-07-37.mp4)

The video output for the processing code.

# Part 2

# NCBI Blast search experiment

# Table



To view the table please double click it. It is an embedded excel file.

# Justification

The way for selecting that particular sequence is due to a major factor known as the max score. Then we check for the query cover. Another way telling this is when the total query is the same as the max query obtained from the sequence. For most of the sequences the E value obtained were zero. Hence, differentiation on the basis of E value was not possible.

Therefore, when checking for some sequences although have low query cover or percent identity they are placed above the ones with more of these factors. The reason being max score.

When checking for query cover if one or more sequences have the same query cover we move on to our final factor known as the Percent identity. The percent identity can never be identical for any two sequences.

# References

### Coding competitions videos

### <https://blast.ncbi.nlm.nih.gov/Blast.cgi>