**映像メディア処理　2019前学期　レポート1**

**氏名　栗山純平　　　クラス　3EP3　　　クラス番号 21　　　　　　貢献度　50%**

**氏名　小島尚之　　　クラス　3EP3　　　クラス番号 22　　　　　　貢献度　50％**

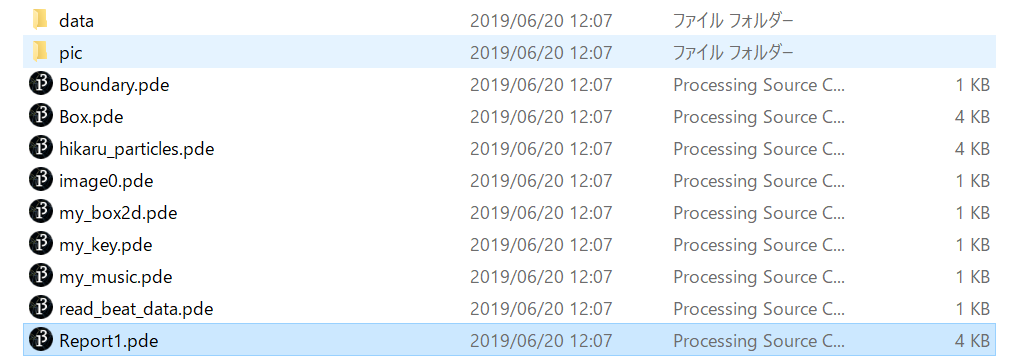
提出締め切り：6月20日

提出物：このレポートの印刷物[グループで一部]、プロジェクトプログラム

満点：20点

演習課題の「音楽ゲーム」完成しましょう。具体的には、以下の機能をプログラミングによって、実現しましょう。それぞれの部分に関連するプログラムコードを入れましょう。

1. 音楽データを用意する（音楽のmp3ファイルとbeatデータファイル）[２点]。dataフォルダーの写真を載せる。



1. 音楽データを読み込み、背景音としてプレーする。また、beatデータを配列に保存する。さらに、beatデータに合わせて、画面の上方にあるstart位置において、Box型のオブジェクトを発生させる。[３点]　関連するコードだけ入れる

・音楽データを読み込み、背景音としてプレーする

void setup\_music()

{

minim = new Minim(this);

song = minim.loadFile("DAOKO × 米津玄師『打上花火』MUSIC VIDEO.mp3");

rectMode(CENTER);

bww = box\_w;

bhh = box\_h;

}

・beatデータを配列に保存する

for(int i=0;i< a.getRowCount();i++)

{

btime[i] = a.getFloat(i, 0);

btype[i] = a.getInt(i, 1);

println(i+" "+btime[i]+" "+btype[i]);

}

}

・Box型のオブジェクトを発生させる

void makeBody(Vec2 center, float w\_, float h\_) {

PolygonShape sd = new PolygonShape();

float box2dW = box2d.scalarPixelsToWorld(w\_/2);

float box2dH = box2d.scalarPixelsToWorld(h\_/2);

sd.setAsBox(box2dW, box2dH);

// Define a fixture

FixtureDef fd = new FixtureDef();

fd.shape = sd;

fd.density = 4.8;

fd.friction = 0.5;

fd.restitution = 0.5;

// Define the body and make it from the shape

BodyDef bd = new BodyDef();

bd.type = BodyType.DYNAMIC;

bd.position.set(box2d.coordPixelsToWorld(center));

body = box2d.createBody(bd);

body.createFixture(fd);

}

}

1. Box型のオブジェクトは、壁（Boundary）を利用して作ったレールに沿って落ちる。また、レールの上方にスタート位置、レールの下方にゴール位置をそれぞれ見えるように表示する。[２点]　関連するコードだけ入れる

・my\_box2d

void setup\_box2d()

{

box2d = new Box2DProcessing(this);

box2d.createWorld();

box2d.setGravity(0, -8);

boxes = new ArrayList<Box>();

boundaries = new ArrayList<Boundary>();

boundaries.add(new Boundary(width-bw\*0.5, 0, bw, 4\*height));

boundaries.add(new Boundary(bw\*0.5, 0, bw, 4\*height));

boundaries.add(new Boundary(bw\*0.5 + (width - bw)/3, 0, bw, 4\*height));

boundaries.add(new Boundary(bw\*0.5 + 2\*(width - bw)/3, 0, bw, 4\*height));

}

・Report

void setup\_game()

{

box\_w = 42;

box\_h = 20;

bw = 84;//boundary width

setup\_box2d();

nnww = (width - 4\*bw)/3.0;

goal\_y = height - 30;

start\_y = 30;

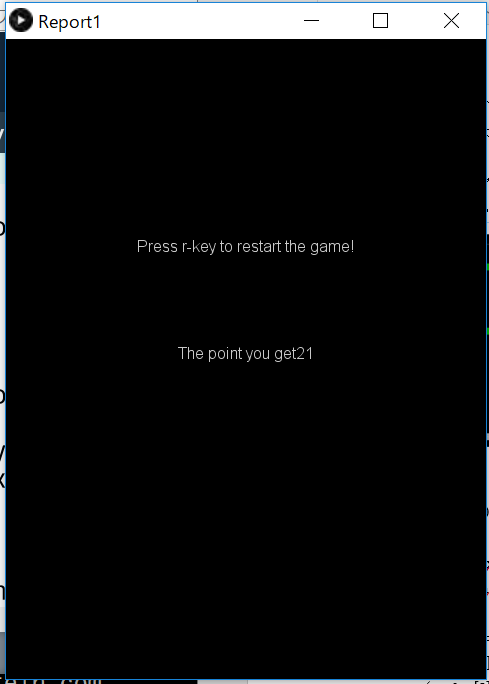
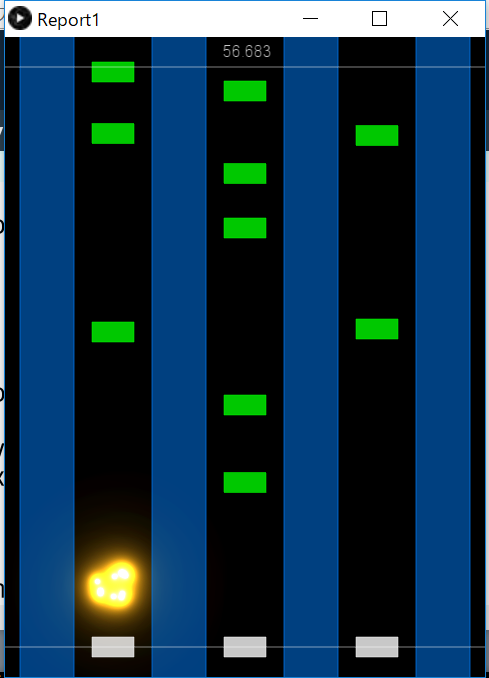
goal\_x[0] = bw + nnww/2.0;

goal\_x[1] = 2\*bw + nnww + nnww/2.0;

goal\_x[2] = width - bw - nnww/2.0;

}

1. キーボードのキーを利用して、それぞれのレールに対応する入力機能を作る。オブジェクトはゴール位置まで落ちてきたとき、タイミングよくキーを押していれば、得点する。得点したとき、ビジュアル的なフィードバックとして、その瞬間で撮影したプレヤーの画像を表示する。また、得点を記録する。[５点]　関連するコードだけ入れる



・Box

boolean compare()

{

Vec2 pos = box2d.getBodyPixelCoord(body);

if ((abs(pos.y - goal\_y) < box\_h/2.0 + box\_h && keyPressed ))

{

if((goal\_x[0] == pos.x && key == 'a') ||

(goal\_x[1] == pos.x && key == 'b') ||

(goal\_x[2] == pos.x && key == 'c'))

{

//得点を足す

sum++;

//文字

text("GREAT", pos.x, pos.y - box\_h);

//画像

if (video.available()) {

video.read();//カメラからビデオ映像を撮影する

}

opencv.loadImage(video);

opencv.useColor();

hikaru\_able = 1;

hikaru\_timer = my\_timer;

cxx = pos.x;

cyy = pos.y - 30; //pos.y;

for (int pid = 0; pid < MAX\_PARTICLE; pid++) {

hikaru\_p[pid].explode();

}

KillBody();

return true;

}

}

return false;

}

・Report

void draw()

{

background(0);

stroke(255);

box2d.step();

　image(src, 0, 100, src.width, src.height);

　if(hikaru\_able == 1 && my\_timer - hikaru\_timer < 0.5)

draw\_hikaru\_particles();

else

{

hikaru\_able = 0;

hikaru\_timer = 0.0;

}

・hikaru\_particles

final int MAX\_PARTICLE = 10;

hikaru\_Particle[] hikaru\_p = new hikaru\_Particle[MAX\_PARTICLE];

final int LIGHT\_FORCE\_RATIO = 5;

final int LIGHT\_DISTANCE = 105 \* 105;

final int BORDER = 105;

float baseRed, baseGreen, baseBlue;

float baseRedAdd, baseGreenAdd, baseBlueAdd;

final float RED\_ADD = 1.2;

final float GREEN\_ADD = 1.7;

final float BLUE\_ADD = 2.3;

int hikaru\_able = 0;

float hikaru\_timer = 0.0;

void setup\_hikaru\_particles()

{

for (int i = 0; i < MAX\_PARTICLE; i++) {

hikaru\_p[i] = new hikaru\_Particle();

}

baseRed = 0;

baseRedAdd = RED\_ADD;

baseGreen = 0;

baseGreenAdd = GREEN\_ADD;

baseBlue = 0;

baseBlueAdd = BLUE\_ADD;

hikaru\_able = 0;

}

void draw\_hikaru\_particles()

{

baseRed += baseRedAdd;

baseGreen += baseGreenAdd;

baseBlue += baseBlueAdd;

colorOutCheck();

for (int pid = 0; pid < MAX\_PARTICLE; pid++) {

hikaru\_p[pid].move(cxx, cyy);

}

int tRed = (int)baseRed;

int tGreen = (int)baseGreen;

int tBlue = (int)baseBlue;

tRed \*= tRed;

tGreen \*= tGreen;

tBlue \*= tBlue;

loadPixels();

for (int pid = 0; pid <MAX\_PARTICLE; pid++) {

int left = max(0, hikaru\_p[pid].x - BORDER);

int right = min(width, hikaru\_p[pid].x + BORDER);

int top = max(0, hikaru\_p[pid].y - BORDER);

int bottom = min(height, hikaru\_p[pid].y +BORDER);

for (int y = top; y < bottom; y++){

for (int x = left; x < right; x++) {

int pixelIndex = x + y \* width;

int r = pixels[pixelIndex] >> 16 & 0xFF;

int g = pixels[pixelIndex] >> 8 & 0xFF;

int b = pixels[pixelIndex] & 0xFF;

int dx = x - hikaru\_p[pid].x;

int dy = y - hikaru\_p[pid].y;

int distance = (dx \* dx) + (dy \* dy);

if (distance <LIGHT\_DISTANCE) {

int fixFistance = distance \* LIGHT\_FORCE\_RATIO;

if (fixFistance == 0) {

fixFistance = 1;

}

r = r + tRed / fixFistance;

g = g + tGreen / fixFistance;

b = b + tBlue / fixFistance;

}

pixels[x + y\* width] = color(r, g, b);

}

}

}

updatePixels();

}

void colorOutCheck() {

if (baseRed < 10) {

baseRed = 10;

baseRedAdd \*= -1;

}

else if (baseRed >255) {

baseRed = 255;

baseRedAdd \*= 1;

}

if (baseGreen < 10) {

baseGreen = 10;

baseGreenAdd \*= -1;

}

else if (baseGreen > 255) {

baseGreen = 255;

baseGreenAdd \*= -1;

}

if (baseBlue < 10) {

baseBlue = 10;

baseBlueAdd \*= -1;

}

else if (baseBlue > 255){

baseBlue = 255;

baseBlueAdd \*= -1;

}

}

class hikaru\_Particle {

int x,y;

float vx, vy;

float slowLevel;

final float DECEL\_RATIO = 0.95;

hikaru\_Particle() {

x = (int)random(width);

y = (int)random(height);

slowLevel = random(100) + 5;

}

void move(float targetX, float targetY) {

x = (int)(targetX + 5.0\*vx\*(my\_timer - hikaru\_timer));

y = (int)(targetY + 5.0\*vy\*(my\_timer - hikaru\_timer));

}

void explode() {

vx = random(100) - 50;

vy = random(100) - 50;

slowLevel = random(100) + 5;

}

}

1. ゲームを三つのstateで制御する。State==0のとき、スタート画面を表示する。State == 1のとき、プレー画面を表示する。State==2のとき、結果画面を表示する。[３点]　関連するコードだけいれる

・Report\_1

if(state == 0) //スタート画面

{

text("Press p-key to start a game!", width/2, height/2);

}

else if(state == 1) //プレイ画面

{ //play game

draw\_my\_game();

if(abs(my\_timer - falling\_time - btime[index]) < 0.01)

{

add\_box(btype[index]);

index++;

}

if (index >= a.getRowCount() ) index = 0;

text(my\_timer, width/2, 20);

my\_timer = my\_timer + 1.0/60.0;

}

else if(state == 2) //結果表示画面

{

text("The point you get" +sum, width/2, height/2);

text("Press r-key to restart the game!", width/2, height/3);

}

}

・My\_key

void keyPressed()

{

if(key == 'p' && state == 0)

{

state = 1;

my\_timer = 0.0;

index = 0;

song.rewind();

song.play();

}

if(key == 'e' && state == 1)

{

state = 2;

song.pause();

}

if(key == 'r' && state == 2)

{

state = 0;

}

}

1. 得点したとき、ビジュアル的なフィードバックとして、独自に用意した解像度の低い画像を、キューブのパーティクルの形で表示する。(5点)　関連するコードを入れる

import gab.opencv.\*;

import processing.video.\*;

PImage src;

OpenCV opencv;

Capture video;

void setup\_image()

{

video = new Capture(this, 640, 480); //カメラを設置する

video.start(); //カメラ撮影スタートする

colorMode(RGB, 255, 255, 255, 255);

opencv = new OpenCV(this, 640, 480);

if (video.available()) {

video.read();

}

　opencv.loadImage(video);

opencv.useColor();

src = opencv.getSnapshot();

if(keyPressed)

{

image(src, 0, 0, src.width/2, src.height/2);

}

}