実験項目	実験 A4 Android アプリ開発
校名　科名 学年　番号	熊本高等専門学校　　人間情報システム工学科 3 年　　42 号
氏名	山口惺司
班名　回数	4 班　　1 回目
実験年月日 建物　部屋名	2023 年　10 月　19 日　木曜　天候 晴 2023 年　11 月　2 日　木曜　天候 晴 3 号棟　　2 階　HIPC 室
共同実験者名	

1. 実験目的

本実験では以下の事柄について学ぶ。

（ア）Processing での Android アプリ開発手順を知ること。

（イ）タップ、フリックなどのスマートフォン特有の操作に対応したアプリを作ること。

（ウ）GPS、方位磁針、ジャイロセンサなどスマートフォン特有のセンサに対応したアプリを作ること。

2. 実験内容

1. モグラたたきソフト

課題文：

あらかじめもぐらの場所を乱数で決め、ユーザがその周辺（50 ピクセル以内）をタップすると「当たり」 とするようなもぐらたたきゲームを開発する。

モグラの場所(mx, my)は、大域変数 mx, my を用意し、setup()の中で mx = random(width)、my = random(height)として設定する。そして既定の関数 onTap()の中で、タップした場所とモグラの場所との距離 を求め、距離が 100 以内なら「当たり」と表示する。距離を求める際には dist(x, y, mx, my)命令を利用する と良い。 また、モグラをタップした後に再度 mx = random(width)、my = random(height)してモグラを別の場所に移動させるとよい。

プログラムについての説明：

画面上にもぐらが現れ、もぐらをタップするとスコアを+1 し、もぐらをランダムに再配置するプログラム。

実行例：

実行例を図 1 に示す。

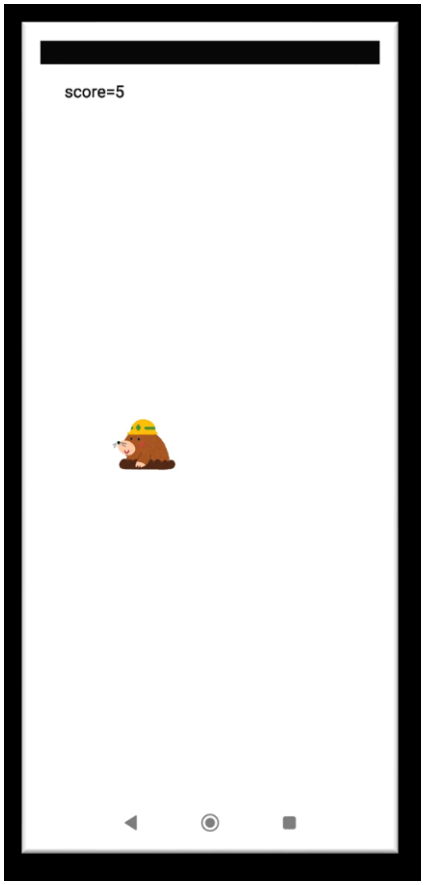

図 1 もぐらたたき実行例

ソースコード：

```
import android.view.MotionEvent;
import ketai.ui.*;

// Ke:tai 機能を使うための大域変数
KetaiGesture gesture;

//--------------------------------------
// ke:tai ライブラリの所定の関数に操作イベントを引き渡すための"おまじない"
public boolean surfaceTouchEvent(MotionEvent e) {
  super.surfaceTouchEvent(e);
  return gesture.surfaceTouchEvent(e);
}

//--------------------------------------
// (動作確認用) 画面に文字列 s を表示する関数
//
void drawString(String s, float x, float y) {
  background(255);
  fill(0);
  textSize(50);
```

```
  text(s, x, y);
}

//--------------------------------------
// アプリ実行開始時に一度だけ実行される初期化用関数
float mx;
float my;
PImage img;
PFont font;

void setup() {
  gesture = new KetaiGesture(this);
  background(255);
  mx = random(width);
  my = random(height);
  img = loadImage("mogura.png");
  imageMode(CENTER);
  image(img, mx, my, 200, 200);
}
int score = 0;
//--------------------------------------
// アプリ実行時に何度も繰り返し実行される main 関数
void draw() {
  text("score="+score, 100, 100);
}

void onTap(float x, float y) {
  float dis = dist(x, y, mx, my);
  if (dis <= 200) {
    drawString("Hit", mx, my);
    score++;
    setup();
  }
}
```

2. ペイントアプリ

課題文：

演習 A と B を組み合わせることで、より実用的な機能を持ったペイントアプリを作る。具体的には以下の 機構を実装する。

・「消しゴムモード」と「鉛筆モード」を切り替え、線を描いたり消したりできるようにする。

・全画面リセット機能を付け、画面を真っ白にリセットできるようにする。 それぞれの機能をどのような操作で行うかは各自で設計すること。(例えば「長押しすると全画面リセット」 するように実装するなど)。

プログラムについての説明：

ペン、消しゴム、色変更のボタンがあり、ボタンをタップすることでツールが切り替わる。

ゴミ箱ボタンを長押しすることで、一括削除することができる。

実行例：

実行例を図 2 に示す。

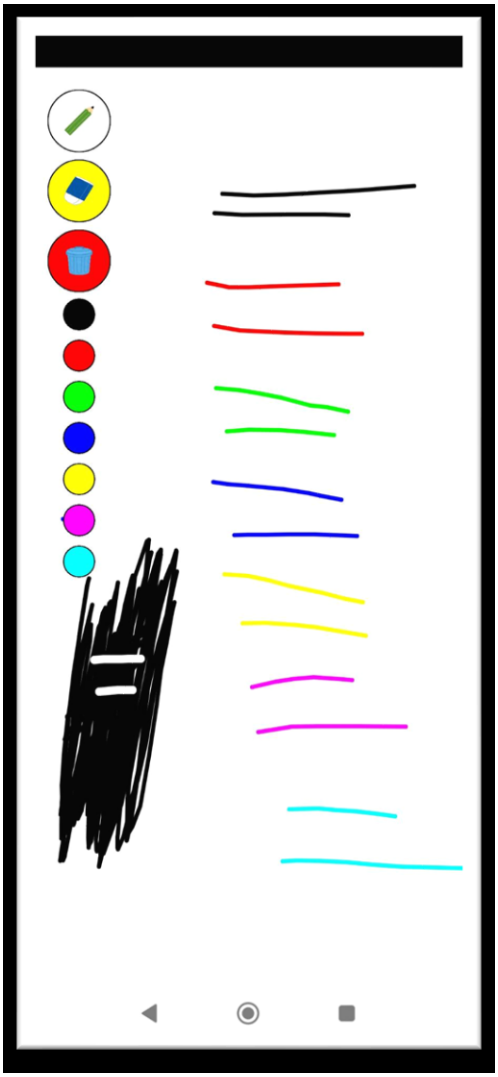

図 2 ペイントアプリ実行例

ソースコード：

```
import android.view.MotionEvent;
import ketai.ui.*;
```

```
KetaiGesture gesture;

public boolean surfaceTouchEvent(MotionEvent e) {
  super.surfaceTouchEvent(e);
  return gesture.surfaceTouchEvent(e);
}

PImage ImgPen;
PImage ImgEra;
PImage ImgGomi;
color black = color(0);
color red = color(255, 0, 0);
color green = color(0, 255, 0);
color blue = color(0, 0, 255);
color yellow = color(255, 255, 0);
color purple = color(255, 0, 255);
color cyan = color(0, 255, 255);
void setup() {
  gesture = new KetaiGesture(this);
  ImgPen = loadImage("pen.png");
  ImgEra = loadImage("eraser.png");
  ImgGomi = loadImage("gomi.png");
  imageMode(CENTER);
  background(255);
}

float ButtonPenX = 130, ButtonPenY = 130;
float ButtonEraX = 130, ButtonEraY = 300;
float ButtonResetX = 130, ButtonResetY = 470;
float colorX = 130;
float blackY = 600, redY = 700, greenY = 800, blueY = 900, yellowY = 1000, purpleY
= 1100, cyanY = 1200;
color colormode = black;
int mode = 0; //mode: 0 = pen, 1 = eraser, 2 = reset

void draw() {
  fill(255);


  if (mode == 0) {
    stroke(colormode);
    strokeWeight(10);
```

```
  line(mouseX, mouseY, pmouseX, pmouseY);
  stroke(0);
  strokeWeight(2);
} else if (mode == 1) {
  stroke(255);
  strokeWeight(20);
  line(mouseX, mouseY, pmouseX, pmouseY);
  strokeWeight(2);
  stroke(0);
}
strokeWeight(2);
if (mode == 0) {
  fill(255, 255, 0);
} else {
  fill(255);
}
ellipse(ButtonPenX, ButtonPenY, 150, 150);
image(ImgPen, ButtonPenX, ButtonPenY, 75, 75);

if (mode == 1) {
  fill(255, 255, 0);
} else {
  fill(255);
}
ellipse(ButtonEraX, ButtonEraY, 150, 150);
image(ImgEra, ButtonEraX, ButtonEraY, 75, 75);

fill(255, 0, 0);
ellipse(ButtonResetX, ButtonResetY, 150, 150);
image(ImgGomi, ButtonResetX, ButtonResetY, 75, 75);

fill(0);
ellipse(colorX, blackY, 75, 75);
fill(red);
ellipse(colorX, redY, 75, 75);
fill(green);
ellipse(colorX, greenY, 75, 75);
fill(blue);
ellipse(colorX, blueY, 75, 75);
fill(yellow);
ellipse(colorX, yellowY, 75, 75);
fill(purple);
```

```
  ellipse(colorX, purpleY, 75, 75);
  fill(cyan);
  ellipse(colorX, cyanY, 75, 75);
}


void onTap(float x, float y) {
  float ButtonPenD = dist(x, y, ButtonPenX, ButtonPenY);
  float ButtonEraD = dist(x, y, ButtonEraX, ButtonEraY);
  float blackD = dist(x, y, colorX, blackY);
  float redD = dist(x, y, colorX, redY);
  float greenD = dist(x, y, colorX, greenY);
  float blueD = dist(x, y, colorX, blueY);
  float yellowD = dist(x, y, colorX, yellowY);
  float purpleD = dist(x, y, colorX, purpleY);
  float cyanD = dist(x, y, colorX, cyanY);

  if (ButtonPenD <= 75) {
    mode = 0;
  }
  if (ButtonEraD <= 75) {
    mode = 1;
  }

  if (blackD <= 37.5) {
    colormode=black;
    mode = 0;
  }
  if (redD <= 37.5) {
    colormode=red;
    mode = 0;
  }
  if (greenD <= 37.5) {
    colormode=green;
    mode = 0;
  }
  if (blueD <= 37.5) {
    colormode=blue;
    mode = 0;
  }
  if (yellowD <= 37.5) {
    colormode=yellow;
    mode = 0;
```

```
  }
  if (purpleD <= 37.5) {
    colormode=purple;
    mode = 0;
  }
  if (cyanD <= 37.5) {
    colormode=cyan;
    mode = 0;
  }
}
void onLongPress(float x, float y) {
  float ButtonResetD = dist(x, y, ButtonResetX, ButtonResetY);
  if (ButtonResetD <= 100) {
    background(255);
  }
}
```

3. 加速度センサを用いたモグラ叩き

課題文：

（基本）

X と Y の加速度センサ値を使い、スマホの傾きに応じて岩（円）が転がり落ちるような動きをするプログラ ムを作れ。その仕組みを用い、(タップでモグラを叩くアプリではなく)岩を転がしてモグラにぶつけるアプリ を作れ。

（発展例１）

Processing では millis()命令でアプリ起動後の経過時間(単位:ミリ秒)を得られる。モグラを出したときの時 間 s、岩をモグラに当てたときの時間 e をそれぞれ取得し、e-s すると所要時間が分かるので、それをスコア として利用すると良い。

（発展例２）

ぶつけてはいけないもの（それが何かは自由に設計する）を配置し、そこにぶつかるとスコアが下がる仕組 みを追加する。

（発展例３）

画面内のある領域（その場所や大きさなどは自由に設計する）では、岩の移動速度が半分になるようにする。

（発展例４）

モグラが動き回るようにする。

プログラムについての説明：

スマホの傾きを利用して、石を動かしモグラに当てるゲーム。

モグラに石を当てるとスコアが＋１され、もぐらをランダムに配置する。

スコアが一定の値になると難易度が上がる。

スコア 10～：

沼が追加され、沼に石が入ると、移動速度が低下する。

スコア 20～：

もぐらが移動するようになる。

もぐらの再配置時にもぐらの移動速度と向きがもランダムに与えられる。

スコア 30～：

画面上部にあった鬼ヶ島から鬼が出現するようになる。

鬼も移動速度と向きがランダムに与えられ、鬼に当たるとスコアがー１される。

もぐらが再配置されたら、鬼も鬼ヶ島に戻る。

鬼はもぐら再配置から１秒経過しないと出てこられない。

実行例：

実行例を図 3～6 に示す。

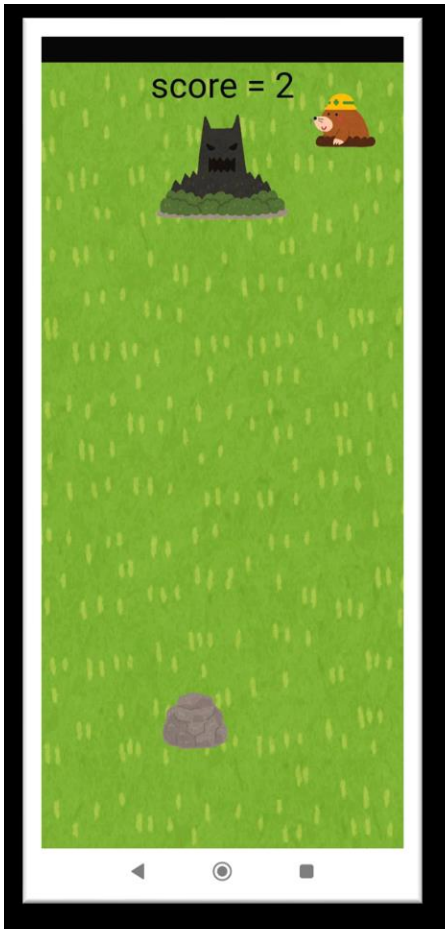

図 1　スコア～10 のモグラ叩き

図 2　スコア 10～のモグラ叩き

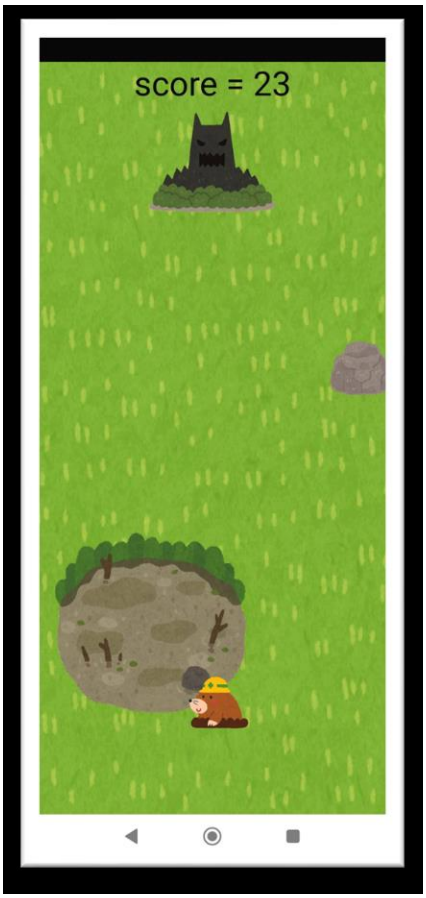

図 3　スコア 20～のモグラ叩き

図 4　スコア 30～のモグラ叩き

ソースコード：

```
//
// 加速度センサを利用したプログラム例。
//
// 大域変数 accX, accY, accZ に随時加速度センサ値が代入されるので、
// draw()関数内でそれらの値に沿った直線を描く。
//
import ketai.sensors.*;

KetaiSensor sensor;
float accX, accY, accZ;   // 加速度センサ値 3 つをいれる変数
float sx, sy;
float mx, my, mxs, mys, md;
float nx, ny, nd, ns = 1;
float ox, oy, od, oxs, oys;
PImage iwa;
PImage mogura;
PImage numa;
PImage oni;
PImage onigashima;
PImage shiba;
float world_time, t;
int score = 0;


void setup() {
  orientation(PORTRAIT); //PORTRAIT=スマホを縦長に固定、LANDSCAPE=横長に固定;
  sensor = new KetaiSensor(this);
  sensor.start();
  strokeWeight(5);
  textSize(36);
  fill(0);
  sx = width/2;
  sy = height/2;
  iwa = loadImage("iwa.png");
  mogura = loadImage("mogura.png");
  numa = loadImage("numa.png");
  oni = loadImage("oni.png");
  onigashima = loadImage("onigashima.png");
  shiba = loadImage("shiba.png");
  imageMode(CENTER);
```

```
  textAlign(CENTER);
  textSize(100);
  moguraHit();
}

void draw() {
  image(shiba, width/2, height/2, width, height);
  world_time = millis();
  sx -= accX*4 * ns;
  sy += accY*4 * ns;

  if (sx-100 < 0) {
    sx = 100;
  }
  if (sx+100 > width) {
    sx = width-100;
  }
  if (sy-100 < 0) {
    sy = 100;
  }
  if (sy+100 > height) {
    sy = height-100;
  }

  if (score >= 10) {
    image(numa, nx, ny, 600, 700);
    nd = dist(sx, sy, nx, ny);
    if (nd <= 320) {
      ns = 0.3;
    } else {
      ns = 1;
    }
  }
  if (score >= 30) {
    image(onigashima, width/2, 300, 400, 400);
  }



  stroke(0);
  image(iwa, sx, sy, 200, 200);
  image(mogura, mx, my, 200, 200);
```

```
  md = dist(mx, my, sx, sy);
  if (md <= 160) {
    score++;
    moguraHit();
  }
  if (score >= 20) {
    mx += mxs;
    my += mys;
    if (mx-100 < 0 || mx+100 > width) {
      mxs = -mxs;
    }
    if (my-100 < 0 || my+100 > height) {
      mys = -mys;
    }
  }
  if (score >= 30) {
    if (world_time - t > 1000) {
      image(oni, ox, oy, 300, 300);
      od = dist(ox, oy, sx, sy);
      ox += oxs;
      oy += oys;
      if (ox-100 < 0 || ox+100 > width) {
        oxs = -oxs;
      }
      if (oy-100 < 0 || oy+100 > height) {
        oys = -oys;
      }
      if (od <= 250) {
        score--;
        moguraHit();
      }
    }
  }

  text("score = " + score, width/2, 100);
}

//--------------------------------
// 加速度センサ値が届いたら随時呼ばれる関数
//
void onAccelerometerEvent(float x, float y, float z) {
  accX = x;
```

```
  accY = y;
  accZ = z;
}

void moguraHit() {
  t = world_time;
  mx = random(100, width-100);
  my = random(100, height-100);
  if (score >= 10) {
    nx = random(300, width-300);
    ny = random(350, height-350);
  }
  if (score >= 20) {
    mxs = random(-15, 15);
    mys = random(-15, 15);
  }
  if (score >= 30) {
    oxs = random(-20, 20);
    oys = random(-20, 20);
    ox = width/2;
    oy = 300;
  }
}
```

3. 感想

今回はProcessingを用いたAndroidアプリ開発を行ったが、ゲーム作りに興味があるためProcessing以外のソフトでもアプリ開発をしてみたいなと思った。

PCにはない操作(タップ、長押し、ダブルタップ、スワイプ、加速度センサ、GPSセンサ)など、様々な操作があり、とても面白かった。すべての操作を活かすことはできなかったが、今後アプリ開発をする機会があれば使用してみたい。