2016 年度情報メディア基盤ユニット
7 月 15 日分課題【総合演習】（修正版）

授業関連資料は http://www.sato-lab.jp/imfu からダウンロード出来ます。【後半にある【参考問題】は、演習時間中に解答出来なくてもかまいません。なお、問題は難易度順に並んでいるわけではありません。問 1～4、6 はキャリアポートフォリオにも解答をアップして下さい。

1. atan2 という関数を利用すると、角度を求めることが出来ます。例えば、2 点(x_0, y_0)と(x_1, y_1)と結ぶ直線と点(x_0, y_0)を通り X 軸に平行な直線のなす角度（単位はラジアン）は$\mathrm{atan2}(y_1 - y_0, x_1 - x_0)$で求めることが出来ます。これを利用すると、画像などを回転させて、ある特定の方向（例えば、マウスカーソルの方向）に向けるようにすることが出来ます。このことを実行したプログラムが次の未完成プログラムです。このプログラムでは、マウスカーソルの方向に、画像の先頭方向が向くようになっています。空欄を埋めて、プログラムを完成させて下さい。なお、読み込みに使っているファイルは右方向が先頭になったものを利用しています。

未完成プログラム

```
PImage panzer4;

void setup(){
  size(600,600);
  panzer4 = loadImage("PanzerIV.png");
}

void draw(){
  background(255);
  imageMode(CENTER);
  float angle = atan2(  (a)  -height/2,  (b)  - width/2);
  translate(width/2,height/2);
  (c) ;
  image(panzer4,0,0,panzer4.width/2,panzer4.height/2);
}
```

2. このプログラムでは、ウインドウの真ん中を中心とする扇型を描いています。開始方向は X 軸の正方向で、マウスカーソルの位置が終了点となります。空欄を埋めて、プログラムを完成させて下さい。

未完成プログラム

```
void setup(){
  size(400,400);
}
void draw(){
  background(255);
  float xCenter = width/2;
  float yCenter = height/2;
  float daimeter = 2*dist(mouseX,mouseY,xCenter,yCenter);
  float angle = atan2(  (a)  ,  (b)  );
  if(angle <  (c) ){
    angle = angle + 2*PI;
  }
  fill(0);
```

```
  stroke(0);
  arc(xCenter,yCenter,daimeter,daimeter,   (d)   ,   (e)   );
}
```

3. 授業でも説明したように、物体の移動をさせる方法の一つに、速度を変化させて、速度を利用して物体の位置を更新していくというものがあります。下のサンプルは、-45 度方向に円盤を投げたような動きをするプログラムになっています。このプログラムを変更して、マウスをクリックしたら、マウスの方向に向かって、円盤が飛び出すようなプログラムを完成させて下さい。なの、このサンプルでは、適当に定数を決めています。

サンプルプログラム

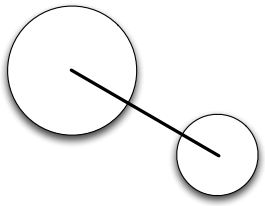
```
float vx;
float vy;
float xBall;
float yBall;
float speed;
float gravity=0.1;
void setup(){
  size(600,300);
  xStart = 0.1 * width;
  yStart = 0.5 * height;
  xBall = 0.1 * width;
  yBall = 0.5 * height;
  speed = 5;
  float angle = radians(-45);
  vx = speed * cos(angle);
  vy = speed * sin(angle);
}
void draw(){
  background(255);
  xBall += vx;
  yBall += vy;
  vy += gravity;
  fill(0);
  ellipse(xBall,yBall,10,10);
}
```

未完成のプログラム

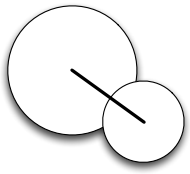
```
float vx;
float vy;
float xBall;
float yBall;
float speed;
float gravity=0.1;
float xStart; // ボールが飛び出す最初の座標
float yStart;

int state;

void setup(){
  size(600,300);
  xStart = 0.1 * width;
  yStart = 0.5 * height;
  xBall = xStart;
  yBall = yStart;
```

```
  speed = 5;
  vx = 0;
  vy = 0;
  state = 0;
}
void draw(){
  background(255);
  if(state == 0){
    fill(0);
    ellipse(xBall,yBall,10,10);
  }else{
    xBall += vx;
    yBall += vy;
    vy += gravity;
    fill(0);
    ellipse(xBall,yBall,10,10);
  }
}
void mouseClicked(){
  xBall = xStart;
  yBall = yStart;
  float dx = __(a)__ - xStart;
  float dy = __(b)__ - yStart;
  float angle = atan2(dy,dx);
  vx = speed * cos(angle);
  vy = speed * sin(angle);
  state = __(c)__;
}
```

4. ２つの円が衝突しているかどうかは、2 円の中心の距離とこの 2 つの円の半径の和を比べることで判定ができます（右図参照)。これを利用して、2 つの円が衝突しているときと、そうでないときで円の色を変えるプログラムとなっています。空欄を埋めて、プログラムを完成させて下さい。



中心間の距離が半径の和よりも
大きければぶつかっていない



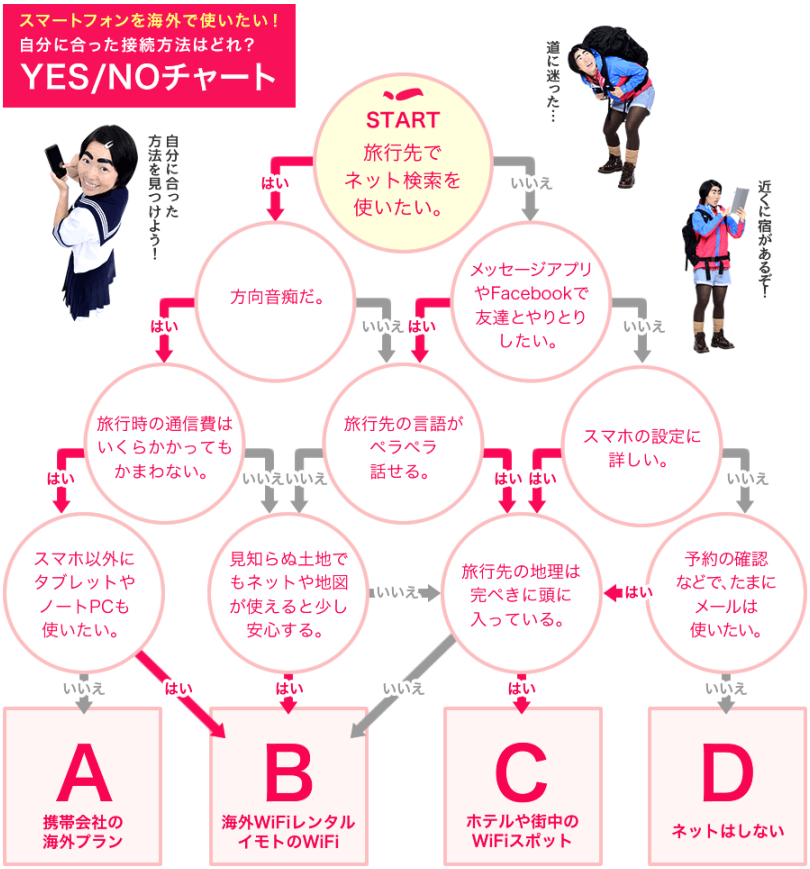
中心間の距離が半径の和よりも
小さければぶつかっている

未完成プログラム

```
float xCenter;
float yCenter;
__(a)__r0;
__(a)__r1;
void setup() {
  size(600, 600);
  xCenter = width/2;
  yCenter = height/2;
  r0 = 0.25*width;
  r1 = 20;
}
void draw() {
  background(255);
  if (__(b)__(mouseX, mouseY, xCenter, yCenter) < __(c)__) {
    fill(255, 10, 10);
  } else {
    fill(10, 255, 10);
  }
```

```
  ellipse(xCenter, yCenter, 2*r0, 2*r0);
  ellipse(mouseX, mouseY, 2*r1, 2*r1);
}
```

5. 次の画像のような Yes/No チャートを Processing のプログラム上で出来るようにして下さい。図は http://www.globaldata.jp/kijiworld/no1410c/のものを利用しています。



6. 以前の演習問題で、Y 軸に平行な線分と円の衝突（接触）判定を扱いました。プログラムを作る中では、両端点を指定し、その 2 点を結ぶ線分と円の衝突判定を行う方法が必要となります。これを行う方法の一つとして、以下のようなものが考えられます。説明の際には、両端点を P_0, P_1 とし、円の中心を Q とし、円の半径を r とします。以下のような場合には、線分と円が衝突しています。

① P_0とQの距離とP_1とQの距離がともにrより小さい。

② 直線P_0P_1と点Qとの距離がrよりも小さく、さらに$\angle P_0P_1Q$と$\angle P_1P_0Q$がともに鋭角となっている。

$\angle P_0P_1Q$が鋭角かどうかは、ベクトルの内積$\overrightarrow{P_1P_0} \cdot \overrightarrow{P_1Q}$の値の正負で判定することができます。内積の値が正であれば鋭角、負の値であれば鈍角となっています。この方針で線分と円が衝突しているかどうかを判定するプログラムが、つぎのサンプルです。以下の文章の空欄に適切な関数名や変数などを入れて下さい。

このサンプルプロラムでは、__(a)__関数によって、線分と円の衝突判定を行っている。この関数では、中心の座標（__(b)__,__(c)__）で半径が__(d)__の円と両端点の座標が(__(e)__,__(f)__)と(__(g)__,__(h)__)の線分の接触判定を行っている。この関数の戻り値は接触しているときには__(i)__、そうでないときには__(j)__となっている。この関数は、このサンプル内で定義している二つの関数を__(k)__と__(l)__を利用している。

サンプルプログラム

```
float vx;
float vy;
float xBall;
float yBall;

float dotProduct(float x1, float y1, float x2, float y2) {
  return x1*x2 + y1*y2;
}

float distance(float x0, float y0,
  float x1, float y1,
  float x2, float y2) {
  float vx1 = x0 - x1;
  float vy1 = y0 - y1;
  float vx2 = x2 - x1;
  float vy2 = y2 - y1;
  float c = dotProduct(vx1, vy1, vx2, vy2);
  return sqrt((vx1*vx1 + vy1*vy1) - (c*c/(vx2*vx2+vy2*vy2)));
}

boolean  isINCollionWith(float xCenter, float yCenter, float radius,
  float x1, float y1, float x2, float y2) {
  if (dist(xCenter, yCenter, x1, y1) <= radius) {
    return true;
  }
  if (dist(xCenter, yCenter, x2, y2) <= radius) {
    return true;
  }
  if (distance(xCenter, yCenter, x1, y1, x2, y2) > radius) {
    return false;
  }
  float vx0 = x2 - x1;
  float vy0 = y2 - y1;
  float vx1 = xCenter - x1;
  float vy1 = yCenter - y1;
  float vx2 = xCenter - x2;
  float vy2 = yCenter - y2;
```

```
  if (dotProduct(vx0, vy0, vx1, vy1) < 0) {
    return false;
  }
  if (dotProduct(-vx0, -vy0, vx2, vy2) < 0) {
    return false;
  }
  return true;
}

void setup() {
  size(600, 600);
  xBall = width/2;
  yBall = 0.9 * height;
  vx = 0;
  vy = -1;
}

void draw() {
  background(255);
  float t = 2*PI*frameCount/360.0;
  float radius = width/4;
  float angle = PI/2 + (PI/2-PI/12)*sin(t);
  float x0 = width/2 + radius * cos(angle);
  float y0 = height/2 +radius * sin(angle);
  float x1 = width/2 + radius * cos(angle+PI);
  float y1 = height/2 + radius * sin(angle+PI);
  line(x0, y0, x1, y1);

  xBall += vx;
  yBall += vy;
  fill(0);
  if (isINCollionWith(xBall, yBall, 10, x0, y0, x1, y1)) {
    fill(255, 10, 10);
  }
  ellipse(xBall, yBall, 20, 20);
}
```

授業中に配布したプリントにゲームに以下のような改良を加えてものを作成して下さい。このプリントの最後に関連する部分を再度載せておきます。

7. 正方形の表示時間を乱数で変更するようにして下さい。
8. hit 回数を表示するようにして下さい。
9. 3 回ミス hit したら、ゲームが終了するような状態遷移図を描いて下さい。
10. (3)で描いた状態遷移図を利用して、3 回ミス hit したら、ゲームが終了するようにして下さい。
11. hit 回数に応じて、正方形の表示時間が短くなるようにして下さい。
12. 単純な正方形ではなく、画像や別な形となるようにして下さい。
13. 繰り返しゲームが出来るようにし、開始画面にそれまでの最大 hit 回数が表示されるようにして下さい。
14. 効果音を付け加えて下さい。

今週は事前学習課題と宿題はありません。最終課題制作に力を入れて下さい。

来週以降の授業教室

7 月 19 日（火）	K1-1201 教室（いつものところ）
7 月 22 日（金）	K3-3506 教室（大きな教室、合同で実施）
7 月 26 日（火）	K1-1201 教室（いつものところ）
7 月 29 日（金）	K3-3506 教室（大きな教室、最終課題発表会）

サンプルプログラム：早撃ちゲーム

```
PFont font;
float xTarget;
float yTarget;
int targetWidth;
int targetHeight;
color targetColor;

final int GAME_READY = 0;
final int GAME_UPDATING = 1;
final int GAME_RUNNING = 2;
final int GAME_HIT = 3;

int currentState;
int time0_msec;

void startElapsedTime(){
  time0_msec = millis();
}

int elapsedTime_msec(){
  return millis()-time0_msec;
}

void setup(){
  size(400,400);
  smooth();
  font = createFont("Serif",48);
  currentState = GAME_READY;
}

void updateTarget(){
  targetWidth = 50;
  targetHeight = 50;
  xTarget = random(targetWidth, width-targetWidth);
  yTarget = random(targetHeight, height-targetHeight);
  targetColor = color(10,10,255);
}
boolean isOnTarget(int x,int y){
  if((xTarget <= x && x < (xTarget+targetWidth)) &&
     (yTarget <= y && y < (yTarget+targetHeight))){
     return true;
   }else{
     return false;
   }
}
void showReadyMessage(){
  textAlign(CENTER);
  textFont(font,48);
  fill(0);
  text("Start to click",width/2,height/2);
}
void showHitMessage(){
  textAlign(CENTER);
  textFont(font,48);
  fill(255,10,10);
```

```
56      text("Hit!!",width/2,height/2);
57    }
58    void showTarget(){
59      stroke(targetColor);
60      fill(targetColor);
61      rectMode(CORNER);
62      rect(xTarget,yTarget,
63          targetWidth,targetHeight);
64    }
65    void draw(){
66      background(255);
67      if(currentState == GAME_READY){
68        showReadyMessage();
69      }else if(currentState == GAME_UPDATING){
70        updateTarget();
71        currentState = GAME_RUNNING;
72        startElapsedTime();
73      }else if(currentState == GAME_RUNNING){
74        showTarget();
75        if(elapsedTime_msec() >= 1000){
76          currentState = GAME_UPDATING;
77        }
78      }else if(currentState == GAME_HIT){
79        showHitMessage();
80        if(elapsedTime_msec() >= 500){
81          currentState = GAME_UPDATING;
82        }
83      }
84    }
85    void mouseClicked(){
86      if(currentState == GAME_READY){
87        currentState = GAME_UPDATING;
88      }else if(currentState == GAME_RUNNING){
89        if(isOnTarget(mouseX,mouseY)){
90          currentState = GAME_HIT;
91          startElapsedTime();;
92        }
93      }
94    }
```

上のプログラムは、

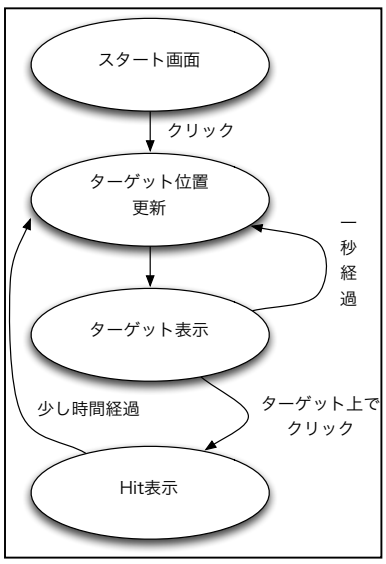
1) 最初は開始画面が表示する。

2) 表示されている正方形（ターゲット）をクリックしたら、hit 表示がされる。

3) hit 表示は少し経ったら(0.5 秒)、再び正方形（ターゲット）を表示する。

4) 正方形（ターゲット）は 1 秒間表示されたら、別な場所に移動する。

といった感じのゲームのようなプログラムです。



上の 1)から 4)から、このプログラムには、下の表のような 4

つの状態（状況）があると考えられます。マウスをクリックしたり、一定時間経ったら、これらの状態間を遷移します。状態から状態への遷移は、右の図のようになっています。このような状態間の遷移の状態を表す図のことを状態遷移図と呼んでいます。このような状態遷移図を作成すると、全体の進行状況を捉えやすくなります。このような状態遷移図をプログラム化するには、一般的に２つのやり方があります。

1) 表示や入力の処理をする関数内(draw などや mouseClicked 関数など)で、状態を条件分岐(if 命令など)で状態を変更する。
2) 状態遷移クラスを作成し、各状態を子クラスとして継承する。

このサンプルでは、1)の方法を利用してプログラムを作成しています。状態を表すために、各状態に数値を割り当て、状態を区別します。数字では、人間がどの状態かを識別ルことが難しいので、適当な変数名をつけた変数を利用しています。この値は変更さえることがないので、定数と呼ばれています。この状態を識別する定数として、上の表の状態定数名という名前の変数を使用しています。また、現在の状態は currentState という変数に保存されています。

状態名	説明	状態定数名
開始画面	開始画面を表示している	GAME_READY
ターゲット位置更新	正方形（ターゲット）の位置を変更している	GAME_UPDATING
ターゲット表示	正方形（ターゲット）を表示している	GAME_RUNNING
Hit 表示	Hit 表示を行っている	GAME_HIT

マウスをクリックして状態が変化する部分は、mouseClicked 関数で処理をしています。また、経過時間などで変化する部分は、draw 関数で処理をしています。

マウスクリック時の処理は、GAME_READY 状態であれば、直ちに GAME_UPDATE 状態に遷移します。また、GAME_RUNNING 状態であれば、マウスをクリックした場所が正方形（ターゲット）上であれば、GAME_HIT 状態に遷移します。経過時間を計るために、状態が遷移するときに、startElapsedTime 関数を呼び出し、elapsedTime_msec 関数で状態の経過時間をミリ秒単位で調べることが出来るようにしています。これらの処理は mouseClicked 関数内に書かれています。

GAME_UPDATE 状態であれば、正方形（ターゲット）の位置情報などを更新後、GAME_RUNNING 状態に遷移します。また、GAME_RUNNING 状態で、経過時間が 1 秒(1000 ミリ秒)経過したら、GAME_UPDATE 状態に遷移します。GAME_HIT 状態で、経過時間が 0.5 秒(500 ミリ秒)経過したら、GAME_UPDATE 状態に遷移します。これらの処理は、draw 関数内に書かれています。