コンピュータ基礎演習

第8回

理工学部 情報科学科 隅田 康明

sumida@ip.kyusan-u.ac.jp

今後の予定

- 第8回：条件分岐(2)、簡単なゲーム作成
- 第9回：繰り返し(1)、メソッドの宣言と呼び出し
  - 複数図形をメソッド化して繰り返し描画
  - 縦横に図形を敷き詰める
- 第10回：繰り返し(2)
  - 配列を使って沢山の図形を動かす
  - レポートとは別に、制作課題で何を作るか、を第11回開始までに提出する
- 第11回～第13回：制作課題の作成
  - 3回分かけてプログラムとレポートを作成
  - 第10回までのプログラムをアレンジして提出
  - 独自性の高いプログラムを作りたい場合は、設計書を提出すること

レポートの締め切り (減点なしで受け付ける期間)

- 第8回：第9回授業日の16時まで
- 第9回：第10回授業日の16時まで
- 第10回：第10回の週の金曜日18:00まで
- 第10回(制作課題の内容)：第10回の週の金曜日18:00まで
  - ベースにする回を書くだけ（第4回と第10回を合わせる、など）
    - これはMoodleで提出。未提出の場合は制作課題の最高点を制限。
  - 制作課題設計書：7/9(木)の23:59まで
    - 独自性の高い作品に挑戦する人のみ。Moodle＋メール提出。
- 制作課題：最終回授業日の18:00まで
- 追加課題：7月24日 18:00まで

- 遅れ提出の締め切り：7/24 18:00まで
  - Moodleでの提出も打ち切り
  - 以降は特別に認めた場合を除いて受け付けない
    - 余程の事情がない限りは認めない
  - 最終的なレポート締め切りと考えて問題ない

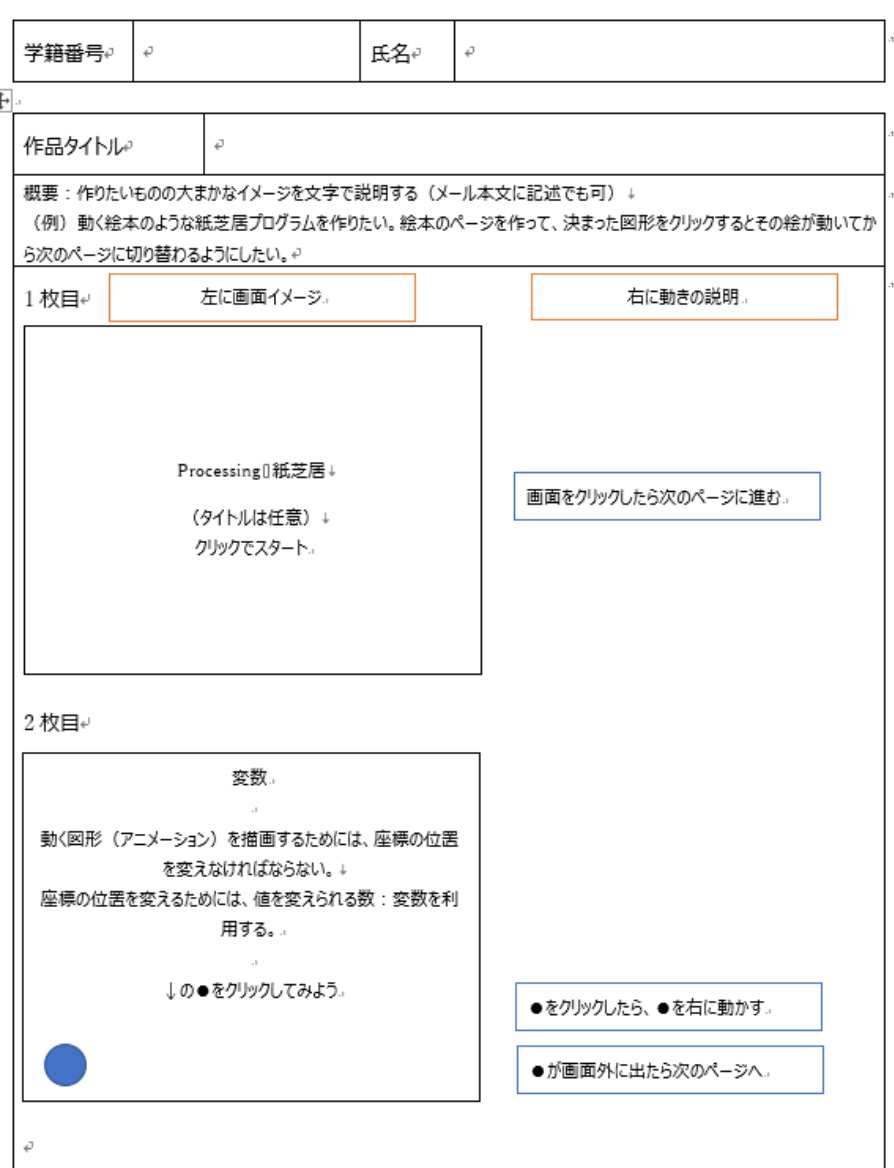
制作課題について

- 第11回～13回の3回でプログラムとレポートを作成
- 第10回までのプログラムのアレンジ
  - 追加課題のアレンジでも可
    - ただし、追加課題をアレンジする場合は、それぞれが別の作品とすること
  - 提出済みのレポートのアレンジでも可、
    ただし、元のプログラムからの変更点が少ない場合は、元のレポートを0点とした上で評価する
- 独自性の高いプログラム
  - 事前の設計書を必ず提出すること
    - 事前に設計書を提出すれば、事前にアドバイスを返す
  - 作れないものを作ろうとすると、単位を落とす危険がある
  - 最終的には自己責任だが、出来ないものを作ろうとしている場合は止めるので、その時は指示に従うこと

制作課題の設計書

- 手書きの紙を撮影した写真か、PDFファイルで提出
  - 写真を提出する場合は、なるべく歪みがないようにすること
    - 読めないものを送らない
  - PDFを作成するものは何でも良い
    - PDFにコメントを付けるので、編集制限は書けないこと
  - 設計書を提出＋設計書のイメージに近い作品を提出で、評点に+5点（それ自体が工夫点になるので）
    - 10回までのアレンジでも満点は取れます
      - ただし、工夫点を多くしないと難しい（独自作品でも同じですが）
- 設計書はメールで提出
  - 「PC基礎：制作課題設計書」の件名で送ること
    - 当日までに返信がなければ催促すること
      - メールで反応がなければOpenChatやZoomで

設計書の例



コンピュータ基礎演習□制作課題設計書

学籍番号		氏名	

作品タイトル	

概要：作りたいものの大まかなイメージを文字で説明する（メール本文に記述でも可）
（例）動く絵本のような紙芝居プログラムを作りたい。絵本のページを作って、決まった図形をクリックするとその絵が動いてから次のページに切り替わるようにしたい。

1 枚目　左に画面イメージ　右に動きの説明

Processing□紙芝居
（タイトルは任意）
クリックでスタート

画面をクリックしたら次のページに進む

2 枚目

変数

動く図形（アニメーション）を描画するためには、座標の位置を変えなければならない。
座標の位置を変えるためには、値を変えられる数：変数を利用する。

↓の●をクリックしてみよう

●をクリックしたら、●を右に動かす

●が画面外に出たら次のページへ

- これはWordで作った例
  - Wordのテンプレートは Moodleで配布
- 手書きでもOK
- PowerPointの場合
  - 1枚目：コンピュータ基礎演習制作課題設計書
  - 2枚目：作品タイトルと概要（説明文）
  - 3枚目～：画面のイメージと動きの説明

独自性の高い作品作りに挑戦する場合

- 作れそうにないものを作ろうとしない事
  - 完成しなければ、レポートを出せない　＝ 単位を落とす

- ある程度動く状態にまず持っていくことを優先する
  - 未完成でも一通りの動作が出来ていれば評価できる
  - こまめにバックアップを取っておくこと
    - Moodleに途中経過のレポートを出しておくと安全

- 妥協することも大事
  - まずはレポートを期限内に提出すること
  - こだわりたいなら、レポートを出してからでも良い
    - 2つ目を出しても評価はする

制作課題の評価

- 基本点(減点式)＋工夫点(加点式)で評価
- 基本点：最低基準を満たしていれば付く点数
  - 必要項目の不足や提出物の不足で減点する
- 工夫点：次のいずれかの工夫ごとに加点
  - プログラムの工夫
    - 変数、乱数、条件分岐、繰り返し、メソッド、座標変換、クラス、三角関数等の数式
  - 画像処理の工夫
    - アプリなどを使って画像を加工
      - どのアプリを使って、どんな工夫をしたのかを明記。どういった仕組みで加工されているのか、 Processingでも実現出来そうかの考察が必要
  - レポート自体の工夫
    - アニメーション、スライドマスタ、スライド背景にProcessingで作った画像を使うなど

追加課題について

- 第10回を目途にMoodleに資料アップロード
  - 動画までアップロードするかは未定
- いくつかの内容を出題するので、
  どれかをアレンジして提出すれば加点
  ※現時点の案で、すべての資料を用意するかは未定
  ※あくまでやる気のある学生向け
  - 3Dプログラミング
  - クラスとオブジェクト
  - 音との連動
  - 画像処理
- これの内一つを制作課題のベースとしても良い

遠隔授業期間中の質問

- **困ったら早めに質問・相談！！**
  - 学生側から質問されないと、
    誰が困っているのか、何が分からないのか、分かりません
- メール：やり取りに時間はかかるが一番確実
- Zoom：授業時間中限定
  - 時間は限られるが、作業中の画面を見ながら教えられるので、問題を短時間で解決出来る可能性が高い
- Line OpenChat：授業時間中限定
  - 文字だけのやり取りに限定（画像アップロードは禁止）
  - 質問内容が他の学生にも分かるので注意すること
    - プログラムの全文貼り付け等は厳禁！

授業についての質問メールについて

[授業名(曜日時限)]についての質問	件名
～先生	誰宛か
[授業名(曜日時限)]を受講しています、 20AA999の九産太郎です。	何者か
（質問内容） --	用件
20AA999 九産太郎 九州産業大学　芸術学部　○○学科 1年	署名

2日返信がなければもう一度送って下さい（なるべく見落としたくない）

レポートの実行画面について

- アニメーションを付けない
- **動画を貼り付けない**
  - Moodleに負荷がかかる

- 早すぎて上手く撮影できない・・・という場合は、
  前回のマウスクリックで停止・再生するプログラムを入れるか、
  frameRate(5);の様にしてFPSを下げましょう。

```
void setup(){
  size(300,300);
  frameRate(5);
}
```

プログラムスライドについて

- 動かないプログラムを送ってきても評価できません

- 動く状態になってから、プログラムをコピー、貼り付けましょう。

- **「全て選択」「Select All」** → コピー
  - マウスやタップで手動で選択しないように！
    - 何回も貼り付けをすると、文字が消える
  - PowerPointでプログラムを修正しないように！
    - 貼り付けた後は余計なことはしない

今回の授業内容

- 条件分岐（if文)
  - 条件によって動きを変えるプログラムを作る
    - 当たり判定を使ったゲーム作成
      - 単純な敵除けゲーム
- レポート内容
  - アレンジしたゲームを提出
  - 条件分岐を使ったプログラムを提出

- 少し注意：条件分岐は少し難しい
  - とにかく慣れるまでは、
    よく分からなくても動かして試すを繰り返しましょう
  - 深く理解できなくても、今は少し変えて動きが変わったでOK

if文：条件分岐

- 構文

```
if(条件) {
    条件を満たした場合の処理
}
```

比較演算子

数学の記号と同じではないことに注意

演算子	==	!=	>	>=	<	<=
意味	等しい	異なる	大きい	以上	小さい	以下

比較演算子を使った式を、論理式という

論理式とboolean型

- 論理式の結果は、正しい(真)か、正しくない(偽)か
  - 論理式が正しい（条件を満たしている）
    ⇒　結果は true
  - 論理式が正しくない（条件を満たしていない）
    ⇒　結果は false
- trueかfalseか、2つに1つ

- boolean型：true,falseを代入できる変数型

条件によって変数の値を変える

- もしも、 x が width より大きければ、xを0にする

```
if(x > width) {
  x = 0;
}
```

  - もしも、xが画面外に出たら、最初の位置に戻す

- もしも、 y が 0 より小さければ、yをheightにする

```
if(y < 0) {
  y = height;
}
```

  - もしも、yが画面外に出たら、最初の位置に戻す

if-else if- else文:複数の条件分岐

- 構文

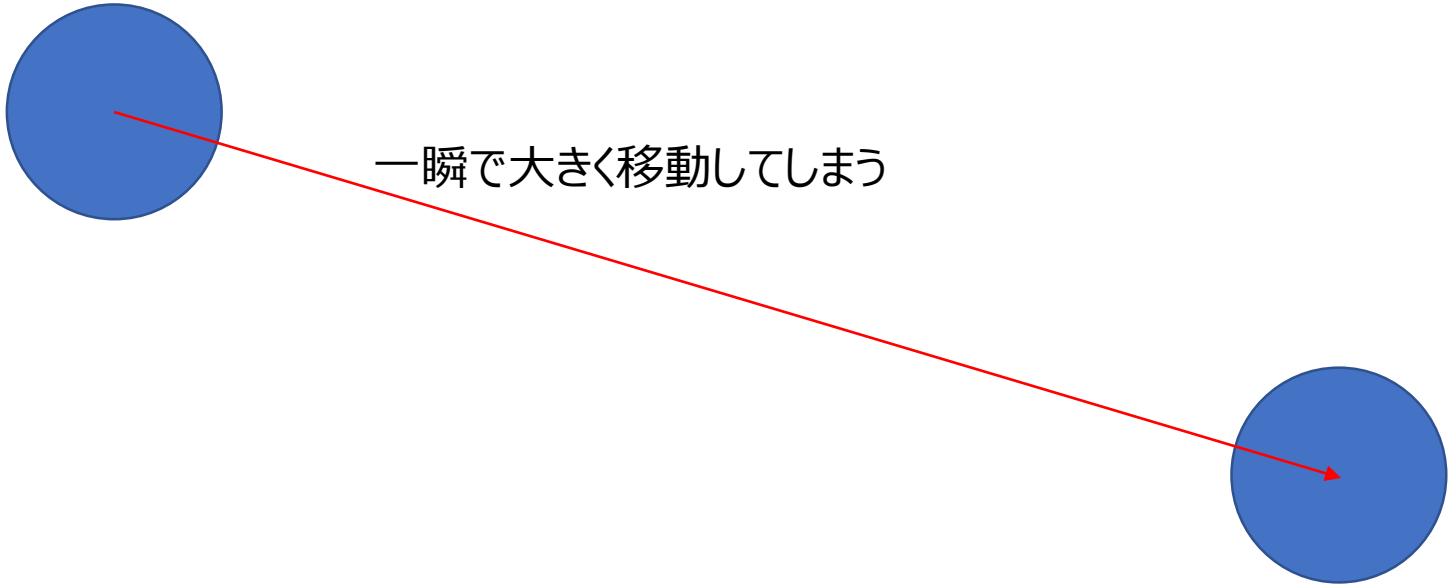
```
if(条件１) {
    条件１を満たした場合の処理;
} else if(条件２){
    条件１を満たさずに、
    条件２を満たした場合の処理;
} else {
    どの条件も満たさない場合の処理
}
```

if文を書く時の注意

- **() {} は必ず開いたら閉じる**
- **閉じすぎにも注意**
- **カッコの対応が付いているか、しっかり確認！！**

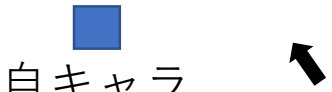
条件分岐を使った図形のマウス移動

- 自キャラの操作をマウスで行う
  - ellipse(mouseX, mouseY, 10, 10);
  - これだと、キャラクターが一瞬で大きく動いてしまう
    - 移動ではなく瞬間移動になる



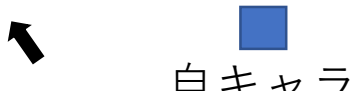
自キャラがマウスに追従して動くプログラム

- もし、マウスが自キャラのx座標よりも右にあれば、
  自キャラの横方向の移動距離を 1 にする



マウスカーソルが右にある　＝　右に移動させる
(座標を+1)

- もし、マウスが自キャラのx座標よりも左にあれば、
  自キャラの横方向の移動距離を -1 にする



マウスカーソルが左にある　＝　左に移動させる
(座標を-1)

単にマウス座標に移動させるのではなく、マウス座標で移動方向を決める

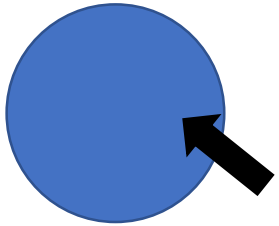
- 自キャラの座標を (x, y) とする

```
//自キャラの移動
 if (x < mouseX) {
   x = x+1;
 } else if (x > mouseX) {
   x = x-1;
 }
 //y方向についても同様の処理を記述
 //ひな形プログラムを各自で埋めること

 fill(255, 0, 0);
 ellipse(x, y, 20, 20); //自キャラの描画
```

条件分岐を使った当たり判定

- もしも、クリックした座標が、図形に近ければ・・・



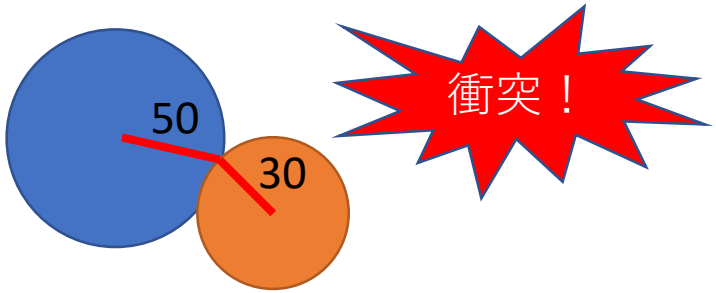
クリックしたら動き出す　等

- 図形の座標と、マウスの座標の距離を計算
  - distメソッド:2点間の距離を求める

```
float d = dist(x, y, mouseX, mouseY);
if( d < 50) { //もしも距離が50(半径)未満なら
}
```

条件分岐を使った当たり判定

- もしも、2つの座標が近ければ・・・

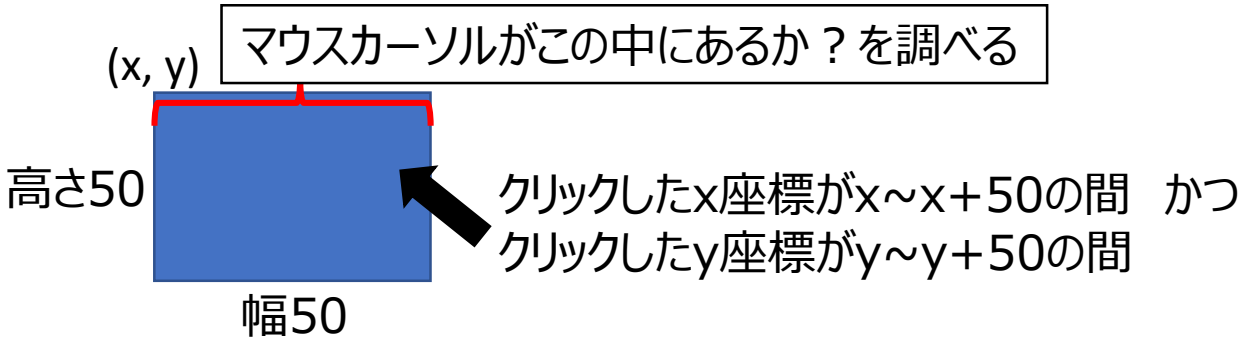


- 図形1の座標と、図形2の座標の距離を計算
  - distメソッド:2点間の距離を求める
  - 2つの円の中心間の距離が、円1の半径+円2の半径未満

```
float d = dist(x, y, tekiX, tekiY);
if( d < (50 + 30) {
}
```

条件分岐を使った当たり判定(四角形)

- もしも、クリックした座標が、図形に近ければ・・・

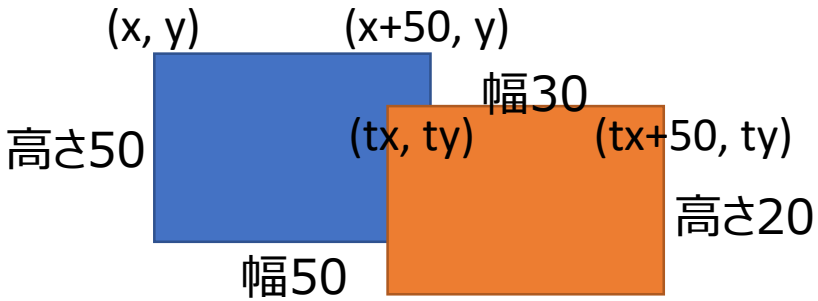


- マウスの座標が矩形の中にあるかを判定

```
if(x < mouseX && x + 50 > mouseX &&
   y < mouseY && y + 50 > mouseY) {
  //当たった時の処理
}
```

長方形同士の当たり判定

- もしも、2つの図形が接触していれば・・・



- 四角形１の頂点が四角形２の中にあるかを判定

```
if ((x+50) > tx && x < (tx+30)) {
    if ((y+50) > ty && y < (ty+20)) {
      //衝突時の処理
    }
  }
```

独自メソッドの利用

- 雛型A・Bの衝突判定は、processingのデフォルトにはない独自メソッド
  - 独自メソッドの作り方は次回以降
    - 予習資料として返り値のないメソッド宣言は今回資料に入れている

```
/*(長方形1のx, y, 幅, 高さ, 長方形2のx, y, 幅, 高さ)
を引数として渡すと、長方形同士の当たり判定結果を返す
長方形同士の衝突時に、長方形1(自キャラ)の下辺が長方形2の上辺よりも
下にいるときに当たった場合には衝突していないとする独自処理を追加している
*/
boolean checkCollision2(float x1, float y1, float w1, float h1, float x2, float y2, float w2, float h2) {
  boolean hit = false;
  if ((x1+w1) >= x2 && x1 <= (x2+w2)) {
    if ((y1+h1) >= y2 && y1 <= (y2+h2)) {
      if ( (y1+h1) <= (y2+h2/2)) {
        hit = true;
      }
    }
  }
  return hit;
}
```

雛型Bの衝突判定処理
長い処理を全部の図形に書いていたら大変なので、
処理をまとめて実行できるようにメソッドを作っておく

メソッドの利用

- **メソッド**とは、複数の処理をまとめたもの
  - 例えば、長方形を描く rect() のメソッドは、線を4本描く命令が集まったもの
  - dist()メソッドは、2点間の距離を計算して返してくれる
    - 内部では三平方の定理で距離を計算している
    - 計算式を知らなくても、distの使い方を知っていれば距離を計算できる！！
- **メソッド内の処理は分からなくても、使い方と得られる結果が分かっていれば利用出来る**
  - 便利な道具は利用すれば良い
    - ただし、人が作ったものを自分の成果にしては駄目
  - 作ることは、次回以降で学習
  - 今回は、標準以外のメソッドを使うことに慣れよう

当たり判定メソッド(円同士)

boolean checkCollision
(float x1, float y1, float w1, float x2, float y2, float w2)

- checkCollision
  - 返り値：boolean型
    - 当たっていればtrue, そうでなければfalse
  - 引数
    - x1:円1のx座標
    - y1:円1のy座標
    - w1:円1の幅
    - x2:円2のx座標
    - y2:円2のy座標
    - w2:円2の幅

当たり判定メソッド(長方形同士)

boolean checkCollision2
(float x1, float y1, float w1, float h1, float x2, float y2, float w2, float h2)

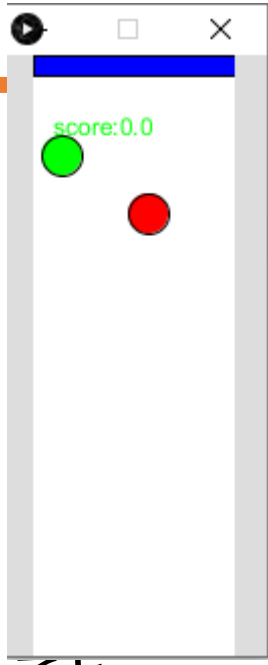
- checkCollision2
  - 返り値：boolean型
    - 当たっていればtrue, そうでなければfalse
  - 引数
    - x1:長方形1のx座標
    - y1:長方形1のy座標
    - w1:長方形1の幅
    - h1:長方形1の高さ
    - x2:長方形2のx座標
    - y2:長方形2のy座標
    - w2:長方形2の幅
    - h2:長方形2の高さ

今回の雛形プログラムについて

- ダウンロードはMoodleから
  - 単純なゲームでもそれなりに長いプログラムになる
- まずは雛型Aを基本に考えること
  - 雛型Bは少し複雑で難しい、雛型Cはゲームではない
- 元々書いてあるプログラムを消さない
  - 敵キャラ、足場のプログラムは除く

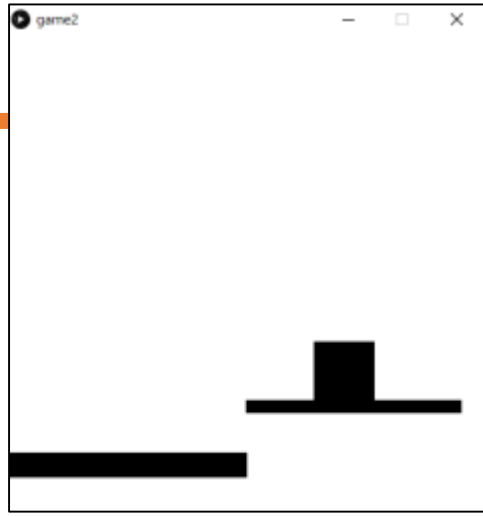
- コピー＆ペーストを上手く使っていくこと
  - 敵や足場のプログラムは、コピー＆ペーストで少し書き換えればどんどん増やせる
  - ただし、括弧のコピーし忘れなどには注意すること

雛型プログラムA



- 敵避けゲーム
  - 赤丸が自キャラ、緑丸が敵キャラ
    - 円と円の当たり判定を利用
  - 敵を避けてゴール（画面上）を目指す
  - 雛型は敵1体（緑丸）との当たり判定のみ
    - まず、自キャラを縦方向にも動かせるようにすること
    - 動く敵を増やしてアレンジ
    - 少し工夫するとシューティングゲームにも出来る
      - クリック（タップ）で弾発射など
      - 興味があるなら、制作課題で挑戦してみましょう

雛形プログラムB



- 横方向アクション風
  - 雛型Aよりも複雑
    - 難しければ雛型Aを選択すること
  - 四角と四角の当たり判定を利用
  - マウスクリックでジャンプする
  - 衝突判定用のメソッドを用意済み
    - 当たり判定後の処理が少し難しいため
  - まずは、自キャラの動きを作る
  - 条件分岐で動く足場を追加する
  - 動く敵などを追加するなどしてアレンジしてみる

雛型プログラムC

- 雛型A、Bがまともに動かない場合に選ぶこと
  - 旧世代iPhoneSEでも動いてるので大丈夫とは思いますが・・・
- カーソルを合わせると色・形・動きが変わる図形を描画
  - ある図形にカーソルを合わせると、
    色が変わる、別の図形が動き始める、背景が変わるなど

- 評価は他と同じ
  - 条件分岐で動く図形を作ることに慣れるのが目的

PowerPointでレポート作成

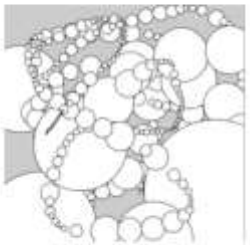
- 1枚目：タイトル

PC基礎 05

学籍番号　氏名

- 2枚目：実行画像

タイトル：（作品のタイトル）



- 説明
  - ～～をイメージして作った
  - マウスの動きに合わせて、丸の大きさが変わるように工夫した

どんなものを作ろうとしたのか？

どんな工夫をしたのか？

簡単でいいので説明を書く

- 3枚目：プログラム

プログラム

プログラムを全て貼り付ける

```
void setup(){
  size(400,400);
  colorMode(HSB,360,100,100,100);
}
void draw(){
  float d = dist(mouseX, mouseY, pmouseX, pmouseY);
  ellipse(mouseX,mouseY,d,d);
}
```

貼り付けたときに、文字が見えない程小さくなっても構わない

これは第3回のレポートの例

実行画面やプログラムは、
（当然）今回の内容にすること

必要事項が揃っていれば、レイアウトは自由

レポートチェックリスト（第8回）

- ミニテストを受験した(レポートの前と後)

- 雛型のプログラムをアレンジした
  - 一部のプログラムを埋めて動くようにした
  - 条件分岐で動く図形を2つ以上追加した
  - 上記の図形に自キャラとの当たり判定を追加した
- PowerPointでレポートを作成した
  - タイトル、作品紹介(工夫点)、プログラムの3枚
- Moodleでレポートを提出した

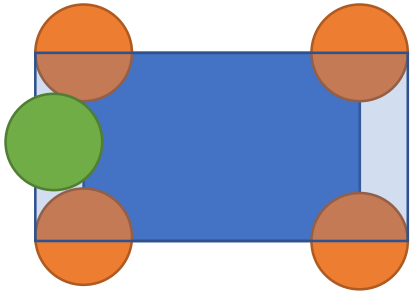
加点項目

- 障害物となる動く図形を多く作った
  - 変数をたくさん使っている
  - 条件分岐を使っている
- 自作メソッドでキャラクターを作って動かした
  - 円の当たり判定を使う
- 複数の雛型でレポートを作って提出した
  - 雛型1つでも工夫点が多ければ満点にはなります
  - 全部触ってみたい場合は、少しずつのアレンジしてみよう
- 丸と四角の当たり判定を取り入れた
  - 2枚目のスライドに工夫点として書いておくこと

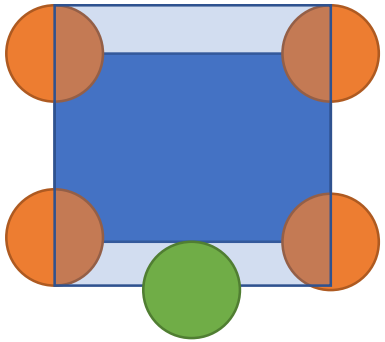
丸と四角の当たり判定

条件１、条件２、条件３の順番に判定して、いずれかを満たしていれば衝突している

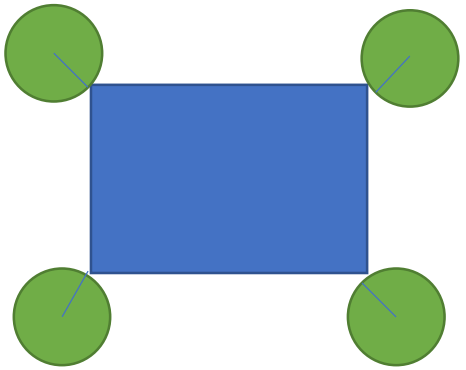
- 条件1
  - 長方形の幅+円の半径の長方形内に、円の中心が入っていれば衝突



- 条件2
  - 長方形の高さ+円の半径の長方形内に、円の中心が入っていれば衝突



- 条件3
  - 長方形のいずれかの頂点と、円の中心の距離が、円の半径未満なら衝突



Moodleにメソッドを公開
利用方法はコメントを読んで考えること

補足資料
自作メソッドの作成

自分で作ったキャラクターを動かす

メソッド

- メソッドの構造
  - メソッドの名前(引数1,引数2,…);
    size(400,400);
    ()の中に書くものは命令によって違う
  - 引数がないメソッドもある
    noStroke();
    noFill();

- size(400,400); や size(400,400); のように、
  メソッドをプログラムに書いて実行することを、
  メソッドを呼び出すという

メソッドを自分で作る

- メソッド：複数の処理をまとめて１つの命令にする機能
    - 関数ともいう
  - メソッドの名前(値1,値2,…);　の構文
  - setup()、draw(),line(),rect(),ellipse()など

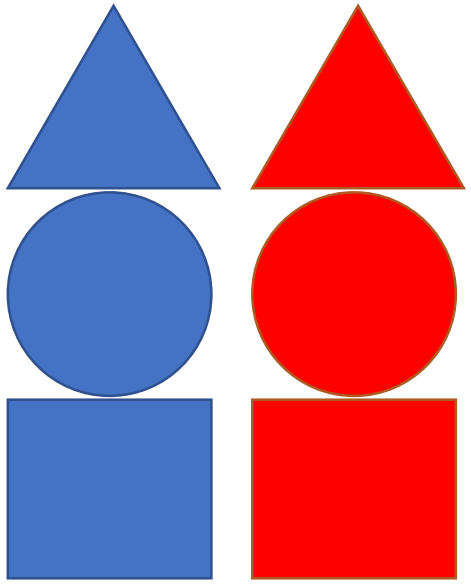
- 複数の命令を１つにまとめたメソッドを自分で作れる

厳密な定義では、メソッドと関数は違うものだが、機能としては同じ

メソッド：処理をまとめて名前をつけた物

青ペンを持つ
三角形を描く
円を描く
四角形を描く

赤ペンを持つ
三角形を描く
円を描く
四角形を描く



```
void oden() {
  三角形を描く
  円を描く
  四角形を描く
}
```

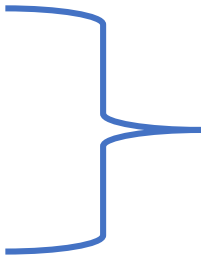
```
fill(0,0,255);
oden();
fill(255,0,0);
oden();
```

色を変える度に同じ命令を書かなくてもよくなる

メソッドの作り方、呼び出し方

- メソッドの作り方

```
void メソッド名(引数1,引数2,引数3,…) {
    処理１；
    処理２；
    処理３；
}
```



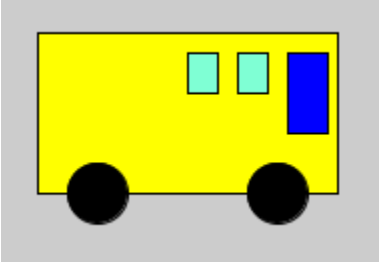
まとめたい処理

- メソッドの呼び出し方
  - setupやdrawの中で、
    メソッド名(引数1,引数2,引数3,…);

オリジナルの図形を描くメソッドを作る

- 大きさの変更は今回はなし
  - 挑戦したい人はしましょう
- 位置だけ指定出来るメソッドを作る

- 例として、こんなのを作ってみる



メソッドを宣言

```
void setup(){
 size(400,400);
}
void draw(){
  background(255);
}
void bus(float x, float y){

}
```

- メソッドの名前は自分で作る図形に合わせて変えること

メソッド呼び出し

```
void setup(){
 size(400,400);
}
void draw(){
  background(255);
  bus(100,50);
}
void bus(float x, float y){


}
```

宣言したものと同じ名前、同じ引数の数

まだ何も書かれない(何も書いていないから)

図形を描いていく

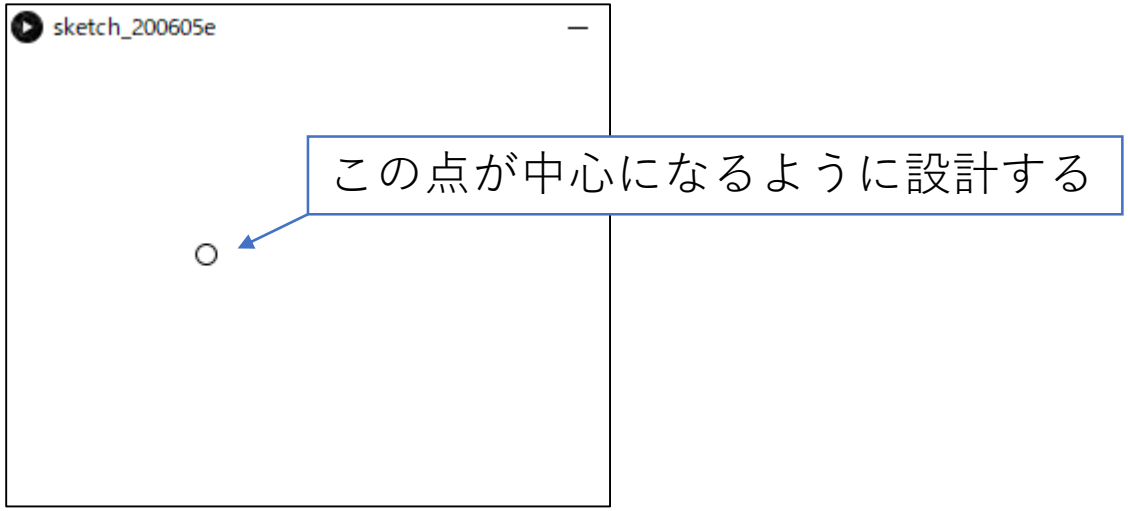
- まず、中心点を描画（ガイドとして描き、後で消す）

```
void bus(float x, float y){

ellipse(x, y, 10, 10);
}
```

中心点（x, y) を変えないこと

中心点（x, y) の座標は、メソッド呼び出しで入力した引数,つまり(100,50)



中心点を描くだけのメソッド（今の状態）

数値を2つ貰ったら、その場所に図形を描いてくれる

busさん

(中心x, 中心y)

(100, 50)

描いといて。
(x, y) の中身は自分で見てね

(x, y, 10, 10)

ellipseさん

(中心x,中心y,幅,高さ)

(x, y)を中心に、幅10,高さ10・・・
xには100が、 y には50が入ってるから
(100, 50) の位置に幅10,高さ10の丸を描きます

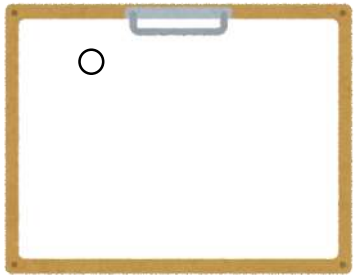
busの設計図

ellipse(x, y, 10, 10);

メソッド内だけで
共有で使える変数になる
（メソッドの外では使えない）

何の図形を描くかは、メソッド内の命令によって変わる

busさんの場合は、(x, y)の位置に幅10,高さ10の円を描く



設計図の中の命令が増えても、基本は同じ

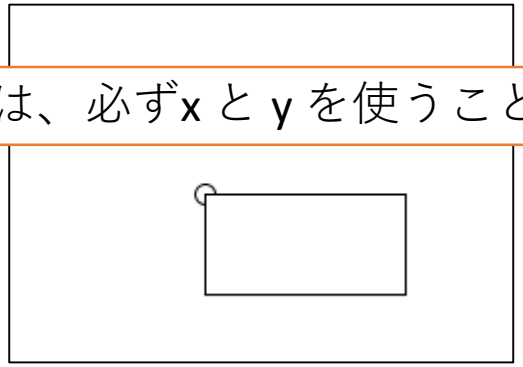
中心点を中心に設計(1)

- 中心点が中心になるように図形を配置していく

```
void bus(float x, float y){
  rect(x, y, 100, 50);
  ellipse(x, y, 10, 10);
  }
```

中心点は最後に描く

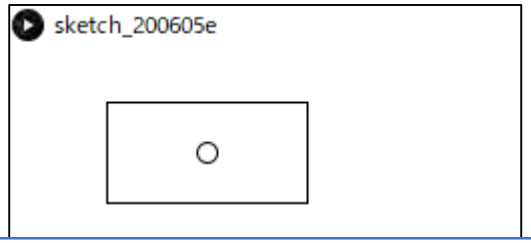
x座標、y座標を指示する引数には、必ずx と y を使うこと



上手くいかないので修正

```
void bus(float x, float y){
  ellipse(x, y, 10, 10);
  rect(x-50, y-25, 100, 50);

}
```



中心点が中心になったら、
次の図形へ

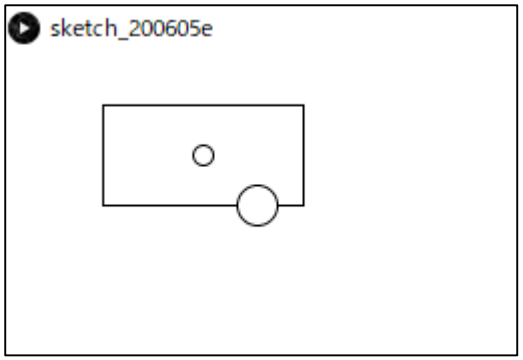
中心点を中心に設計(2)

- 微調整しながら図形を配置していく

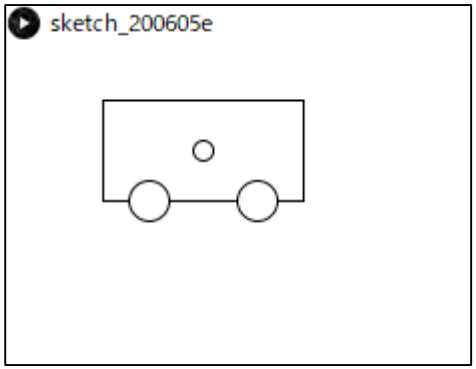
```
void bus(float x, float y){
  rect(x-50, y-25, 100, 50);
  ellipse(x+27, y+25, 20, 20);
  ellipse(x, y, 10, 10);
}
```

中心点は最後に描く

x座標、y座標を指示する引数には、必ずx と y を使う



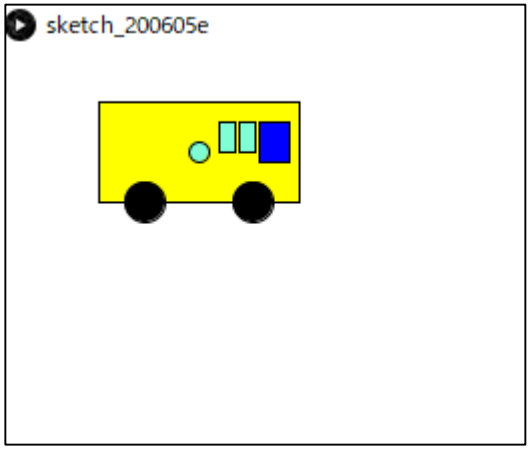

```
void bus(float x, float y){
  rect(x-50, y-25, 100, 50);
  ellipse(x+27, y+25, 20, 20);
  ellipse(x-27, y+25, 20, 20);
  ellipse(x, y, 10, 10);
}
```



色付き図形を配置していく

- 色を付け、図形を配置、を繰り返して動かす図形を作る

```
void bus(float x, float y) {
  fill(255, 255, 0);
  rect(x-50, y-25, 100, 50);
  fill(0, 0, 0);
  ellipse(x+27, y+25, 20, 20);
  ellipse(x-27, y+25, 20, 20);
  fill(0, 0, 255);
  rect(x+30, y-15, 15, 20);
  fill(127, 255, 212);
  rect(x+20, y-15, 8, 15);
  rect(x+10, y-15, 8, 15);
  ellipse(x, y, 10, 10);
}
```



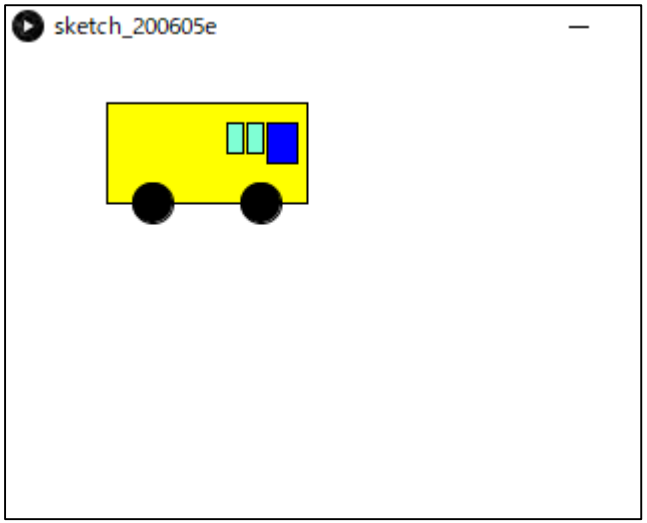
x座標、y座標を指示する引数には、必ずx と y を使う

中心点は最後に描く、
中心点の座標は(x, y)のまま変更しない

中心点が中心になるように配置すること（円同士の衝突判定が使える）

中心点をコメントアウトして完成

```
void bus(float x, float y) {
  fill(255, 255, 0);
  rect(x-50, y-25, 100, 50);
  fill(0, 0, 0);
  ellipse(x+27, y+25, 20, 20);
  ellipse(x-27, y+25, 20, 20);
  fill(0, 0, 255);
  rect(x+30, y-15, 15, 20);
  fill(127, 255, 212);
  rect(x+20, y-15, 8, 15);
  rect(x+10, y-15, 8, 15);
  //ellipse(x, y, 10, 10);
}
```

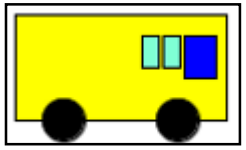


後で確認に使えるので、コメントで残しておく

// の後に書かれた文字は、実行されない
実行しない命令を一時的に消したり、メモを書くときに使う

四角いキャラクターを使いたい場合

- 中心点ではなく、
  キャラクター全体を囲う長方形を書いておき、
  それをガイドにする
  - 長方形同士の当たり判定が使えるようになる
- この場合はガイドの左上の頂点を(x,y)とすること



作ったメソッドで描いた図形を動かす

- 横に動かしたい場合（前回と同様の手順）

```
void setup() {
  size(400, 400);
}
void draw() {
  background(255);
  bus(100, 50);
}
void bus(float x, float y) {
  //バスを描く処理
}
```

ここを変えると左右に動く

ここは各自で作ったメソッド

作ったメソッドで描いた図形を動かす

- 横に動かしたい場合（前回と同様の手順）

```
float x = 100;
void setup() {
  size(400, 400);
}
void draw() {
  background(255);
  bus(100, 50);
}
void bus(float x, float y) {
  //バスを描く処理
}
```

変数を宣言、初期値を設定（同じにする）

ここは各自で作ったメソッド

作ったメソッドで描いた図形を動かす

- 横に動かしたい場合（前回と同様の手順）

```
float x1 = 100;
void setup() {
  size(400, 400);
}
void draw() {
  background(255);
  bus(x1, 50);
}
void bus(float x, float y) {
  //バスを描く処理
}
```

変えたい数値の場所に変数を入れる

ここは各自で作ったメソッド

作ったメソッドで描いた図形を動かす

- 横に動かしたい場合（前回と同様の手順）

```
float x1 = 100;
void setup() {
  size(400, 400);
}
void draw() {
  background(255);
  bus(x1, 50);
  x1 = x1 + 1;
}
void bus(float x, float y) {
  //バスを描く処理
}
```

変数に数値を足して増やす(x座標が大きくなって右に動く)

ここは各自で作ったメソッド

ありそうな質問

- Q:左右反転できないの？
  - 右用と左用を作るのは面倒
- A:ちょっと面倒だけどできます

```
translate(x2,y2); //書きたい座標にする
scale(-1,1); //左右反転させる
bus(0,0); //座標は(0,0)にする
resetMatrix(); //動かしたり反転した座標系を元に戻す
```

  - 詳しい説明は割愛。気になるなら聞いてください。
    - 座標系に関しては第8回か9回の内容