# 実験の目的

何をすることを目的として実験を行ったのかを短くまとめて書く。今回の実験の場合には、「～を行うプログラムを作成する。」などのように、どんなプログラムを作成することが目的であったのかを記述するとよい。

# 名前の描画方法

自分の名前を描画する方法を詳しく説明する。

* どのように名前を描画しようと考えたか。自分のオリジナルな部分が明確に分かるように書く。何かイベントがあったこと(例えば、時間変化やキーボード入力・マウスクリッック) で画像を変化させたような場合は、どのようなイベントに対して色や形などをどのように変えることにしたかを記述する。
* どのようにしてそれをプログラムとして実装したかについても説明する。プログラム中でどのようなデータをどのような形(変数・配列) で持ったか、どのような関数を使用して意図する図形を描画したか、動きなどがある場合はイベントに対してそれをどのように変化させたかなどについて説明する。

などについて述べる。図なども必要に応じて入れるとよい。

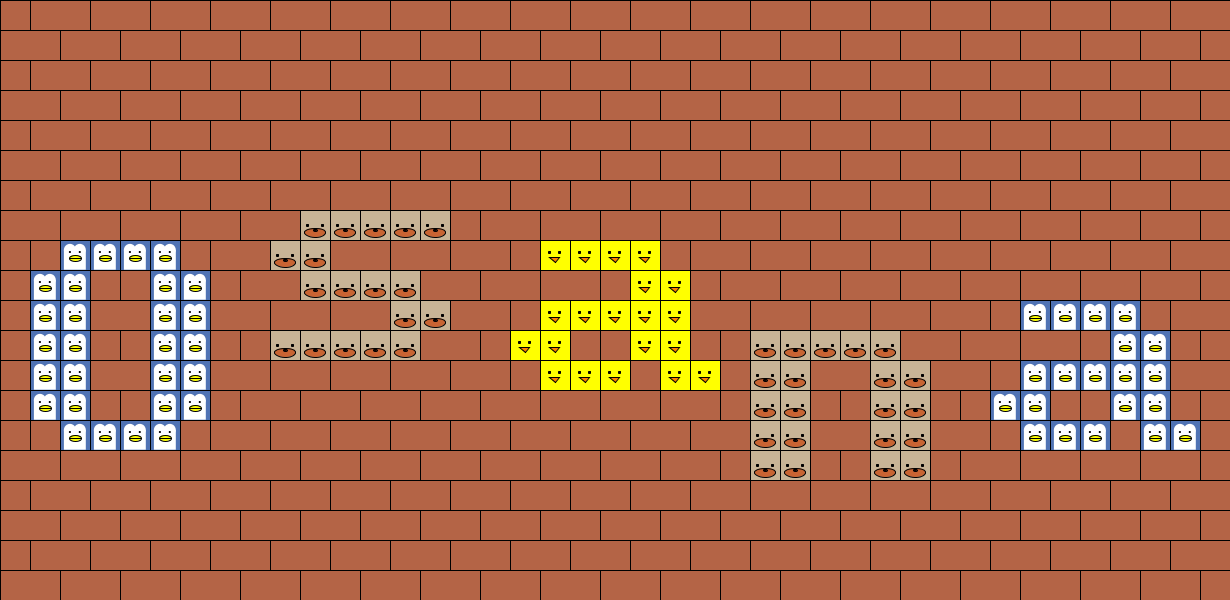


図 1　名前の描画方法

# ソースコード

作成したプログラムのソースコードをグローバル変数の宣言および関数ごとに節(サブセクション) に分けて載せる。ソースコードには、教科書のソースコードのようにコメント(プログラムの説明) をきちんと書き、第三者が見ても何をやっているのか分かるようにする。特に

* 各変数は何に用いるものなのか（関数の引数のほか，関数内で定義されるローカル変数についても）
* 各関数や処理のブロックは何を行うものなのか
* 各関数の戻り値はどういう意味・様式のデータか

などについては丁寧にコメントをつけるようにする。

　また、ソース全体に対してインデント（字下げ）を適切に行う。Processingでは、Edit 🡪 Auto Format （または Ctrl+T）でプログラム全体に対して自動的にインデントを整えることができるようになっている。

例えば以下のように説明していく。

## グローバル変数

プログラム 1にグローバル変数の宣言部分のソースコードを示す。ここでは、～を行っている。

このプログラムの中では～。

プログラム 1 グローバル変数

1. // グローバル変数
2. int gSize = 30; // 格子のサイズ
3. // 動物ブロックのサイズはgSize×gSize
4. // 煉瓦ブロックのサイズは(gSize\*2)×gSize
5. int gridW = 41; // 横方向の格子の数
6. int gridH = 20; // 縦方向の格子の数
7. int count; // 落下中のブロックの段数をカウント
8. int animalNum = 4; // 動物ブロックの種類
9. int[][] block = new int[gridH][gridW]; // ブロックの種類
10. // 0 : 2マス分の煉瓦ブロックの右側
11. // 1 : 1マス分の煉瓦ブロック
12. // 2 : 2マス分の煉瓦ブロックの左側
13. // 3 : くまブロック
14. // 4 : ペンギンブロック
15. // 5 : ねこブロック
16. // 6 : あひるブロック
17. float[][] y = new float[gridH][gridW]; // 格子の各マスのy座標
18. // 横長の煉瓦ブロックの場合は
19. // 右側のマスに相当する部分の値は
20. // 使用しない
21. float[][] dy = new float[gridH][gridW]; // 格子の各マスのy座標の変化量
22. // 横長の煉瓦ブロックの場合は
23. // 右側のマスに相当する部分の値は
24. // 使用しない

## setup()関数

プログラム 2にsetup()関数のソースコードを示す。setup()関数の中では、～を行っている。

ここでは、～。

プログラム 2　setup()関数

1. // setup()関数
2. void setup() {
3. size(1230, 600); // 1230×600のウィンドウを生成
4. // 1230 (=gSize\*gridW)
5. // 600 (=gSize\*gridH)
6. count=-1; // 落下中のブロックの段数countを-1に設定
7. initBlock(); // ブロックの初期化
8. }

## draw()関数

プログラム 3にdraw()関数のソースコードを示す。draw()関数の中では、～を行っている。ここでは、～。

プログラム 3　draw()関数

1. // draw()関数
2. void draw() {
3. background(255); // 背景を白で塗りつぶす
4. // 新しい段のブロックの落下開始
5. if(count<gridH-1){ // countがgridH-1未満ならば
6. if(frameCount%100==0){ // 100フレームごとに次の段の落下を開始
7. count++; // countを1増やして次の段の処理を開始
8. }
9. }
10. // 座標の更新
11. if(0 <= count){ // countが0以上ならば
12. // (最初の100フレームはcountが-1なので何も起こらない)
13. for(int j = 0; j < gridW; j++){ // 1段(gridW個)分繰り返す
14. y[count][j] += dy[count][j]; // count段目のブロックjのy座標を更新
15. if(y[count][j] > height-gSize\*(count+1)){
16. // count段目のブロックjのy座標が最終的な位置
17. // height-gSize\*(count+1)よりも下になったら
18. y[count][j]= height-gSize\*(count+1);
19. // count段目のブロックjのy座標を最終的な位置
20. // height-gSize\*(count+1)に設定
21. dy[count][j] = 0; // count段目のブロックjのy座標の変化量を0に変更
22. }
23. }
24. }
25. // ブロックの描画
26. for(int i=0; i<=count; i++){ // 0～count段目について
27. for(int j=0; j<gridW; j++){ // 1段(gridW個)分繰り返す
28. switch(block[i][j]){ // i段目のj番目のブロックの状態に応じて描画
29. case 1 : // 1マス分の煉瓦ブロック
30. fill(180, 100, 70); // 塗りつぶし色を(180, 100, 70)に設定
31. rect(j\*gSize, y[i][j], gSize, gSize);
32. // (j\*gSize, y[i][j])を左上とするgSize×gSizeの矩形を描画
33. break;
34. case 2 : // 2マス分の煉瓦ブロック
35. fill(180, 100, 70); // 塗りつぶし色を(180, 100, 70)に設定
36. rect(j\*gSize, y[i][j], gSize\*2, gSize);
37. // (j\*gSize, y[i][j])を左上とする(gSize\*2)×gSizeの矩形を描画
38. break;
39. case 3: // くまブロック
40. drawBear(j\*gSize, y[i][j]);
41. // (j\*gSize, y[i][j])を左上とする位置にくまブロックを描画
42. break;
43. case 4: // ペンギンブロック
44. drawPenguin(j\*gSize, y[i][j]);
45. // (j\*gSize, y[i][j])を左上とする位置にペンギンブロックを描画
46. break;
47. case 5: // ねこブロック
48. drawCat(j\*gSize, y[i][j]);
49. // (j\*gSize, y[i][j])を左上とする位置にねこブロックを描画
50. break;
51. case 6: // あひるブロック
52. drawDuck(j\*gSize, y[i][j]);
53. // (j\*gSize, y[i][j])を左上とする位置にあひるブロックを描画
54. break;
55. }
56. }
57. }
58. }

## ～を行う関数　xxxxx()

プログラム 4に～を行う関数xxxxx()のソースコードを示す。この関数では、～を行っている。

* + これ以降の部分は人によって違うはず

プログラム 4 xxxxx()関数

1. // ～を行う関数
2. void xxxxx(){
3. ????
4. }

# 実行結果

プログラムを実行した結果を画像として保存し、実行結果として載せる。作品が時間的に変化

するようなものの場合は、画面の変化したところで数種類の画像を保存し、説明を載せるとよい。

図を載せるだけでなく、実行結果に関する説明も記述する。

例えば、以下のように説明する。

作成したプログラムの実行結果を図 2～4に示す。このプログラムでは、～。

* + この例では図をかなり小さくしているが、実際には必要な情報が見えるように

図の大きさは適切に調整する。

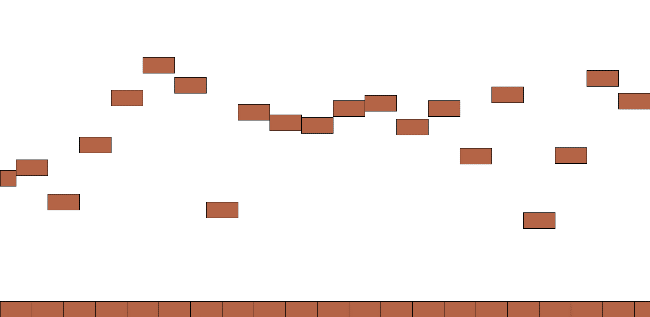


図 2　実行結果 (215フレーム目)

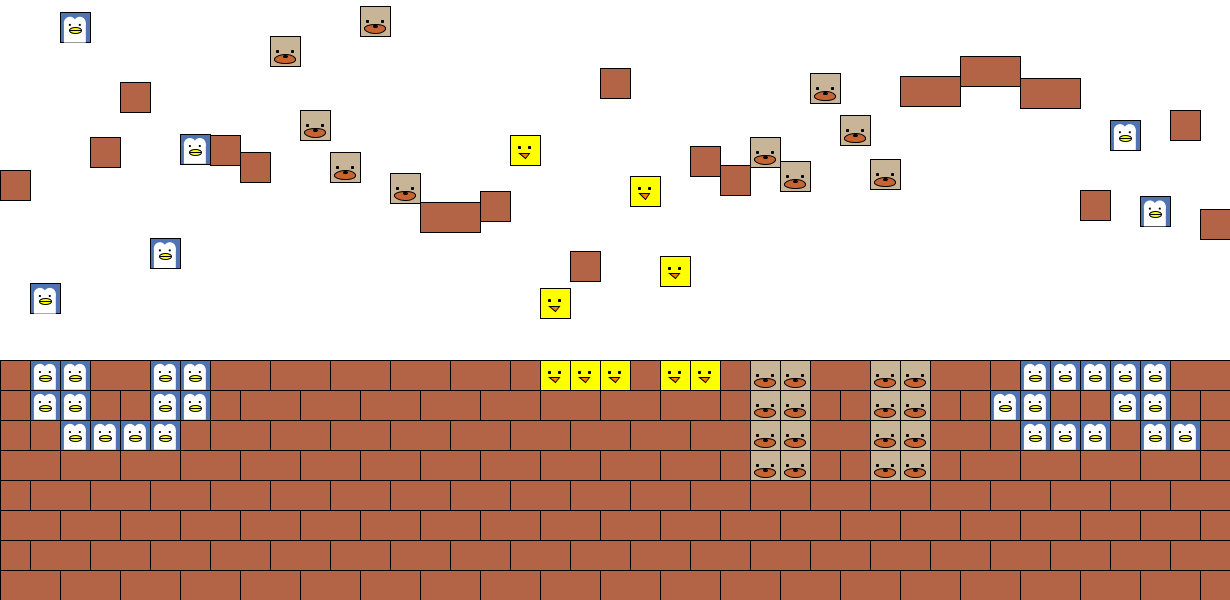


図 3　実行結果 (911フレーム目)

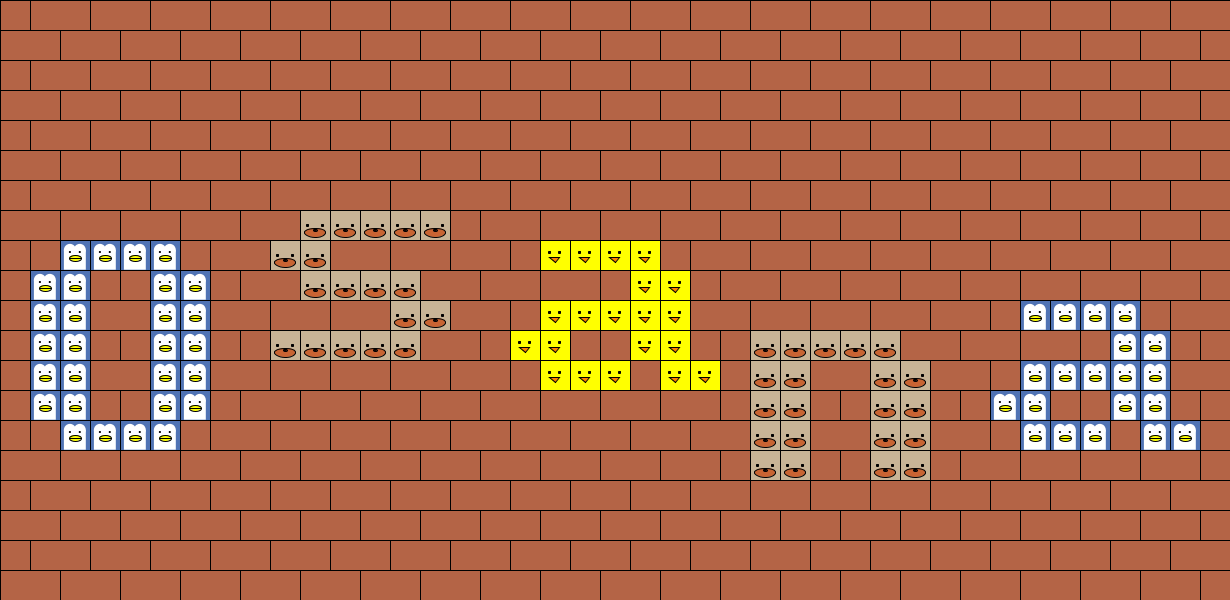


図 4　実行結果 (2130フレーム目)

# 工夫した点・苦労した点・作成時間

名前のデザインや表現方法・プログラムを作成する際に工夫した点や苦労した点などについて

述べる。デザインを考える(構想を練る) のにかかった時間、プログラムの作成にかかった時間に

ついて記述する。なお、時間は正確な時間でなく、おおよその時間で構わない。

## 工夫した点

プログラムを作成する際に以下の点を工夫をした。

* ～した点
* ～した点

## 苦労した点

　～をするのに苦労した。

## 作成時間

　～をするのに～時間程度かかった。

# 可能であれば実現したかったこと

今回の課題では実現できなかったが、時間や技術があれば取り組んでみたかったことについて

述べる(Processing の機能としてこのようなものがあればこんなことをしてみたかった…というようなことでもよい)。

参考文献

1. 長名, 石畑, 菊池, 伊藤 : つくって学ぶProcessing入門, コロナ社, 2017.
2. “Processing”, https://processing.org, （参照 2016-06-27）