# ドット絵でプログラミング！
# 難解言語『Piet』勉強会

KC3 2017 勉強会
KMC　base64 (@basemusi)

突然ですが皆さん、

# あの！話題の画期的なプログラミング言語

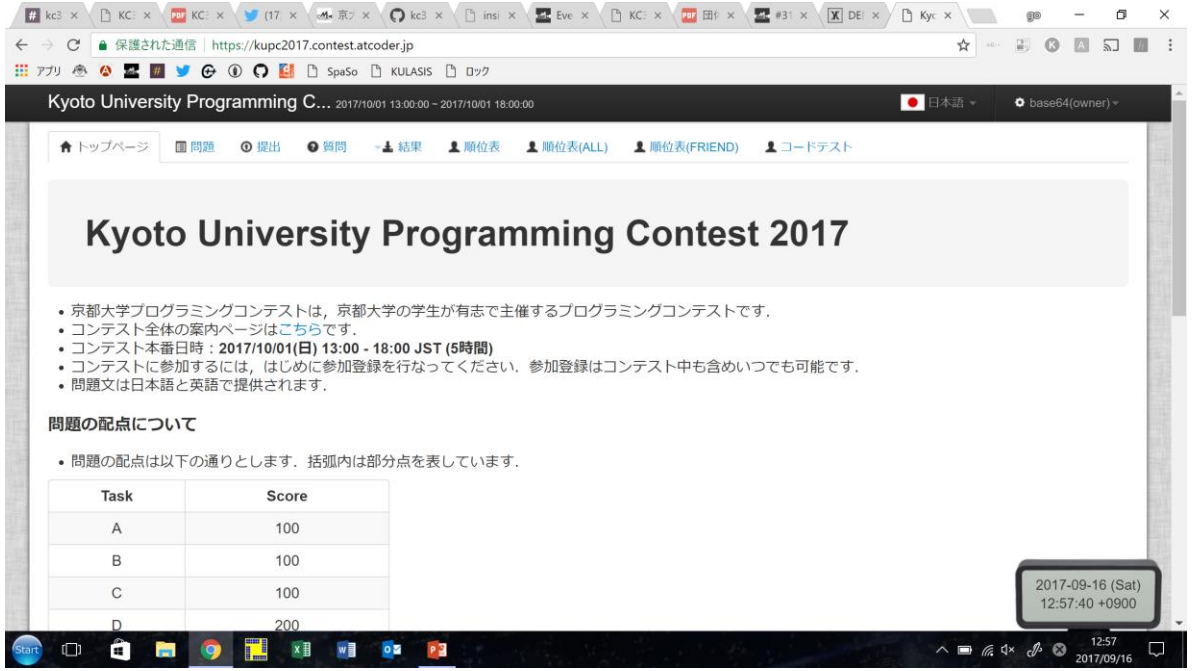
# Pietを
# 知っていますか！？

# 近況

京都大学プログラミングコンテスト
KUPCの運営をやっています

競技プログラミングに
興味のある人は是非参加を！！

日時：10/1(日)13:00~18:00



# 目次

1. 環境構築
2. Pietの紹介
3. 実際にPietを描いてみる
4. 実際にPietを描いてみる
5. 実際に……
6. ……
∞. おわりに

このスライドとほぼ同じ内容のものが
下のリンク先にあるので参照してください

https://goo.gl/ouJC2Z

# 環境構築

Pietの本格的な勉強会に移る前にまず、

KMCのdama氏製作の便利なのPietのIDE、Pidetを
インストールしてもらいます

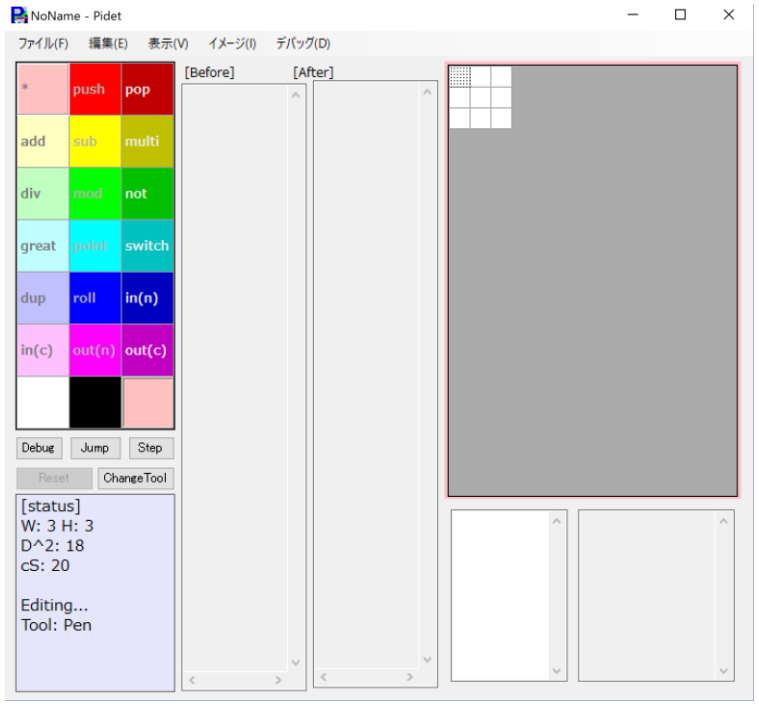
1. “pidet”で検索
2. “dnek/Pidet: IDE for Piet. – Github”
3. “releases”をクリックし、“Pidet20170614”をダウンロード

下のリンクからでも可

https://github.com/dnek/Pidet/releases/download/ver20170614/Pidet20170614.zip

# 環境構築

ダウンロードが終わったら、適当な場所に展開した後
“Pidet.exe”を実行してPidetが起動することを確認してください



# 目次

1. 環境構築
2. Pietの紹介
3. 実際にPietを描いてみる
4. 実際にPietを描いてみる
5. 実際に……
6. ……

# そもそもPietって何

- 読み方は「ピエト」であり、「ピエット」ではない
- ソースコードがドット絵である難解プログラミング言語
- David Morgan-Mar氏が考案
- Piet Mondrianの作品に影響を受け、名前もこれに由来
- 雑に描いても抽象画っぽくなる

Pietで描かれたバブルソートのプログラム→

# KMCとPiet

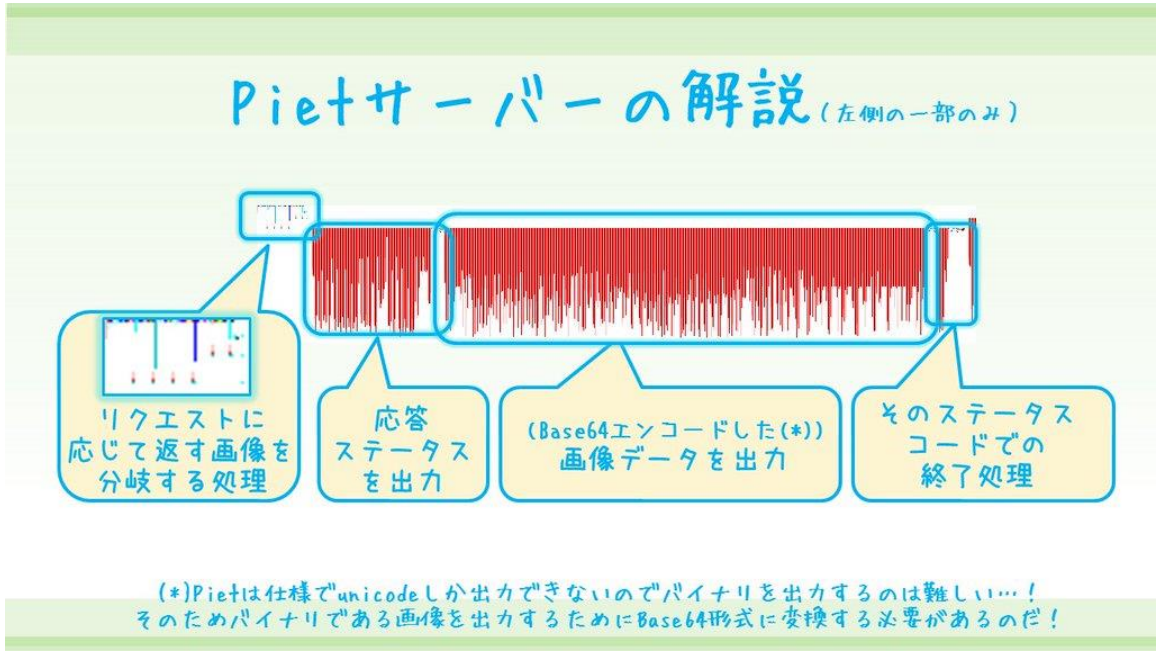
KMCでは、これまで3年間ほどPiet勉強会が続き、
日本ではほぼ最先端で、Pietに関する様々な活動をやってきた

その一部始終は「Piet」で検索するとKMC関連のページが多くヒットすることからも明らかであろう

その中で特に大きい出来事を紹介する

# 京大Pietクラブ

- 2016年のエイプリルフール企画で、
KMCのウェブページをPietのサーバーで動かした
- 京大Pietクラブへ改名




# Piet08事件

私がプログラミングコンテストにPietを使って出た話をブログに挙げていたら、UTMCの部員からPidetの仕様が間違っている
という趣旨のメッセージが飛んでくる

↓

Pidetの挙動が公式のものと一部違うことが判明

（2008年に明確化された仕様が取り込まれていなかったらしい）

KMC部員が3年間ぐらい頑張って書いてきたものは
実はPietではなかった！！

# Pietってどうやって動くの

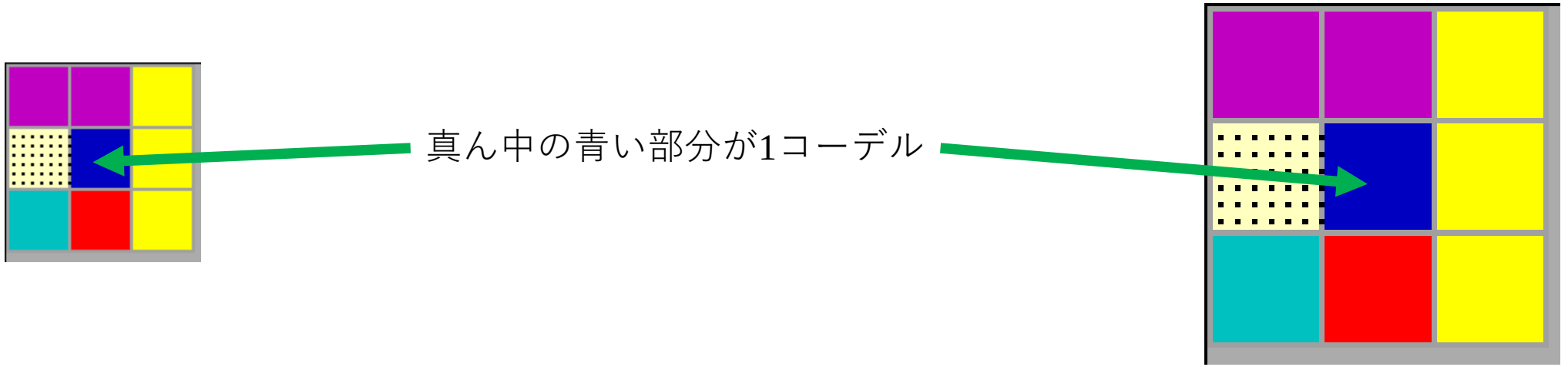
とりあえずPidetを使ってプログラムを実行してみる

# Pietってどうやって動くの

デモだけだと雰囲気しかわからないので、
詳細を説明していきます

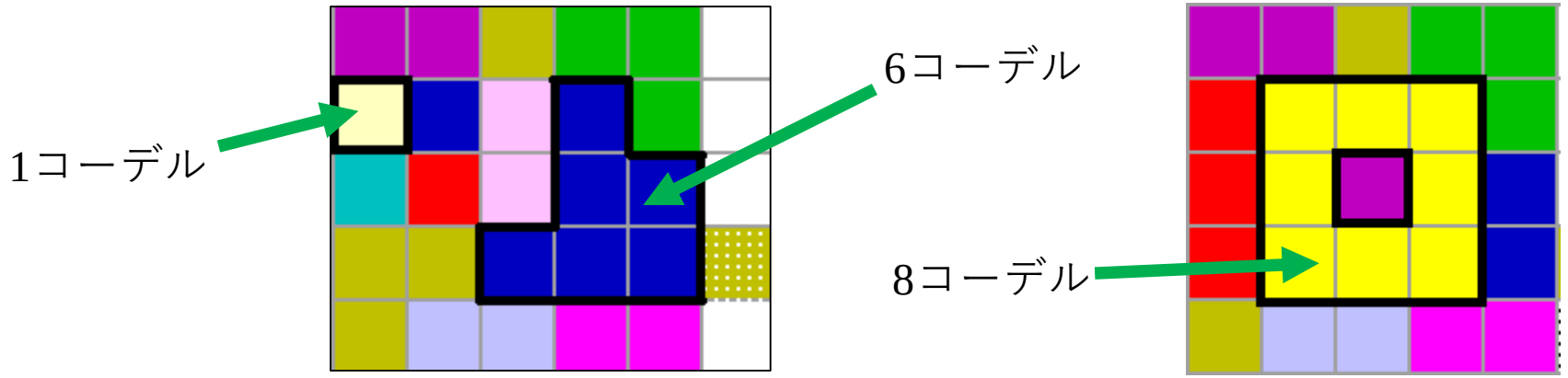
# コーデル(codel)

- Pietのコードを構成する最小単位の正方形のドットのこと
- Pidetでは画像の読み込み時と保存時に1コーデルが何ピクセルか指定する



# カラーブロック

- 同色のコーデルが縦横に並んでいるもの
- Pietのコードの基本単位



# プログラムの実行の仕方

Pietのコードの上をPP(プログラムポインター)と呼ばれる点が移動しながら命令が実行される

最初PPは左上隅のコーデルにあり、DPとCCに従って移動していく

PPの移動元と移動先の色の差によって異なる命令が実行される

# DPとは

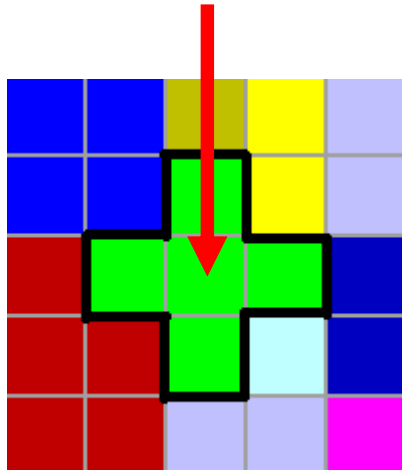
↑・↓・←・→の4状態あり、PPの移動方向を決める

PPは現在位置のカラーブロックの中で最もDP方向にある
コーデルから、DP方向にあるカラーブロックに移動する（仮）

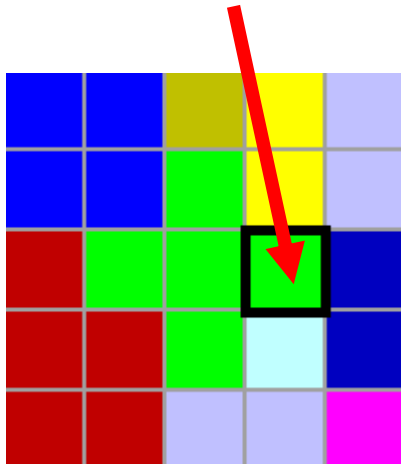
# DPとは

例えば、DPが「→」のとき、下図のように移動する

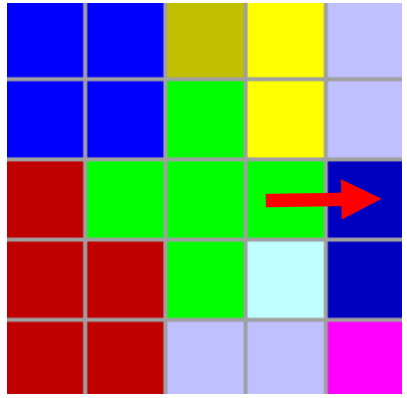
現在のPPの位置



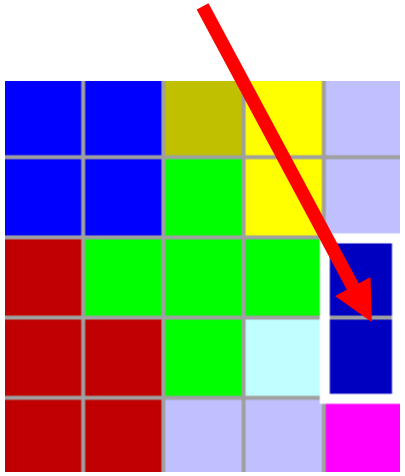
最もDP方向にあるコーデル



DP方向に移動



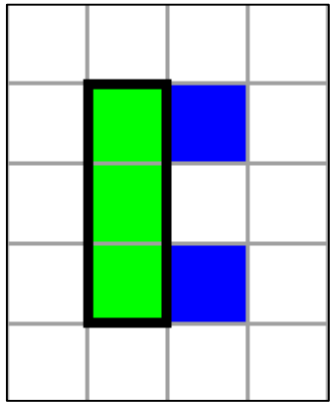
PPの移動先

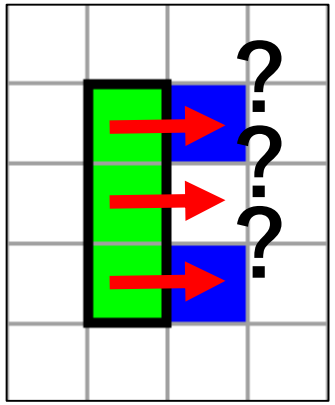


# DPだけでは困る

DPが「→」のとき、PPが下図の緑のカラーブロックから移動しようとすると、移動先が分からない

このとき、移動先をCCによって決める

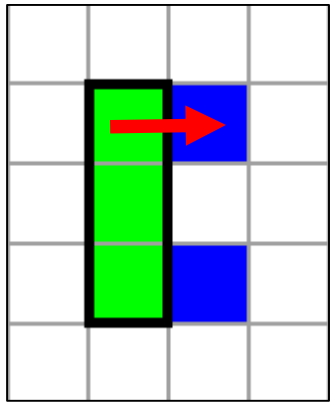




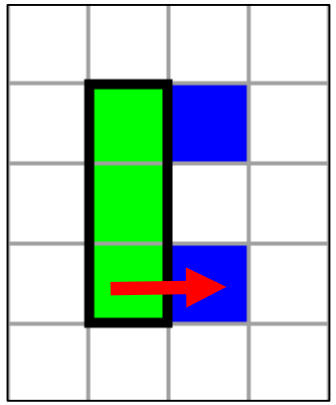
# CCとは

左(L)・右(R)の2状態ある

PPは、現在位置のカラーブロックの中で最もDP方向にある
コーデルのうち、DP方向に向かって最もCC側にある
コーデルからDP方向に移動する



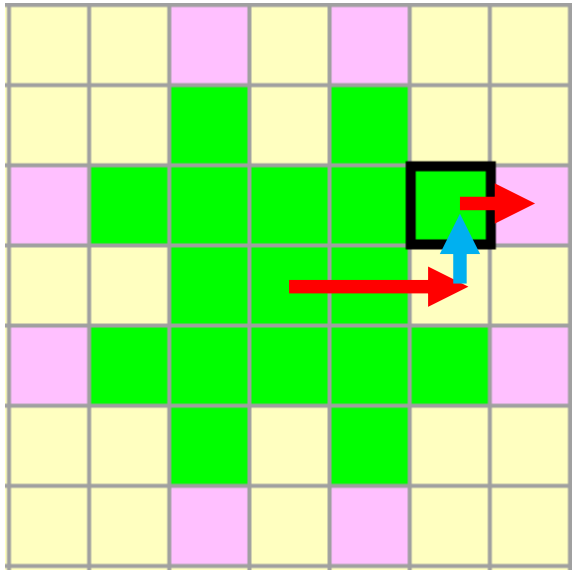
DP→, CC左の場合



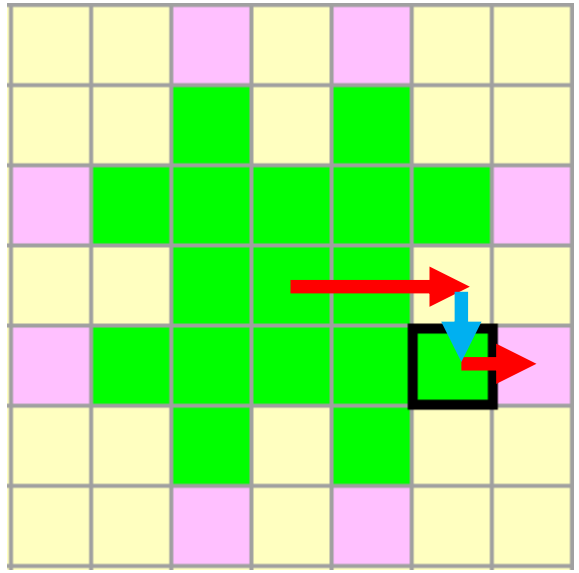
DP→, CC右の場合

# DPとCC

緑のカラーブロックからPPが移動するとき……



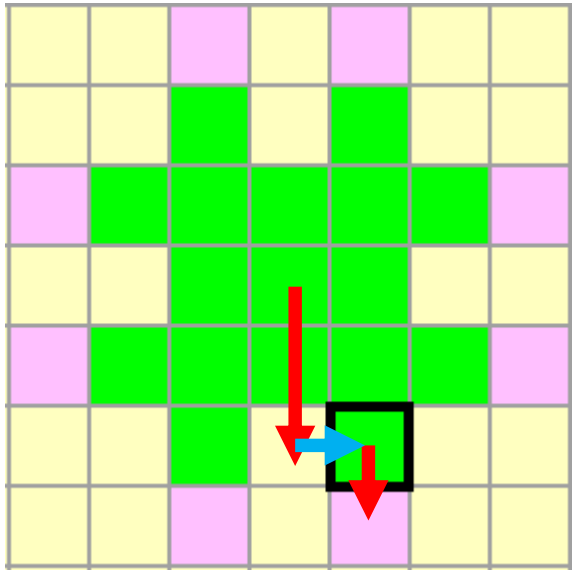
DP→, CC左



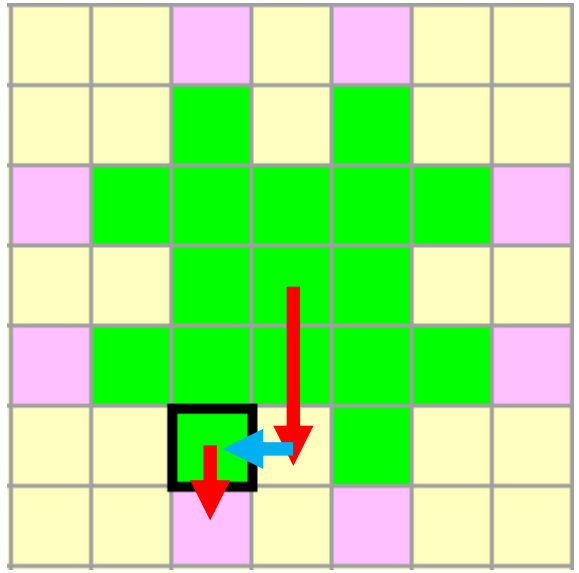
DP→, CC右

# DPとCC

緑のカラーブロックからPPが移動するとき……



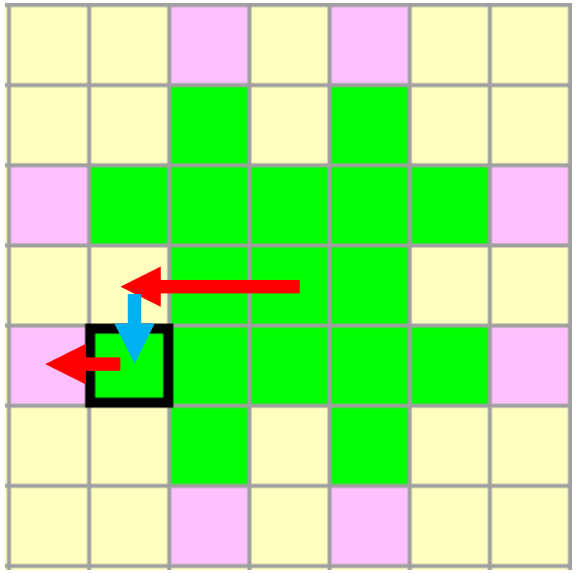
DP↓, CC左



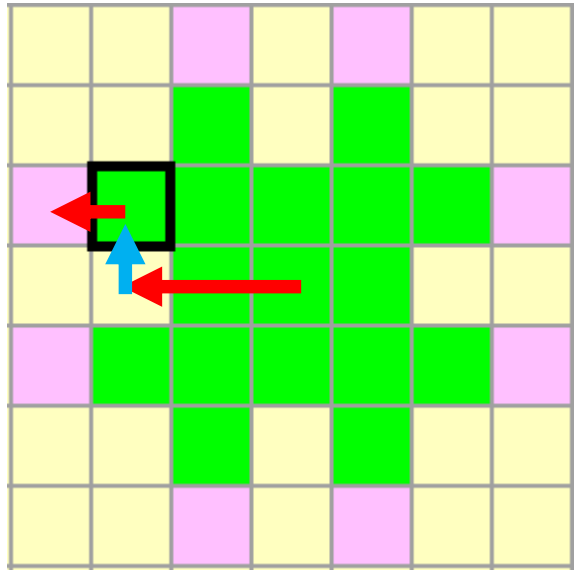
DP↓, CC右

# DPとCC

緑のカラーブロックからPPが移動するとき……



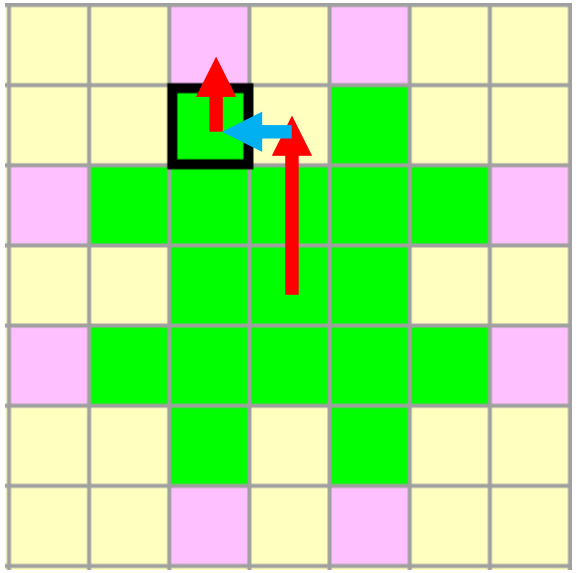
DP←, CC左



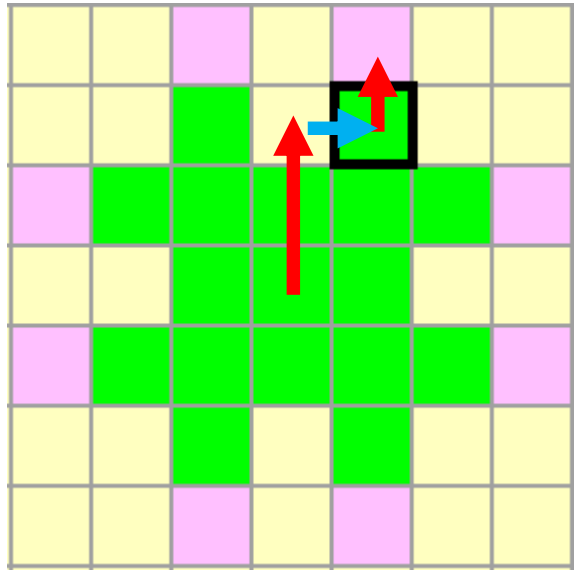
DP←, CC右

# DPとCC

緑のカラーブロックからPPが移動するとき……



DP↑, CC左



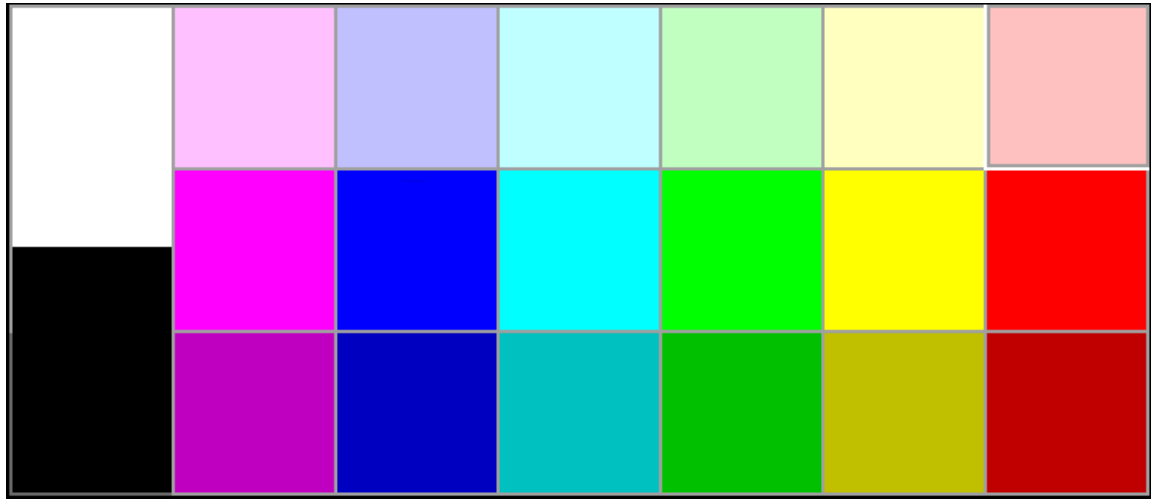
DP↑, CC右

# Pietで使える色

Pietで使用できる色は 3(明度) * 6(色相) + 2(白黒) = 20種類

それ以外の色はPidetでは白と解釈する

白と黒は他の色とは異なり、特殊な扱いがされる



# 黒の扱いと終了条件

PPの移動先が黒のコーデルやコードの外側になったときは、PPは移動に失敗する

移動に失敗した場合は、CCを切り替えてもう一度移動しようとする

それでも移動に失敗したら、DPを時計回りに90度変えて移動しようとする

それでも移動に失敗したら、CCを……

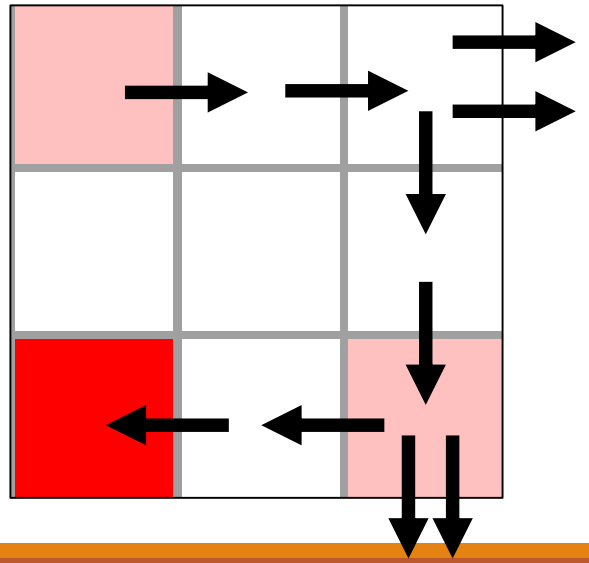
CCとDPを交互に切り替えて、連続で8回移動に失敗するとプログラムが停止する

# 白の扱い

白のカラーブロックは大きさが1コーデルのものだけであり、
白のコーデル同士が隣接していても別のカラーブロックとみなす

移動元もしくは移動先が白のコーデルのとき、移動時に
命令が実行されない

ただし、白のコーデルだけで移動が無限ループ
した場合はプログラムが停止する



# 命令

白と黒を除いた18色を用いて、PPの移動元と移動先の
色の差によって17種類の命令が実行できる

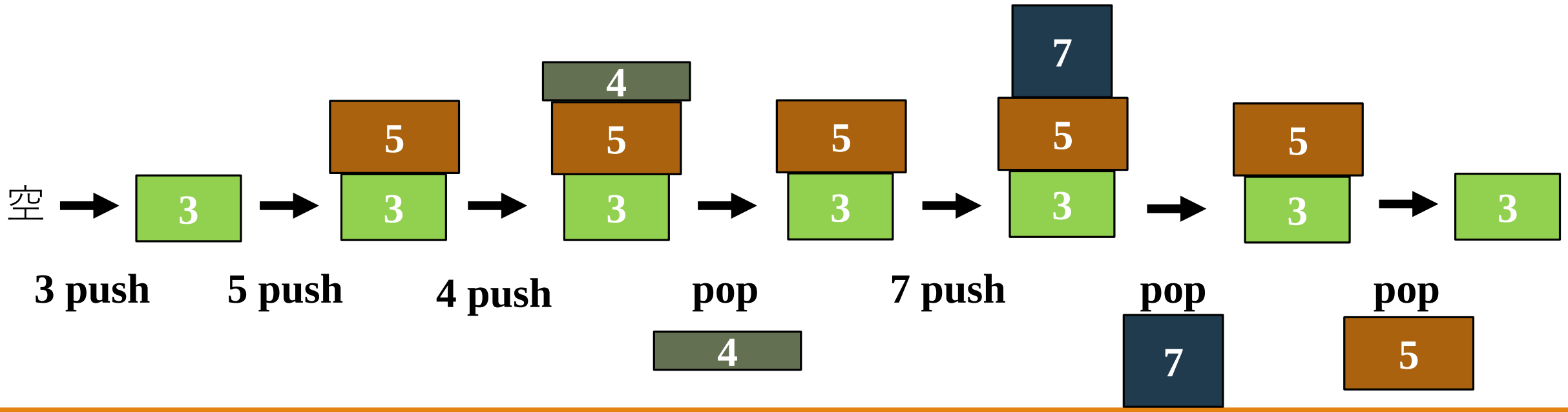
なお、記憶領域として使えるのは整数のスタック一つのみ

変数などというややこしいものは使えないので楽です(嘘)

# スタックについて（簡単な説明）

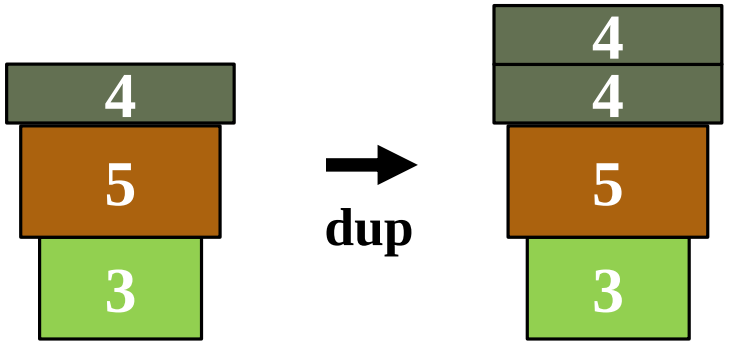
イメージとしては数の積み木

一番上に数を積むpushと一番上の積み木を取り出すpopの操作がある



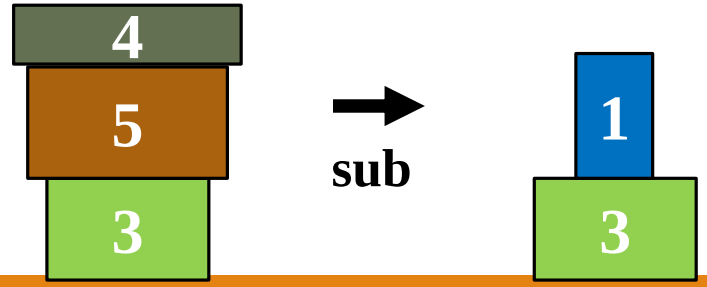
# 命令（スタック）

- push・・・移動元のカラーブロックのコーデル数をpushする
- pop・・・1回popする
- dup・・・1回popし、popした数を2回pushする



# 命令（演算）

- add・・・2回popし、popした数の和をpushする
- sub・・・2回popし、2回目にpopした数－1回目の数をpushする
- multi・・・ 2回popし、popした数の積をpushする
- div・・・ 2回popし、2回目にpopした数÷1回目をpushする（端数切捨て）
- mod・・・2回popし、2回目にpopした数 mod 1回目をpushする



# 命令（論理）

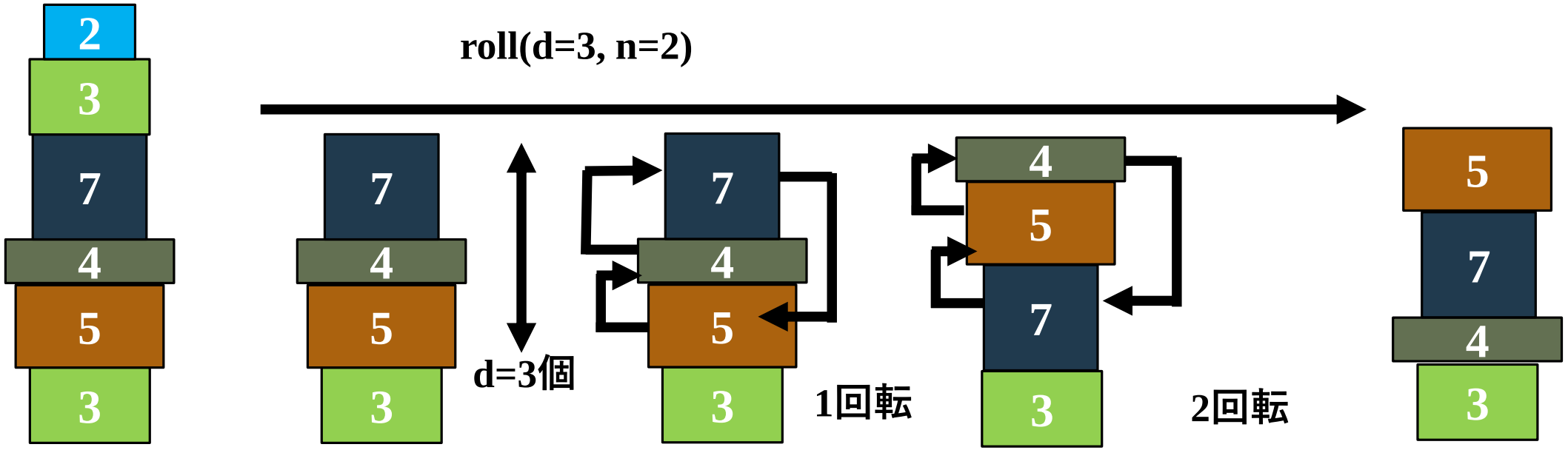
- not・・・1回popし、popした数が0ならば1を、そうでなければ0をpushする
- great・・・2回popし、2回目にpopした数>1回目ならば1を、そうでなければ0をpushする

# 命令（DPとCC）

- point・・・1回popし、DPを時計回りに
(popした数)×90度回転させる
- switch・・・ 1回popし、popした数の回数CCを切り替える

# 命令（roll）

- roll・・・スタックの中身を回転させるやっかいな命令
  まず2回popし、1回目にpopされた数をn、2回目をdとする
  その後、スタックのtopからd個の要素をn回、回す



# 命令（入出力）

- in(n)・・・標準入力から整数を受け取り、その数をpushする
- in(c)・・・標準入力から文字を受け取り、そのUnicode値をpushする
- out(n)・・・1回popし、popした数を標準出力に出力する
- out(c)・・・1回popし、popした数をUnicode値として持つ1文字を標準出力に出力する

(10をout(c)すると改行できて便利)

# 目次

1. 環境構築
2. Pietの紹介
3. 実際にPietを描いてみる
4. 実際にPietを描いてみる
5. 実際に……
6. ……

∞. おわりに

# Task1 簡単なPietプログラム

- 2つの整数を標準入力から受け取り、その和を標準出力に出力するPietのプログラムを描いてください
- 時間が余って余裕がある人はコードを小さくしたり、見た目を改善したりするとよいです
- KMCでのPietコードゴルフ界隈では、コードの対角線の長さが小さいほどよいことになっています

# Task2 簡単なループ

- 1つの正整数を標準入力から受け取り、その数が0になるまで2で割り続け、その過程を標準出力に出力するPietのプログラムを描いてください

例. 入力:“7” 出力:“7310”

入力:“16” 出力:“1684210”

- 最初にもともとの数を出力するかどうか、最後に0を出力するかどうかは自由にしてよいです
- 数と数の間に区切りを入れるとよい (入力7に対して7,3,1,0等)

# Task3 rollを理解する

- 1つの非負整数Nを標準入力から受け取り、N!を出力するPietプログラムを描いてください

例. 入力:“3” 出力:“6”

入力:“5” 出力:“120”

- roll命令を使わないと多分無理です

# Task4 ほどよい難易度

- 2つの正整数a, bを標準入力から受け取り、aとbの最大公約数を出力するPietプログラムを描いてください

例. 入力:“4 8” 出力:“4”

入力:“15 6” 出力:“3”

- 最小公倍数でもよいです
- ユークリッドの互除法を使いましょう

# Task5 Pietを完全に理解する

- 時間が余ってしょうがない人向けです
- 生半可な覚悟で取り掛からないようにしましょう
- 非負整数Nと項数Nの数列xを標準入力から受け取りxを昇順にソートした数列を出力してください
- 入力形式: N x_1 x_2 , … , x_N

例. 入力:“3 2 1 4” 出力:“1 2 4”

入力:“5 4 10 2 2 -2” 出力:“-2 2 2 4 10”

- しんどい

# 目次

1. 環境構築
2. Pietの紹介
3. 実際にPietを描いてみる
4. 実際にPietを描いてみる
5. 実際に……
6. ……

∞. おわりに

# おわりに

- きっとPietの分岐とループの描き方を学んだはず。
  これで任意のPietコードが描けるはず
- 絵や文字っぽいプログラムを描くときは、先に絵を
  用意してからPietに　修正していくと描きやすいらしい
- QRコードとしても意味を持つPietコードは先にQRコードを
  用意して、白い部分に明るい色を、黒い部分に暗い色を使うとできる
- 俺たちのPietライフはまだまだ始まったばかりだ
- (ちなみに私はもうPietは飽きた)