

# リファクタリング演習

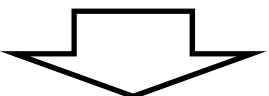
ゲームプログラミングA #06

向井 智彦

# 今日の流れ

- リファクタリング演習
  - Vector2の利用
  - カプセル化したクラスの導入
- 応用リファクタリング
  - 4つの課題を順不同でも構わないので可能な限りたくさん達成して来週授業までにプルリクエスト
    - 未完成状態でも構わないのでコミット&プルリク

# リファクタリング課題1

- ゲームクリア判定を変更
    - 現状：update関数内で各ブロックのvisibilityを毎回調べる冗長な処理
- 
- 残っているブロックの数or消したブロックの数をグローバル変数でカウント&一定数を超えたらゲームクリア
    - 変数名は `numDisappearedBlocks/numActiveBlocks` 等
    - ボールの総数 `kNumBlocks` との比較でクリア判定

## 課題2(改造)

- ブロックを消すたびボールを加速
  - BallクラスにIncreaseSpeedメンバ関数を追加
  - ブロックを消すたびIncreaseSpeedを呼出し

注意)速くなりすぎない程度に調整

# リファクタリング課題3

- BlockとBallの衝突判定をBlockクラス内で行う
  - Blockクラスに`checkBall(Ball& ball)`メンバを追加
  - 関数`checkBlock` の処理内容を移植
    - 衝突判定およびボールの進行方向変更などを処理
  - `update`関数内で`checkBall`メンバ関数を呼び出し
- PlayerBoxとBallの衝突判定をPlayerBox内で行う(同上)

# リファクタリング課題4

- 壁クラス **Wall** を作成し、壁のサイズや位置を **initializeGame** 内で指定できるように修正
  - 例) ブロックと上側の壁を隙間なくくっつけるなど
  - 他のクラスと同じく、init、draw、update メンバ関数を追加しておく(中身は空でOK)
- Ball との衝突判定を **Wall** クラス内で行う
  - 課題3と同じ
    - Wall クラスに **checkBall(Ball& ball)** メンバを追加

# 課題3&4の目的

- Ball 以外の全てのゲームオブジェクトの処理をカプセル化 & 共通化
  - Block, PlayerBox, Wallのいずれも...
    - ボールとの衝突処理を行う
    - init, update, draw, checkBall という同名のインタフェース(メンバー関数)を提供
    - Ballクラスも init, update, draw を提供

→ 全ての**ゲームオブジェクト**に共通性を与える

# 来週以降の予告

- プログラム全体の動作の解説
  - ゲームの改造
    - 物体の形を変えたり模様を貼り付ける
    - プレイヤーの動きを変える、弾を飛ばす
    - ブロックを動かす、消えるときのエフェクトを変更
- など、数週間にわたって改造していきます