2025年06月24日(火)
3,4,5限 ＠研A302

プロジェクト演習 テーマD

第3回

担当：CS学部 講師 伏見卓恭
連絡先：fushimity@edu.teu.ac.jp

授業の流れ

- 第1回：実験環境の構築／Gitの基礎／Pygameの基礎
- 第2回：Pygameによるゲーム開発の基礎／コード規約とコードレビュー
- 第3回：オブジェクト指向によるゲーム開発／GitHubの応用
- 第4回：Pygameによるゲーム開発の応用／共同開発の基礎
- 第5回：共同開発演習（個別実装）
- 第6回：共同開発演習（共同実装）
- 第7回：共同開発演習（成果発表）

本日のお品書き

1. オブジェクト指向なゲーム開発
2. ブランチとマージ
3. Pygameの演習
4. コードレビュー，ブランチ間差分表示

目標：オブジェクト指向なコードを実装でき，
ブランチを切って作業，ブランチをマージできる

3限：オブジェクト指向／ブランチとマージ

配布物の確認

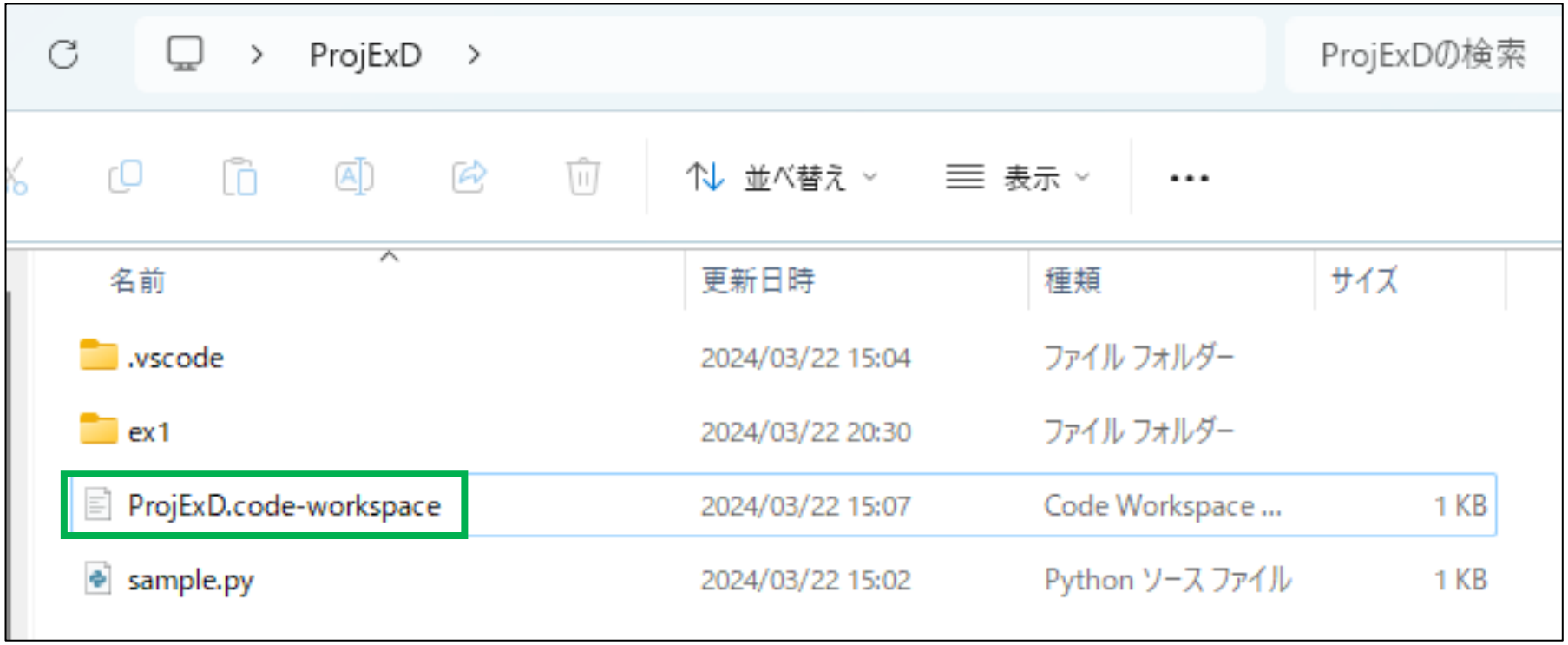
- ex3.zipをProjExDフォルダにDLし，展開する

【配布物配置後のディレクトリ構造】

- ProjExD/
  - sample.py
  - ex2/
  - ex3/　　本日の配布物
    - fight_kokaton.py　・・・たたかえ！こうかとん
    - fig/
      - pg_bg.jpg　・・・背景画像
      - {0, …, 9}.png　・・・こうかとん画像
      - beam.png　・・・ビーム画像　← 指示があるまで追加しないで
      - explosion.png　・・・爆発画像　← 指示があるまで追加しないで

VScodeの起動

- ProjExDフォルダ内の「ProjExD.code-workspace」をダブルクリックし，ワークスペースを開く



gitでの作業

- 「ProjExD¥ex3」フォルダを右クリックし，Git Bashを起動する
- gitリポジトリを初期化する：`git init`
- 全ファイルをステージングする：`git add .`　beamとexplosionはフォルダに置かないで
- 「初期状態」というコメントでコミットする：`git commit -m "初期状態"`

- GitHubに「ProjExD_3」という公開リポジトリを作成する
- 公開リポジトリの情報を「origin」という名前で登録する：
  `git remote add origin https://github.com/自分のアカウント/ProjExD_3.git`
- 公開リポジトリにコミット履歴をプッシュする：`git push origin main`

Git for Windowsでない人は
SSH版をコピペ

コードの解説（全体像）

- 前回からの変更点：
  クラス（Bird, Bomb）を定義

【オブジェクト指向な書き方】
- **クラス**を定義し，ゲーム内のオブジェクトを
  クラスから生成した**インスタンス**として扱う
- 多数のオブジェクトが登場するゲームでは
  オブジェクト指向の書き方は必須

Birdクラス →

Bombクラス →

```python
import os
import random
import sys
import time
import pygame as pg


WIDTH = 1100  # ゲームウィンドウの幅
HEIGHT = 650  # ゲームウィンドウの高さ
os.chdir(os.path.dirname(os.path.abspath(__file__)))


def check_bound(obj_rct: pg.Rect) -> tuple[bool, bool]:…


class Bird:…


class Beam:…


class Bomb:…


def main():…


if __name__ == "__main__":
    pg.init()
    main()
    pg.quit()
    sys.exit()
```

【復習】クラス定義とインスタンス生成

対象となるモノやコト（オブジェクト）の設計図（クラス）を定義し，
定義に基づき実際のオブジェクト（インスタンス）を生成する

クラスの定義

```
class クラス名:
    def __init__(self, パラメータ1, パラメータ2):
        self.属性1 = パラメータ1
        self.属性2 = パラメータ2
        self.属性3 = リテラル
```

インスタンスの生成

```
インスタンス1 = クラス名(引数1, 引数2)
インスタンス2 = クラス名(引数1, 引数2)
              :
```

→

```
class Bird:
    def __init__(self, xy):
        self.img = 画像Surface作成
        self.rct = Rect取得
        self.rct.center = xy


bird = Bird((300,200))
```

イニシャライザ `__init__()` はインスタンス生成と同時に
属性に値を設定するのに用いられる特殊メソッド．
selfを除く位置パラメータの数と，位置引数の数は一致する必要がある

【復習】インスタンス変数・メソッド

生成されるインスタンスそれぞれに紐づけられる変数（属性）・メソッド

インスタンス変数（属性）へのアクセス

インスタンス名．インスタンス変数名

インスタンスメソッドの定義

```
class クラス名:
    def インスタンスメソッド名(self, パラメータ):
        インスタンス変数に対する処理
        クラス変数に対する処理　など
```

インスタンスメソッドの使用

インスタンス名．インスタンスメソッド名(引数)

```
class Bird:
    def __init__(self, xy):
        self.img = 画像Surface作成
        self.rct = Rect取得
        self.rct.center = xy

    def update(self, screen):
        self.rct.move_ip(vx, vy)
        screen.blit(self.img, self

while True:

    bird.update(screen)
```

コードの解説（Birdクラス）

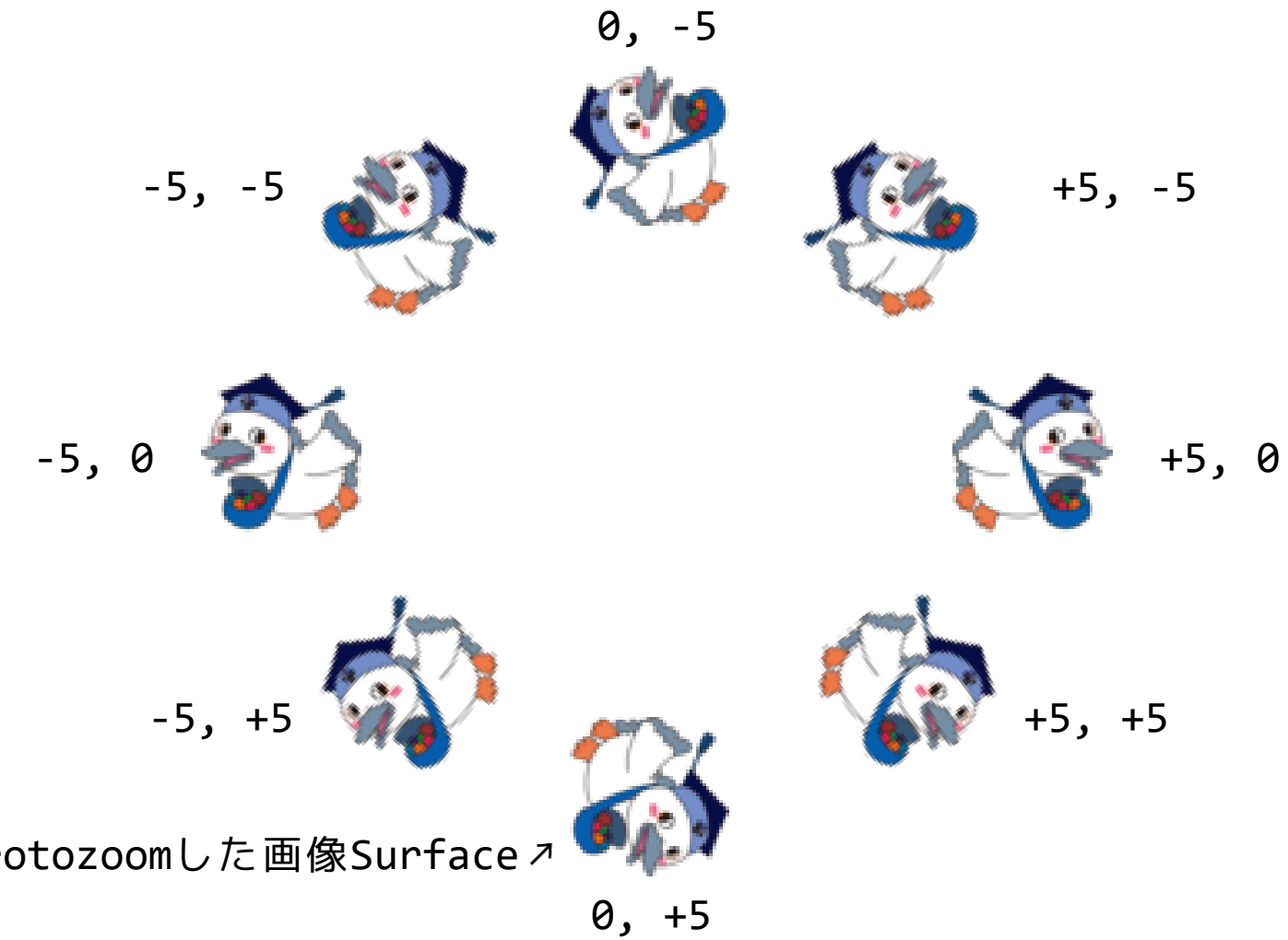
__class__：
当該クラス（この場合Bird）を意味する

- 押下キーと移動量の対応辞書（クラス変数）：delta
- 押下キーvs回転後画像Surfaceの辞書（クラス変数）：imgs
- イニシャライザ：__init__
  ←前回main関数内**while文前**に記述した内容
  - デフォルト画像のSurfaceを生成：self.img
  - SurfaceのRectを抽出し，初期座標を設定：self.rct
- 画像切り替えメソッド：change_img
  - 切り替え後の画像Surfaceを生成し，self.imgを置き換える
  - 置き換え後のSurfaceを画面にblit
- 位置更新メソッド：update
  ←前回main関数内**while文内**に記述した内容
  - 合計移動量を求めmove_ip
  - check_boundで画面外でないか確認
  - 位置更新後のSurfaceを画面にblit

押下キーvs回転後画像Surfaceの辞書

- 押下されたキーに応じて，
  rotozoomで回転させた
  画像Surfaceをblitする

- 押下キーに対する移動量
  の合計値リストをキー，
  rotozoomしたSurfaceを
  値とした辞書を用意しておく



コード解説（Bombクラス）

- イニシャライザ：__init__ ←前回，while文の前に記述した内容
  - 指定色，指定サイズの爆弾円のSurfaceを生成：self.img
  - SurfaceのRectを抽出し，初期座標を設定：self.rct
  - 速度を設定：self.vx，self.vy

- 位置更新メソッド：update ←前回，while文の中に記述した内容
  - check_boundで画面外でないか確認
  - 速度に応じて移動後の位置を求めmove_ip
  - 位置更新後のSurfaceを画面にblit

Birdクラス，Bombクラス，これから作るBeamクラス
いずれもイニシャライザでSurfaceとRectの設定をして，updateメソッドで動きを定義するので
統一感のある書き方による可読性の向上が見込まれる

コードの解説（main関数）

```
bird = Bird((300, 200))
bomb = Bomb((255, 0, 0), 10)

while True:
    if bird.rct.colliderect(bomb.rct):
        bird.change_img(8, screen)
        pg.display.update()
        time.sleep(1)
        return

    key_lst = pg.key.get_pressed()
    bird.update(key_lst, screen)
    bomb.update(screen)
    pg.display.update()
    clock.tick(50)
```

← こうかとんと爆弾の生成を1行で書けるようになった
　各クラスのイニシャライザを呼び出している

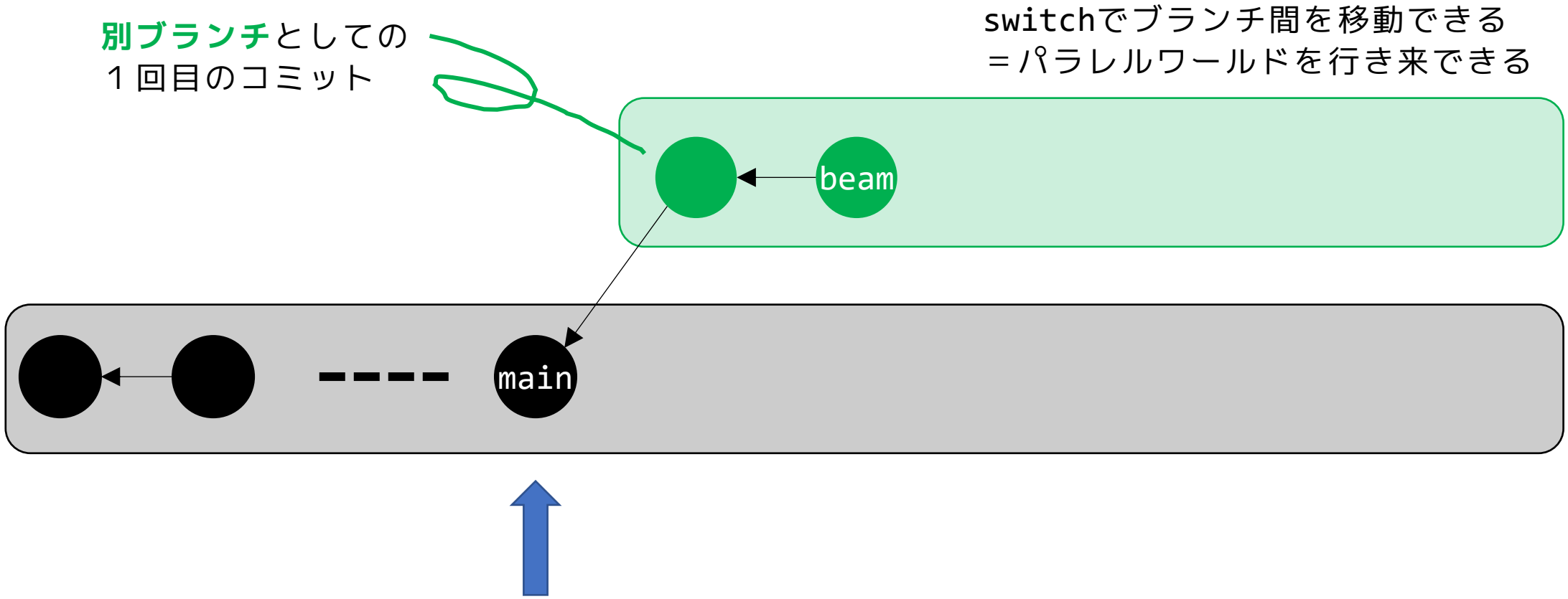
← こうかとんと爆弾の衝突判定
← Birdクラスのchange_imgメソッドを呼び出し，
　ゲームオーバー時にこうかとん画像を切り替える

← こうかとんと爆弾の位置更新を1行で書けるようになった

「たたかえ！こうかとん」を実装しよう

- 野原で遊ぶこうかとんに爆弾が襲い掛かる．
  ビームで爆弾を打ち落とすゲームを実装する．

コミットとブランチ(イメージであり実際は異なる)



現時点の完成版（基本機能）は壊さずに残し，
別ブランチにて新たな機能を開発する

ブランチを切る

- 現在のブランチ(main)を基に，新しいブランチを作り，
  新ブランチで追加機能を実装する．
- これにより，基本機能が実装されたmainブランチを壊さずに済む．

- ブランチを作る：`git branch ブランチ名`
- ブランチリストを確認する：`git branch`
- ブランチを切り替える：`git switch ブランチ名`

ブランチ作成と切替
を同時にできる

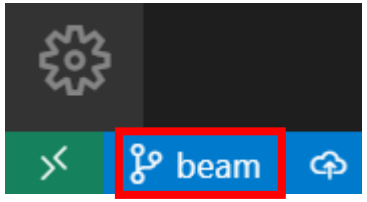
`git switch -c ブランチ名`

```
Admin@MSI MINGW64 ~/Desktop/ProjExD/ex3 (main)
$ git branch beam

Admin@MSI MINGW64 ~/Desktop/ProjExD/ex3 (main)
$ git branch
  beam
* main
Admin@MSI MINGW64 ~/Desktop/ProjExD/ex3 (main)
$ git switch beam
Switched to branch 'beam'

Admin@MSI MINGW64 ~/Desktop/ProjExD/ex3 (beam)
$
```

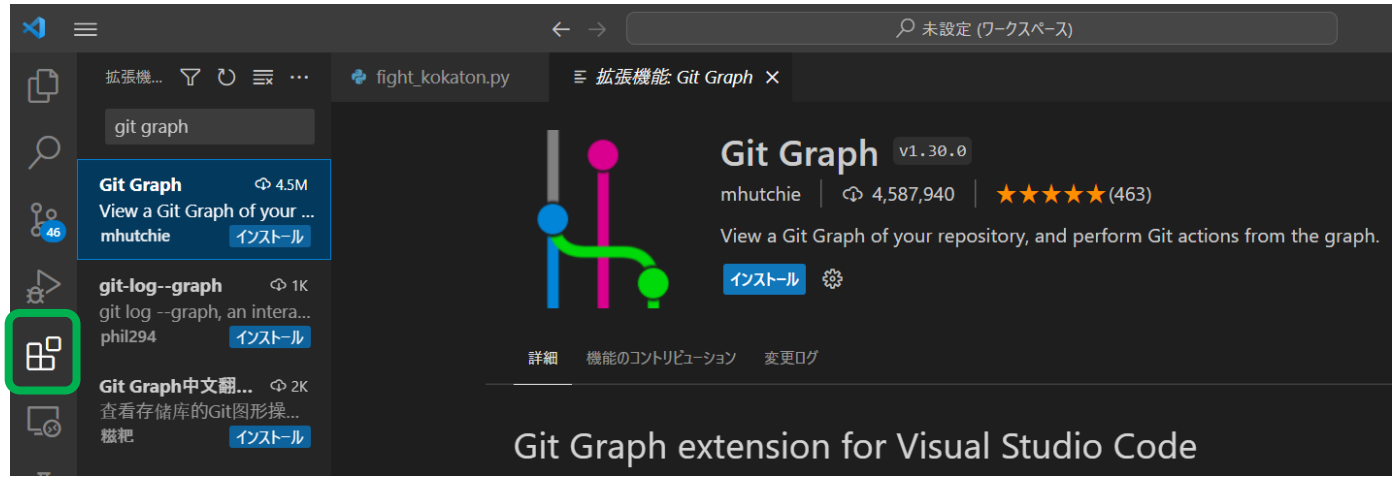
←現在のブランチ

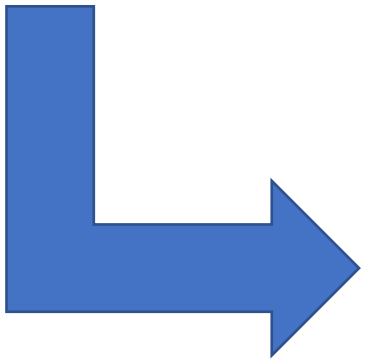


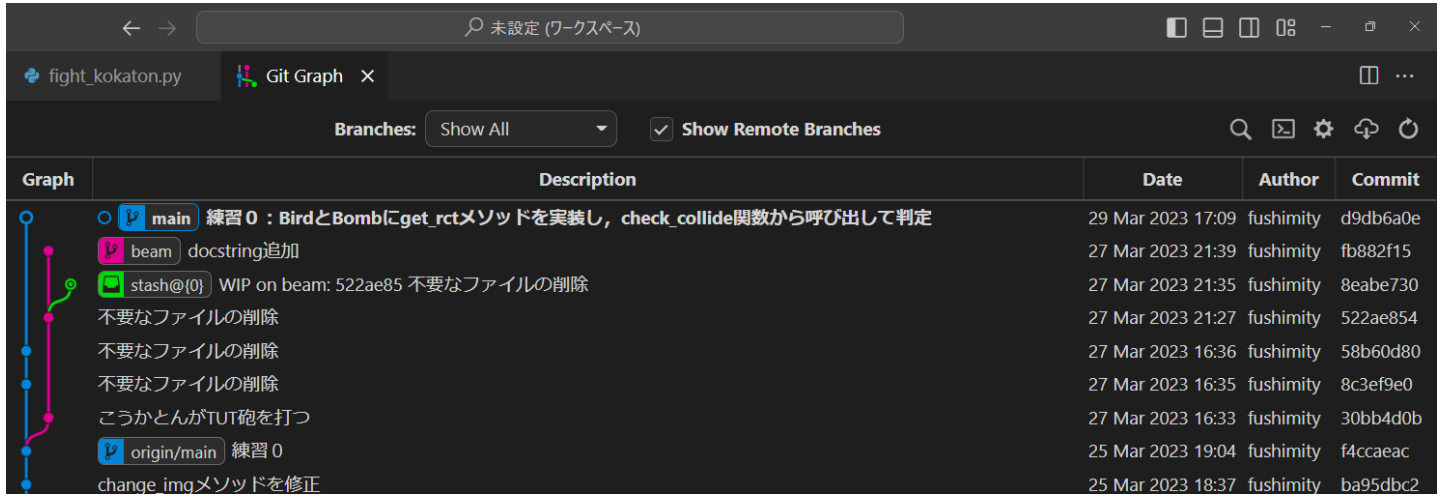
← VScode左下のブランチ名も
自動で切り替わっているのが望ましい

GitLens／Git Graph

- ブランチ間の関係を視覚的に見ることができる拡張機能







練習問題

1. beamブランチにて，Beamクラスを実装せよ．
   - 準備：beam.pngをex3/fig/フォルダに配置　← git statusを見てみよう
   - イニシャライザ：
     - beam.pngをロードして画像Surfaceを生成
     - こうかとんの右に配置(ビームの左座標がこうかとんの右座標になるように)
     - 速度は横方向に５／縦方向に０

     ←Birdクラスを参考に
   - updateメソッド：画面内なら座標を更新し，screenにblit

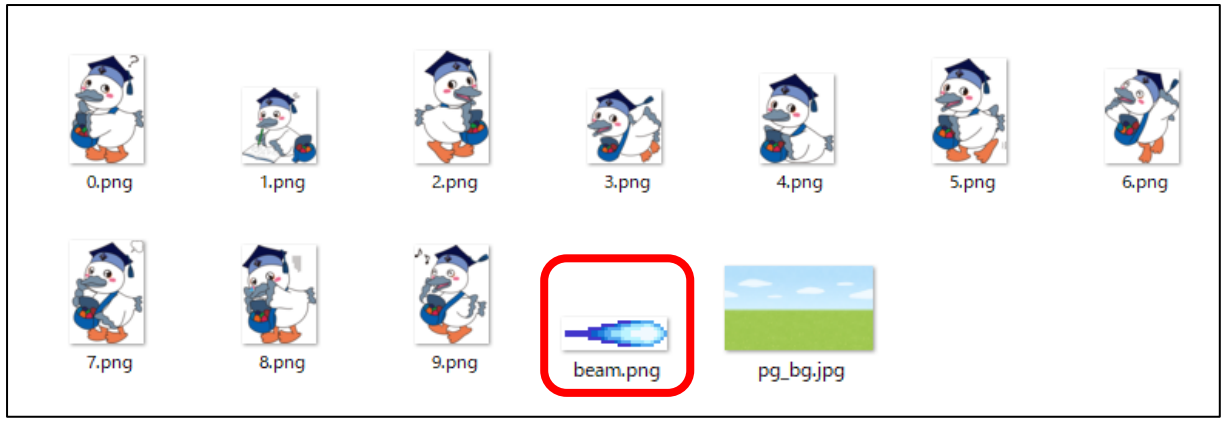
     ←Bombクラスを参考に

   - main関数(初期化)：beam変数をNoneで初期化
   - main関数(ループ)：
     - スペースキー押下でBeamインスタンスを生成　←第1回講義資料P.60を参考にする
     - updateメソッドを呼び出して座標更新＋blitする

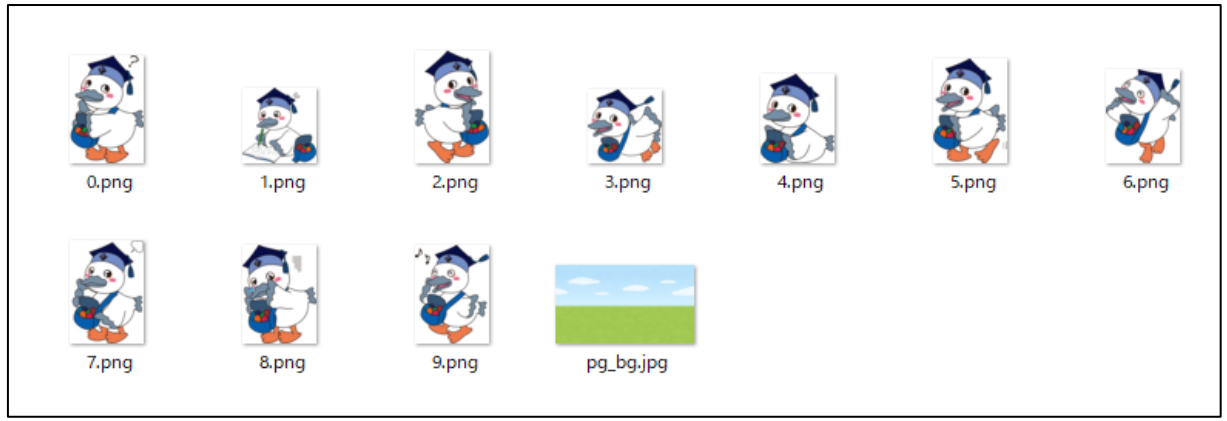
練習問題

- 実装→fight_kokaton.pyとfig/beam.pngをステージング→コミットしたら，一度mainブランチに戻ってみよう
  - 練習１を実装する前の状態に戻っていることを確認（beam.pngも見えない）

beamブランチ



mainブランチ



練習問題

2．再びbeamブランチにて，爆弾とビームが衝突した際に
Beamインスタンス，Bombインスタンスを消滅させよ→Noneにする

↑ こうかとんと爆弾の衝突判定を参考にする

- Noneにすると，各種判定ができずエラーが発生する（Noneはrctを持っていない）
以下のようにNoneチェックが必要
  - 爆弾とビームの衝突判定の前に爆弾とビームのNone判定が必要

```
if bomb is not None:
    if beam is not None:
        if beam.rct.colliderect(bomb.rct):
```

  - こうかとんと爆弾の衝突判定の前に爆弾のNone判定が必要

```
if bomb is not None:
    if bird.rct.colliderect(bomb.rct):
```

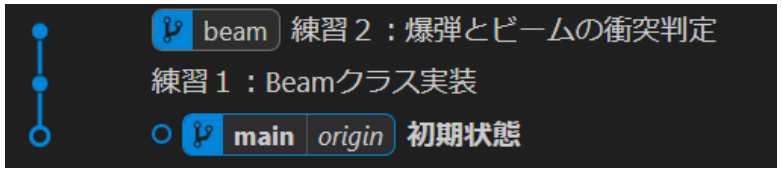
  - 爆弾のupdateの前に爆弾のNone判定が必要
  - ビームのupdateの前にビームのNone判定が必要

【重要】コミットと別ブランチへの移動

- コミットしないで，mainブランチに戻ってみる
  →加えた変更がmainブランチと競合しない場合，
  　変更がmainブランチに持ち込まれてしまう

- 基本的には，
  - ステージング・コミットしてから，別ブランチへスイッチする
  - 変更をスタッシュ(次ページ参照)してから，別ブランチへスイッチする

- スタッシュしなかった人は，
  beamブランチにてステージング→コミットしましょう



【参考】スタッシュ：作業ツリーの変更を一時退避する

- 【重要】
  基本的には，ステージング，コミットしてからブランチを切り替える

- 一時退避する状況の例
  - 作業中に他のブランチに移動したい
  - まだ中途半端だからコミットはしたくない
  - でも，今の作業内容を消したくない
    → コミット前に一時退避して，後から作業を続行する
- 手順
  - 変更を退避する：`git stash`
  - 確認する：`git stash list`／差分を確認する：`git diff stash@{0}`
  - 他のブランチに移動し作業する
  - 他のブランチでの作業を終え，戻ってくる
  - 退避した変更を戻す：`git stash pop`／`git stash pop stash@{0}`

popした変更はstashリストから削除される（applyの場合は削除されずに残る）

【参考】スタッシュを試してみる

- beamブランチで，コードをちょっと変更する

たとえばBirdクラス
を丸ごと消してみる

- ステージング，コミットせずに，mainブランチにスイッチする
  → エラーが出て，スイッチできない（変更内容によってはエラーは出ない）

```
Admin@MSI MINGW64 ~/Desktop/ProjExD/ex3 (beam)
$ git switch main
error: Your local changes to the following files would be overwritten by checkout:
        fight_kokaton.py
Please commit your changes or stash them before you switch branches.
Aborting
```

- スタッシュしてから，
  mainブランチにスイッチする → エラーなくスイッチできた

消したBirdクラスが戻っている＝最後にコミットした状態になる

- beamブランチに戻る：この時点では，スタッシュした変更は見えない
- 退避した変更を戻す
- 必要に応じて，変更を継続したり，ステージング，コミットする

ブランチをマージする

- あるブランチを別のブランチに統合することをマージすると言う.
- 切ったブランチにて追加機能の実装が完了し，mainブランチに統合する.

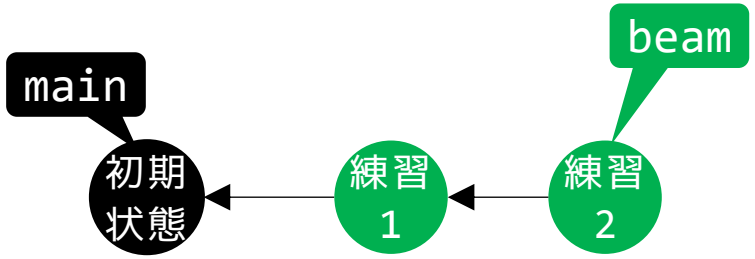
【beamブランチをmainブランチにマージする場合】

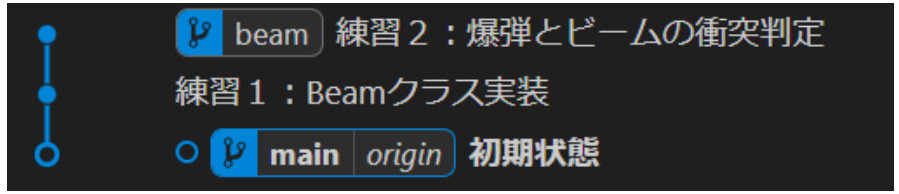
- mainブランチに戻る：`git switch main`
- beamブランチをマージする：`git merge beam`

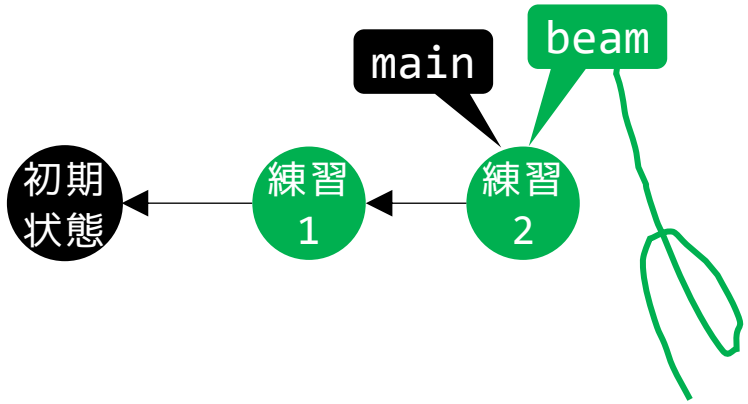
```
Admin@MSI MINGW64 ~/Desktop/ProjExD/ex3 (main)
$ git merge beam
Updating 7a06ff2..cc1b362
Fast-forward
 fig/beam.png      | Bin 0 -> 1031 bytes
 fight_kokaton.py |  54 ++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++-------
 2 files changed, 47 insertions(+), 7 deletions(-)
 create mode 100644 fig/beam.png
```

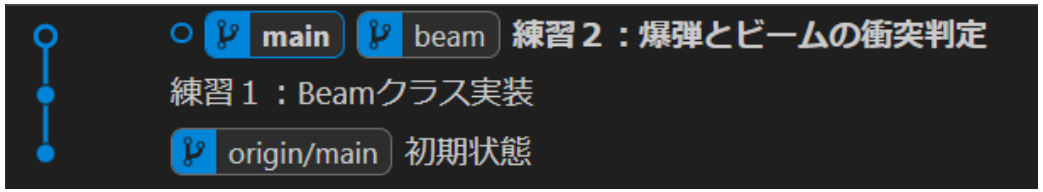
ブランチのマージ(Fast-forwardマージの場合)

ブランチとは，ある特定のコミットのこと







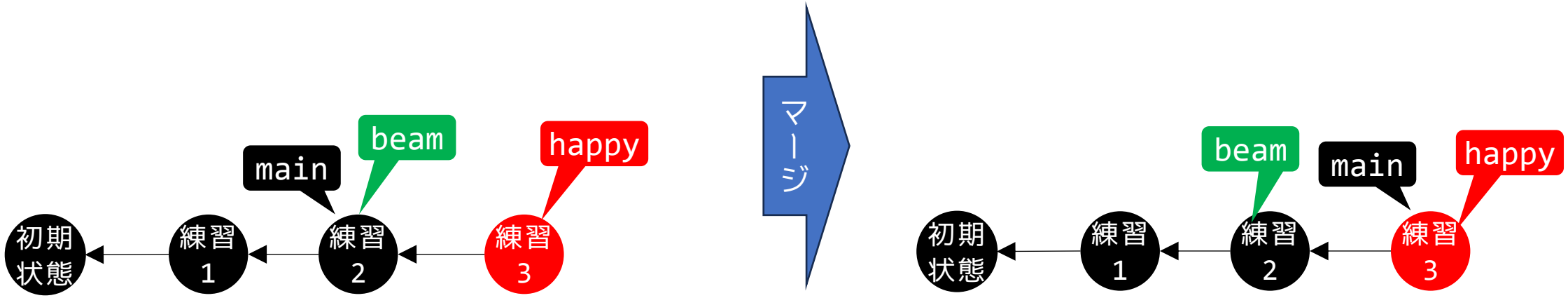


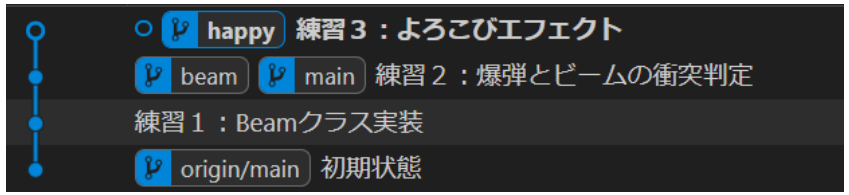
マージしても削除しなければ残ったまま

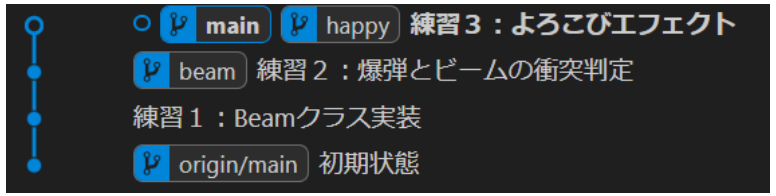
練習問題

3．happyブランチにて，ビームで爆弾を撃ち落とした際に
こうかとんが喜ぶエフェクトを実装せよ ←Birdクラスのchange_imgメソッドを利用する

- 実装→ステージング→コミット→mainブランチにスイッチ
→happyブランチをmainブランチにマージしてみよう





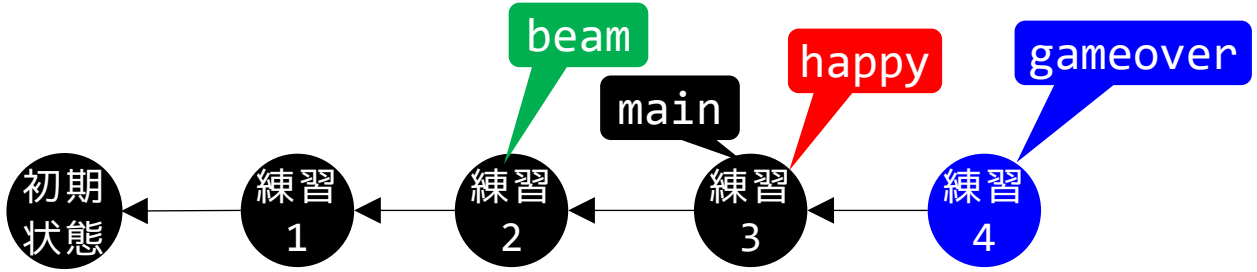


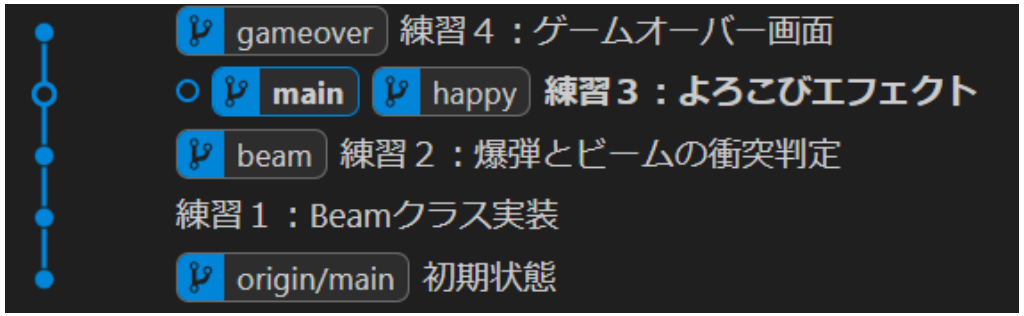
練習問題

4. gameoverブランチにて，
   簡易ゲームオーバー時の画面を実装せよ

```
fonto = pg.font.Font(None, 80)
txt = fonto.render("Game Over", True, (255, 0, 0))
screen.blit(txt, [WIDTH//2-150, HEIGHT//2])
```

- 実装→ステージング→コミット→mainブランチにスイッチ
  →今回はマージしない





練習問題

gameoverブランチをマージしていない
mainブランチからbombブランチを作っている

5. bombブランチにて，複数の爆弾を実装せよ．

- トップ：爆弾の数を表す定数NUM_OF_BOMBSを定義（「5」を代入しておく）

Bombクラス
- main関数(初期化)：NUM_OF_BOMBS個のBombインスタンスを要素とするリストを生成(内包表記で書いてみよう)
- main関数(ループ)：
  - リストの要素1つずつに対して，こうかとんと衝突判定／ビームと衝突判定し，衝突した要素はNoneとする
  - 爆弾リストに対して，要素がNoneでないものだけのリストに更新する

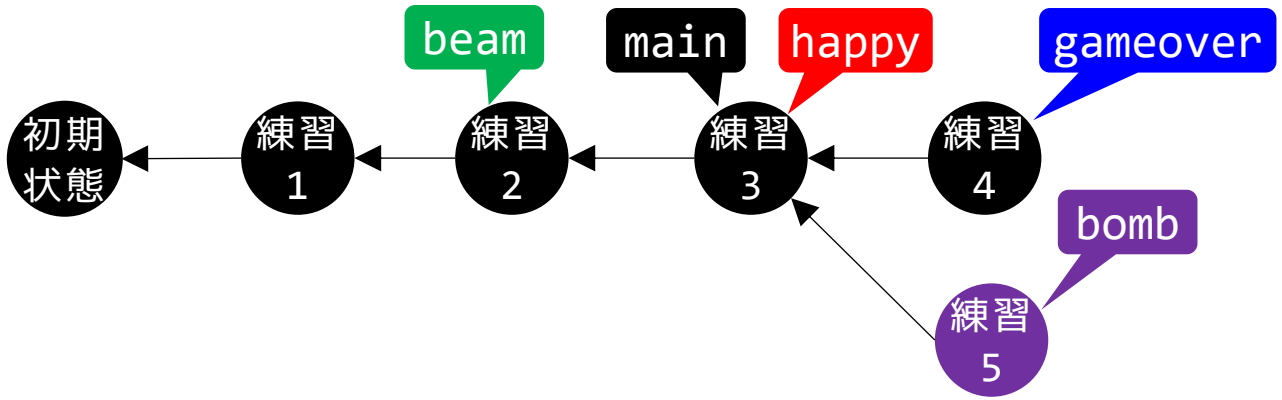
```
bombs = [bomb for bomb in bombs if bomb is not None]
```

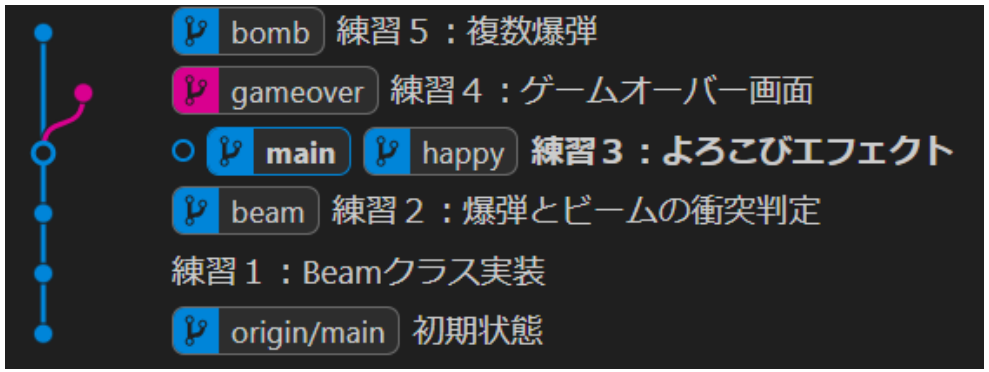
  - 更新した爆弾リストの要素1つずつに対して，updateメソッドを呼び出す（Noneチェック不要）

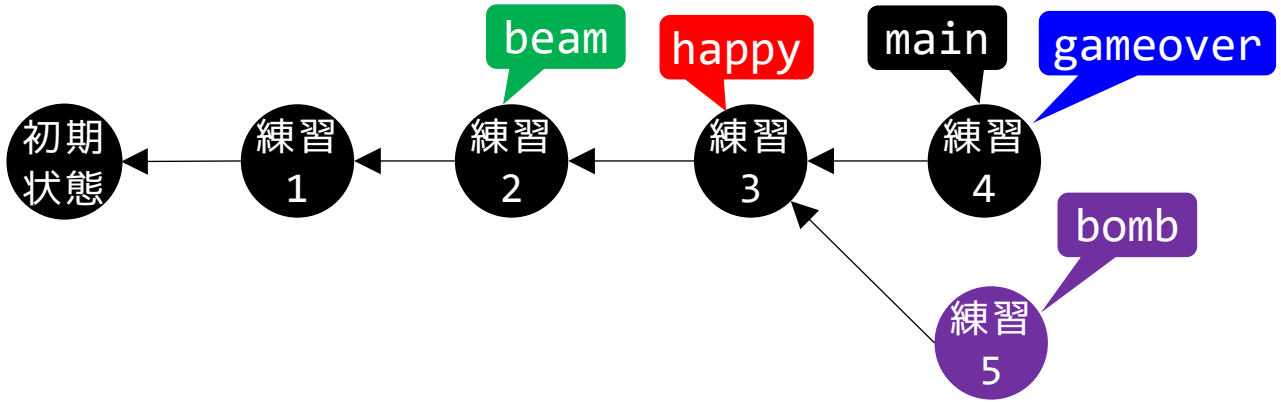
ブランチをマージしてみる（1/2）

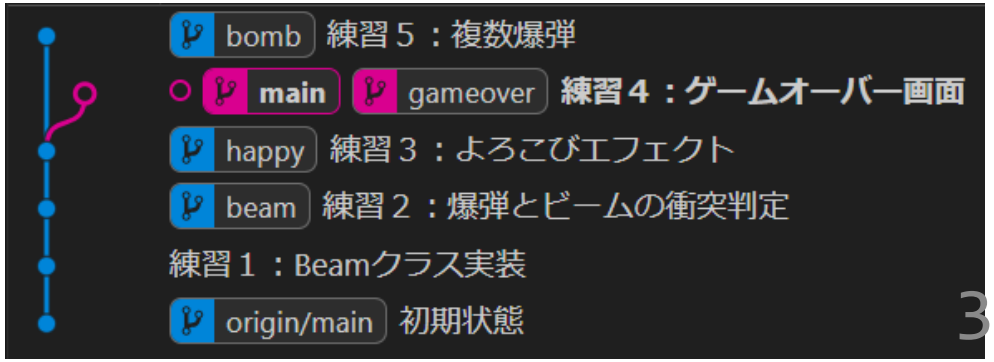
- mainブランチに，gameoverブランチをマージする
- mainブランチから見て，gameoverブランチは**直列**なので，
  Fast-Forwardマージされる ＝ mainがgameoverの場所に来るだけ

ある行を追加／削除しただけ







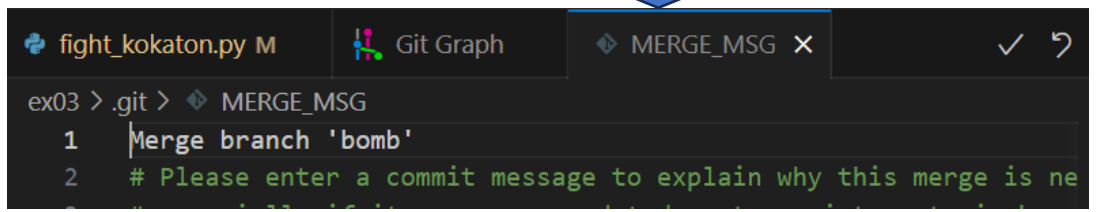


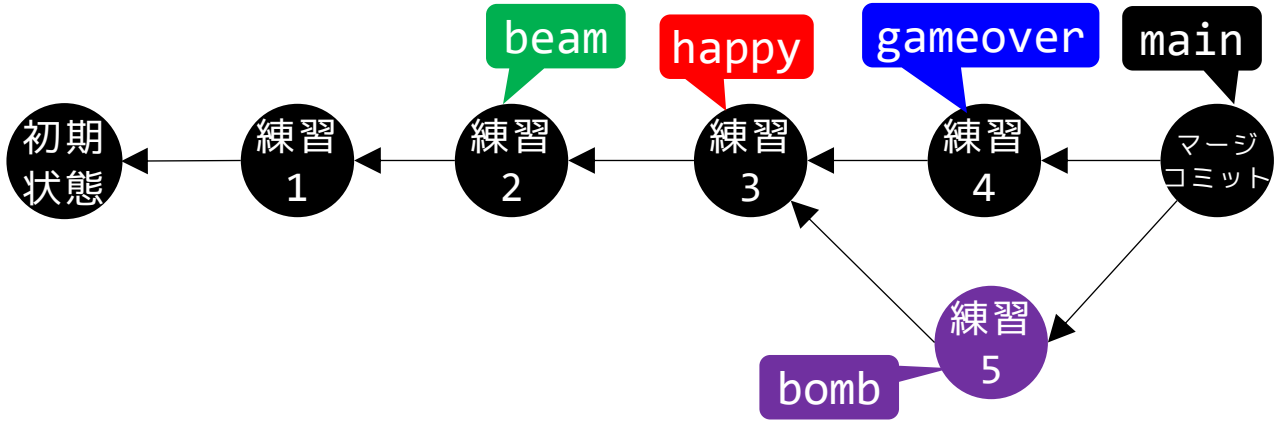
ブランチをマージしてみる（2/2）

- mainブランチに，bombブランチをマージする
- mainブランチから見て，bombブランチは**並列**なので，
  マージコミットが必要　※枝分かれしているので直列つなぎできない
  - コミットメッセージを入力するためのVScodeが裏に立ち上がる
  - コミットメッセージを入力し，保存し，閉じる

bombブランチには，gameoverブランチの変更が反映されてない

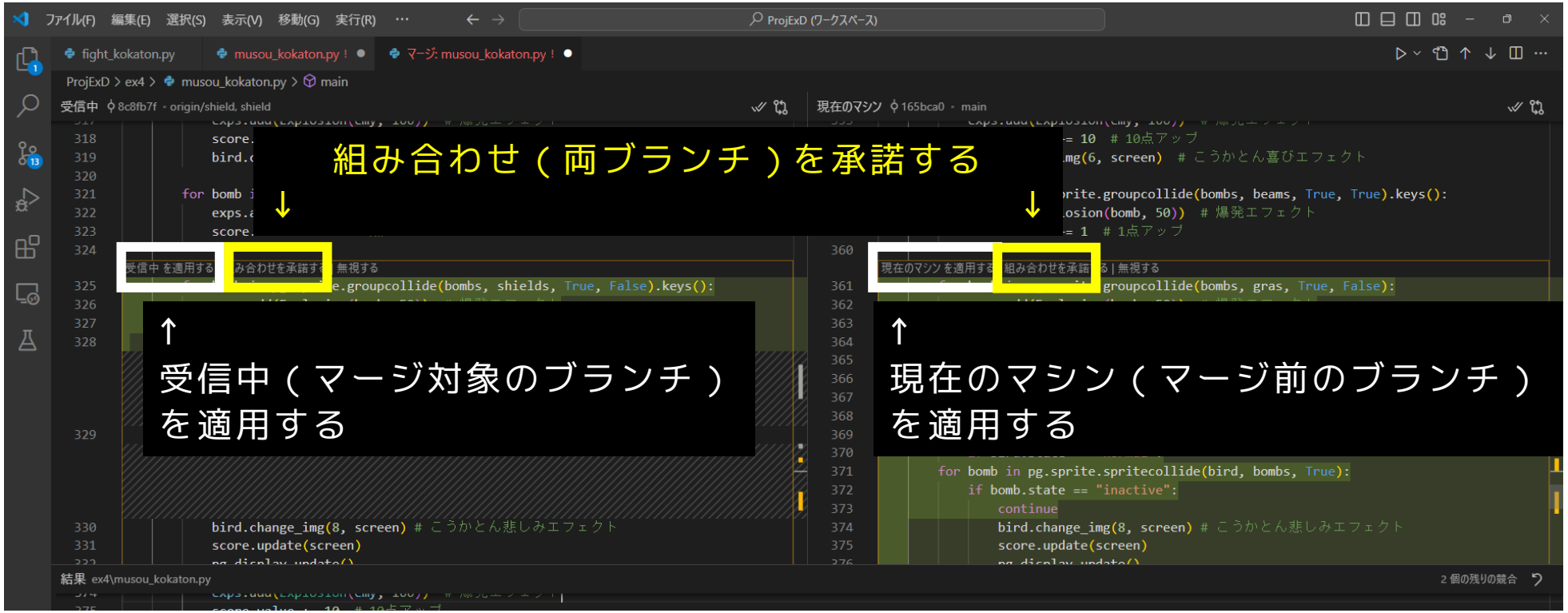
```
Admin@MSI MINGW64 ~/Desktop/ProjExD/ex3 (main)
$ git merge bomb
Auto-merging fight_kokaton.py
hint: Waiting for your editor to close the file...
```





コンフリクトの解消

- 複数のブランチで並行して変更を加えると，コンフリクト（競合）が発生する．
  （コードの同一箇所に変更を加えた場合に発生する）
- マージエディタ↓で解消する → マージの完了 → 動作確認 → ステージング・コミット



※組み合わせを承諾し，不要な部分を手動で削除するのが無難

マージを中止したい場合

- コンフリクト発生時は，作業ディレクトリもステージも.gitディレクトリ内も，マージ途中の状態である
- コンフリクトを解消しない＝マージを中止したい場合は `git merge --abort` によりマージ前の状態に戻す

4限：Pygameの演習

始める前に，mainブランチをプッシュしておこう：`git push origin main`

演習課題：「たたかえ！こうかとん」の改良

- 以下の機能を，機能ごとのブランチにて実装せよ
  1. 打ち落とした爆弾の数を表示するスコアクラス
  2. 複数のビームを打てるようにする
  3. 爆弾打ち落とし時の爆発エフェクトクラス
  4. こうかとんの向きに応じたビーム
  5. その他，独自の機能

各ブランチでの作業

- ブランチでの作業が**一段落**したら，
  1. 保存→ステージング→コミットする
     - どの追加機能を実装したのかわかるように，
       コミットコメントに機能番号を含めること（採点時に必要）
     - 例：`git commit –m "追加機能１：とりあえず完了"`
     - 例：`git commit –m "追加機能１：変数名を統一"`
  2. 機能ブランチで機能ブランチをプッシュ：`git push origin 機能ブランチ名`

- 追加機能が**完成**したら，
  1. コンフリクトを防ぐために，mainにマージ：
     `git switch main` → `git merge ブランチ名`
  2. mainブランチから新たなブランチを作り，次の機能を実装する
- 追加機能が完成していない or バグがあるブランチはマージしない

1.スコア表示：scoreブランチ

- Scoreクラス
  - イニシャライザ
    - フォントの設定：`self.fonto = pg.font.SysFont("hgp創英角ﾎﾟｯﾌﾟ体", 30)`
    - 文字色の設定：青 → (0, 0, 255)
    - スコアの初期値の設定：0
    - 文字列Surfaceの生成：`self.img = self.fonto.render(“表示させる文字列”, 0, 色)`
    - 文字列の中心座標：画面左下(横座標：100, 縦座標：画面下部から50)
  - updateメソッド
    - 現在のスコアを表示させる文字列Surfaceの生成
    - スクリーンにblit
- main関数
  - (初期化)Scoreインスタンスの生成
  - (ループ)爆弾を打ち落としたらスコアアップ（1点）
  - (ループ)updateメソッドを呼び出してスコアを描画

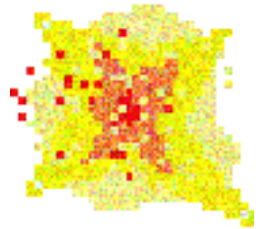
↑ ex1/japanese_font.pyの14行目を参照

2.複数ビーム：multibeamブランチ

- main関数(初期化)
  - Beamクラスのインスタンスを複数扱うための空のリストを作る
- main関数(ループ)
  - スペースキー押下でBeamインスタンス生成，リストにappend
  - リストの要素1つずつに対して爆弾と衝突判定し，衝突した要素はNoneとする
  - ビームリストに対して，要素がNoneでないものだけのリストに更新
  - 画面の範囲外に出たらリストから削除する ← しないとリストが大きくなる

※この追加機能は新たなクラスを定義しないので，無条件で１点加算

3.爆発エフェクト：explosionブランチ



- Explosionクラス
  - イニシャライザ
    - 元のexplosion.gifと上下左右にflipしたものの2つのSurfaceをリストに格納
    - 爆発した爆弾の`rct.center`に座標を設定
    - 表示時間（爆発時間）`life`を設定
  - updateメソッド
    - 爆発経過時間`life`を１減算
    - 爆発経過時間`life`が正なら，Surfaceリストを交互に切り替えて爆発を演出

↑目がチラチラしないように工夫しよう

- main関数
  - (初期化)Explosionインスタンス用の空リストを作る
  - (ループ)bombとbeamが衝突したらExplosionインスタンスを生成，リストにappend
  - (ループ)`life`が0より大きいExplosionインスタンスだけのリストにする
  - (ループ)updateメソッドを呼び出して爆発を描画

- その他：explosion.gifをex3/fig/フォルダに配置

4.向きに応じたビーム：directionブランチ

- Birdクラスを改良
  - イニシャライザ：こうかとんの向きを表すタプル`self.dire=(+5,0)`を定義 ↓デフォルト右向き
  - updateメソッド：合計移動量`sum_mv`が`[0,0]`でない時，`self.dire`を`sum_mv`の値で更新
- Beamクラスを改良
  - イニシャライザ
    - Birdの`dire`にアクセスし，こうかとんが向いている方向を`vx, vy`に代入
    - `math.atan2(-vy, vx)`で，直交座標`(x, -y)`から極座標の角度`θ`に変換
    - `math.degrees(θ)`で弧度法から度数法に変換し，`rotozoom`で回転

← ここまでとりあえずやってみよう
← その後ビームの位置ずれを調整しよう

    - こうかとんの`rct`の`width`と`height`および向いている方向を考慮した初期配置
      - ビームの中心横座標＝こうかとんの中心横座標＋こうかとんの横幅×ビームの横速度÷5
      - ビームの中心縦座標＝こうかとんの中心縦座標＋こうかとんの高さ×ビームの縦速度÷5

※この追加機能は新たなクラスを定義しないので，無条件で１点加算
※pygameのバージョンが古いと，右上，右下はビームが出ないけど気にしないこと．

5限：コードレビュー／ブランチ間差分表示

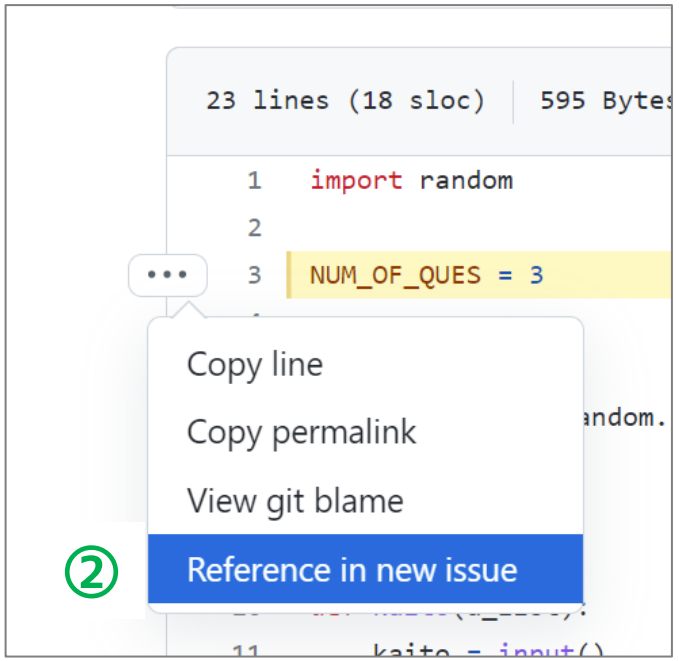
始める前に，完成したブランチをマージしたmainブランチをプッシュしよう：
`git push origin main`
※画像ファイルなど，ゲーム実行に必要なものはすべてプッシュすること

コードレビューの手順

1. グループメンバー1人の公開リポジトリURLを入手する
2. コードを見て，修正すべき点をIssueにより送信する
   - 修正すべき点がない完璧なコードの場合，他のメンバーにレビューを依頼する
   - それでも修正すべき点がない場合，TASA，教員にレビューを依頼する

Issueを送る手順

1. グループメンバーのGitHubのページにアクセスし，当該コードを開く
   https://github.com/アカウント名/ProjExD_3/blob/main/fight_kokaton.py
2. コードの該当行を選択し，「・・・」→「Reference in new issue」をクリックする
3. タイトルとコメントを書く
   - 対象が複数行の場合，手動で行数を追加する：「#L13」→「#L13-L16」



Issueを送る手順

4. Previewで確認する
5. 「Submit new issue」をクリックし送信する

Issueが届く

- Issuesタブから，Issueコメントを読み，対応する

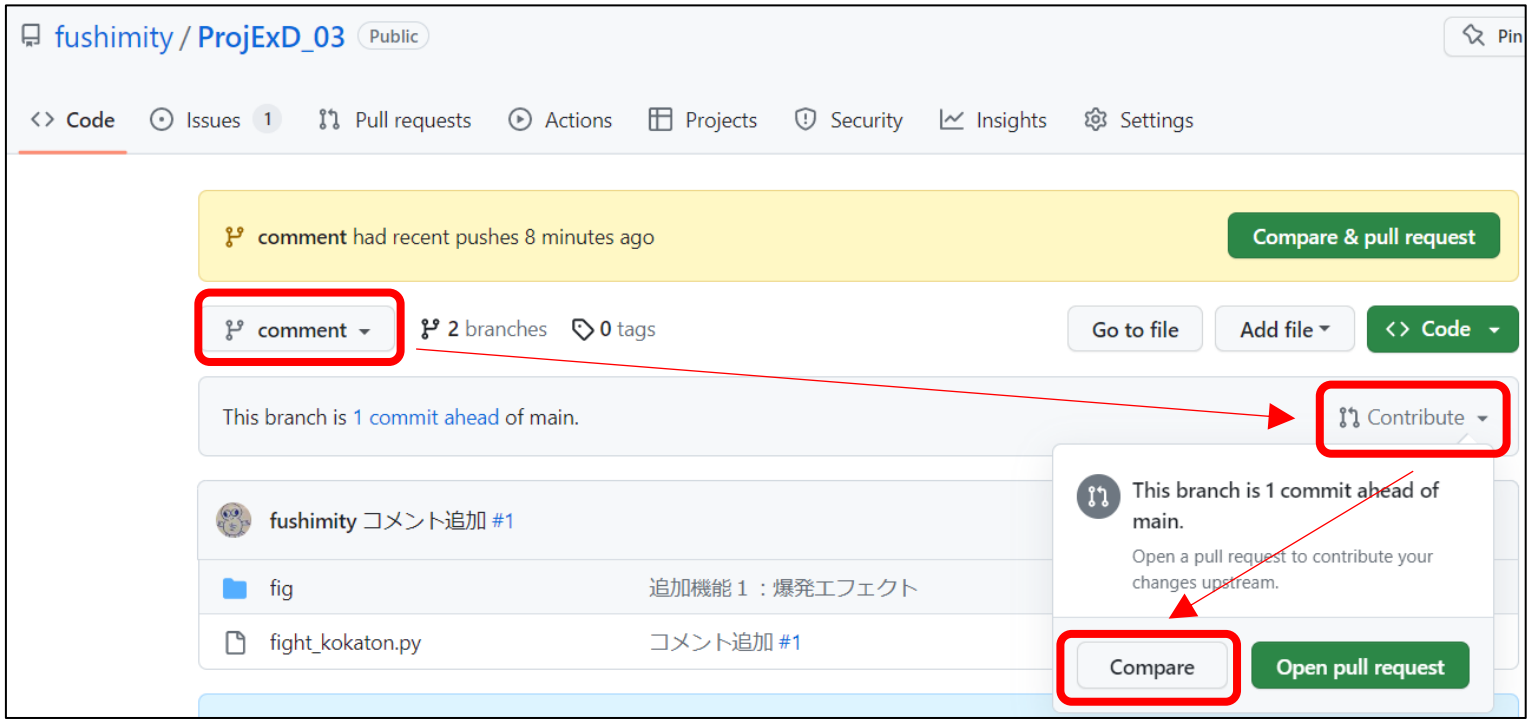
コード修正の手順

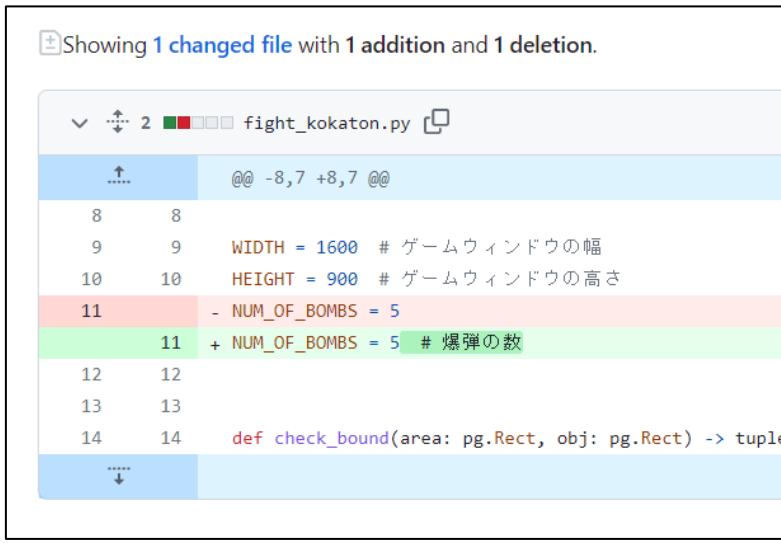
1. 新たなブランチを切り，切り替える：`git switch -c issue1`
   - ブランチ名は「`issue1`」のようにIssue番号を付けること
2. 修正が終わったらステージングする：`git add ファイル名`
3. ステージングされた内容をコミットする
   ※重要：コミットコメントに，対応したIssue番号を付けること
   `git commit -m "コメント #番号"`
   - 「#」の前に半角スペースが必要
   - 「#番号」は半角で入力する

   例：`git commit -m "変数名統一 #1"`
4. 上記ブランチをプッシュする：`git push origin issue1`
   - 今回は，mainブランチにマージしないこと
5. GitHub上で，mainブランチと上記ブランチを差分表示させる
   - 次ページ参照

ブランチ間の差分表示

- 表示するブランチを切り替え，「Contribute」→「Compare」で https://github.com/アカウント名/ProjExD_3/compare/issue1 にアクセス→PDF化





チェック項目

1. 追加機能の実装数(各機能に対して3点満点／最大12点)
   - 新たなクラスを定義している：1点
   - 新たなブランチで開発している：1点
   - バグなく動作する：1点

2. 複数のブランチをプッシュしているか：1点
3. Issueブランチでコメント対応している：1点
4. Issue番号を付してコミットし，紐づけできている：1点

5. 提出物不備（ファイル名，クリッカブル，内容物）は1点ずつ減点

提出物

学籍番号は，半角・大文字で

- ファイル名：C0A24XXX_kadai3.pdf
- 内容：以下の順番でPDFを結合して提出すること
  - コードの最終版（Issue対応をマージしていないmainブランチの最終版）
    https://github.com/fushimity/ProjExD_3/blob/main/fight_kokaton.py
  - ブランチ一覧
    https://github.com/fushimity/ProjExD_3/branches
  - Issue対応ブランチの差分表示
    https://github.com/fushimity/ProjExD_3/compare/issue1

自分のアカウント名

Issue対応ブランチ名

提出物の作り方

※スクショ画像は認めません．
以下の手順に従ってPDFを作成し，提出すること

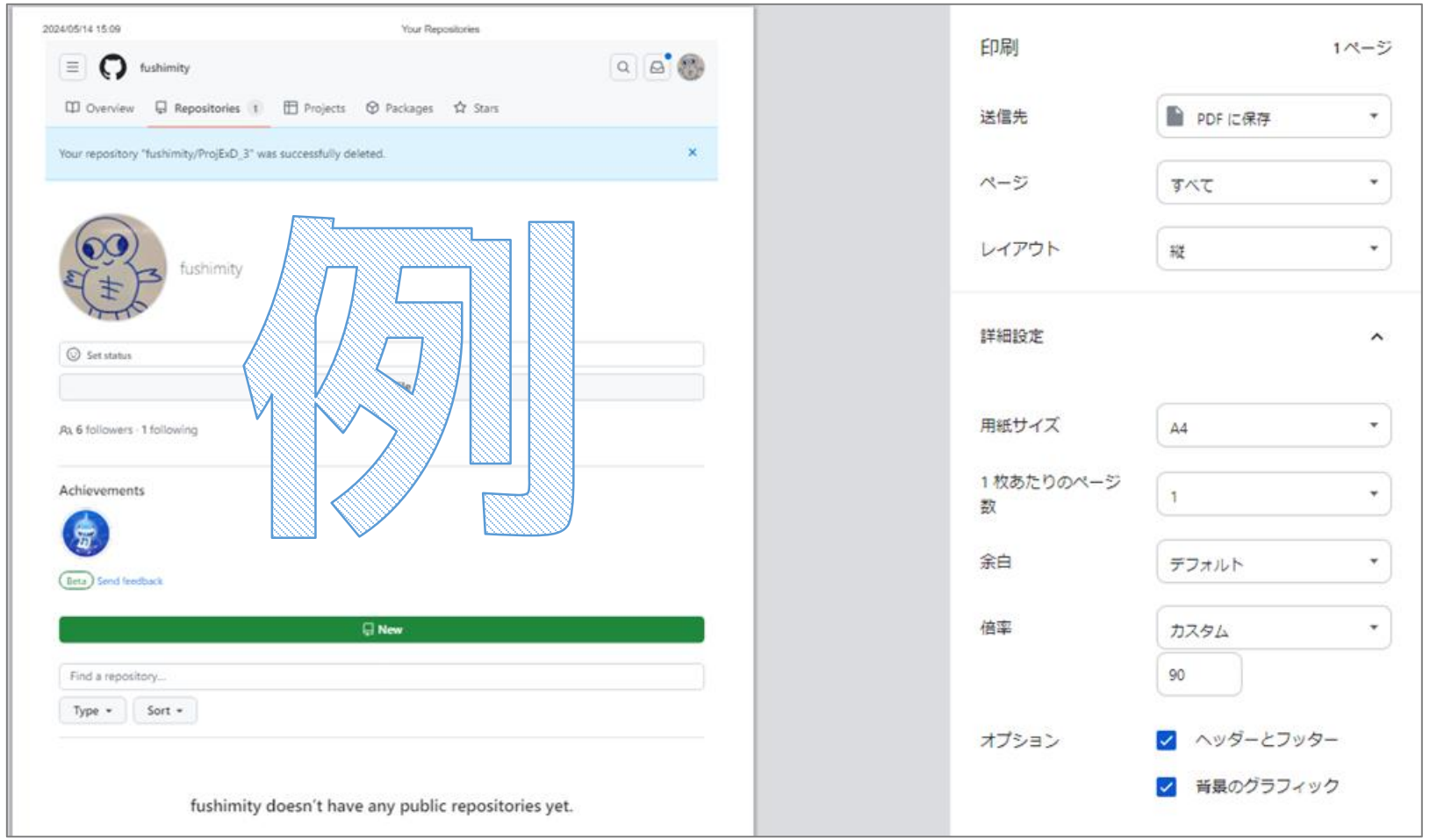
1．ChromeでPDFとして保存する（次ページを参照）

2．以下のURLから各PDFをマージする
https://www.ilovepdf.com/ja/merge_pdf

3．ファイル名を「C0A24XXX_kadai3.pdf」として保存する

ChromeでPDFとして保存する方法

1. 該当ページを表示させた状態で「Ctrl＋P」
2. 以下のように設定し，「保存」をクリックする



Microsoft Print to PDFはダメ

←送信先：PDFに保存

←ページ：すべて

←レイアウト：縦

←用紙サイズ：A4

←余白：デフォルト

←倍率：90

←両方チェック

チェックの手順

※基本的に，TASAチェックを何度も受けることはできませんので，
慎重に提出物PDFを作ること．どうしてもの場合は要相談．

1．受講生：提出物（pdf）を作成する
2．受講生：担当TASAに提出物と成果物（ゲーム）を見せに行く
3．TASA：提出物とゲームのデモを確認し，点数を確定する
4．受講生：提出物をMoodleにアップロードして，帰る
5．FSM：近日中に課題と点数を確認し，Moodleに登録する

- 時間内にチェックが終わらなそうな場合は，
  提出物をMoodleに提出し帰る
  （次回までor次回の3限にチェックされる）

← 時間外提出扱いになり
割引いて採点するので
できるだけチェックを
受けること