2023年6月27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／Pygameの基礎／Git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

- MoodleからEx03.zipをDLし，ProjExD2023フォルダの下に配置

【配布物配置後のディレクトリ構造】

- ProjExD2023/
  - sample.py
  - ex02/
  - ex03/　　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

gitでの作業

- git bashを起動し，ex03フォルダに移動する：`cd Desktop/ProjExD2023/ex03`
- ex03フォルダで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 *`
- 「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_03」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/fushimity/ProjExD_03.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`

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

```python
import random
import sys
import time

import pygame as pg


WIDTH = 1600  # ゲームウィンドウの幅
HEIGHT = 900  # ゲームウィンドウの高さ


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


class Bird:…


class Bomb:…


def main():…


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

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
- 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 ←前回の追加機能候補の1つ
  - 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
  - キーの押下状態と移動量辞書に応じて移動後の位置を求め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

コード解説（Bombクラス）

- イニシャライザ：__init__ ←前回main関数内while文前に記述した内容
  - 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main関数内while文内に記述した内容
  - 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

コードの解説（main関数）

```
bird = Bird(3, (900, 400))
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．

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`

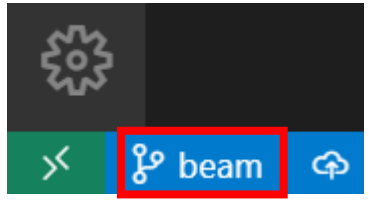
```
admin@LAPTOP-63464884 MINGW64 ~/Desktop/ProjExD2023/ex03 (main)
$ git branch beam

admin@LAPTOP-63464884 MINGW64 ~/Desktop/ProjExD2023/ex03 (main)
$ git branch
  beam
* main

admin@LAPTOP-63464884 MINGW64 ~/Desktop/ProjExD2023/ex03 (main)
$ git switch beam
Switched to branch 'beam'

admin@LAPTOP-63464884 MINGW64 ~/Desktop/ProjExD2023/ex03 (beam)
```

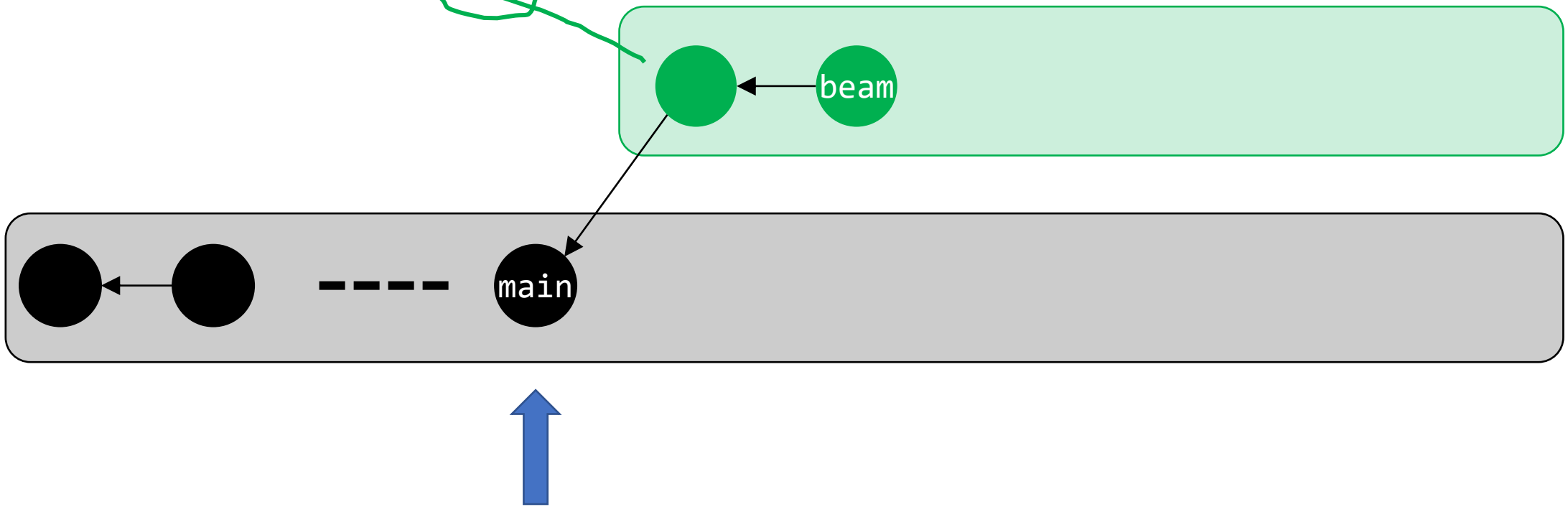
←現在のブランチ



コミットとブランチ

別ブランチとしての
１回目のコミット

switchでブランチ間を移動できる
＝パラレルワールドを行き来できる



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

練習問題

※1問ずつステージング，コミットしよう

1. beamブランチにて，Beamクラスを実装せよ．
   - イニシャライザ：beam.pngのSurfaceを生成
   - イニシャライザ：こうかとんの右に配置

   ←Birdクラスを参考に

   - イニシャライザ：速度は横方向に５／縦方向に０
   - updateメソッド：初期位置から右に移動＋画面SurfaceにBlit

   ←Bombクラスを参考に

   - main関数：スペースキー押下でBeamインスタンスを生成
   - main関数：爆弾との衝突で，Beamインスタンス，Bombインスタンスを消滅
   - main関数：updateメソッドを呼び出して更新する
   - その他：beam.pngをex03/fig/フォルダに配置
- 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）

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`

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@LAPTOP-63464884 MINGW64 ~/Desktop/ProjExD2023/ex03 (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
```

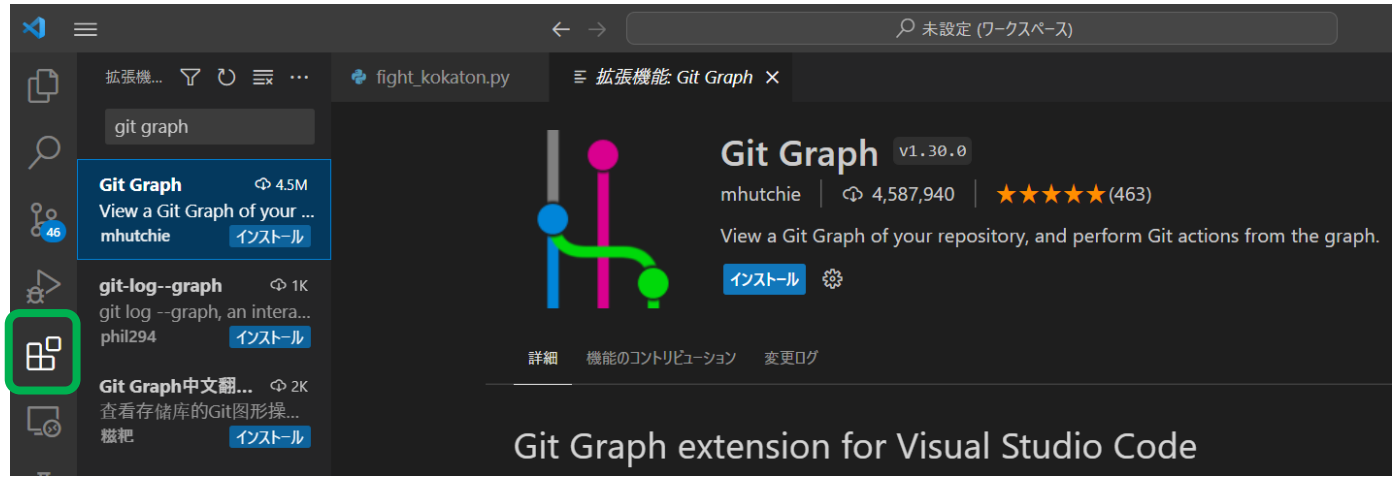
- スタッシュしてから，
  mainブランチにスイッチする → エラーなくスイッチできた

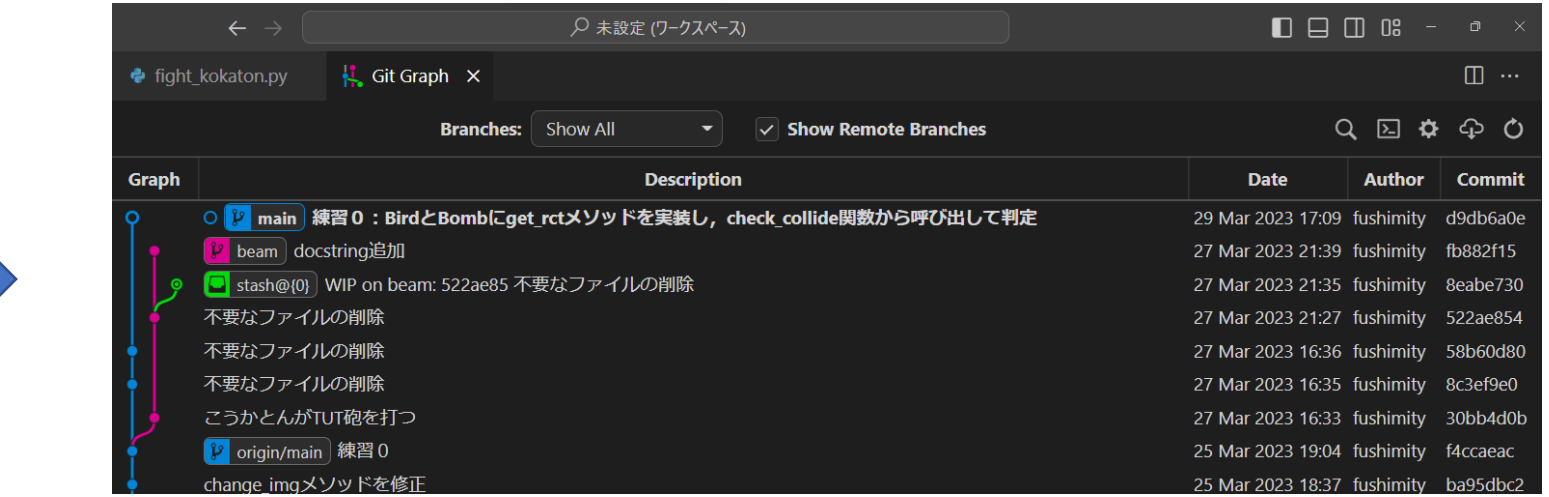
消したBirdクラスが戻っている＝最後にコミットした状態になる

- 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

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





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．

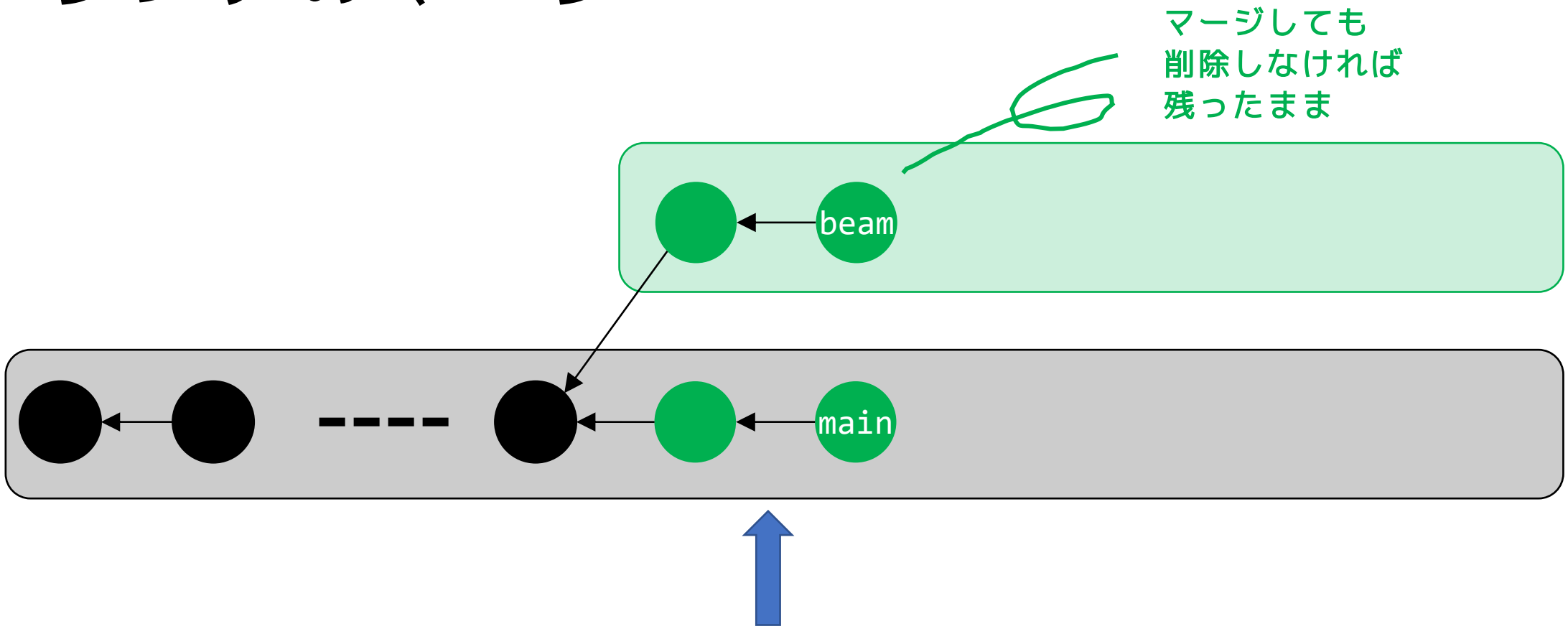
【beamブランチをmainブランチにマージする場合】

- 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`　← やってみよう

```
admin@LAPTOP-63464884 MINGW64 ~/Desktop/ProjExD2023/ex03 (beam)
$ git switch main
Switched to branch 'main'

admin@LAPTOP-63464884 MINGW64 ~/Desktop/ProjExD2023/ex03 (main)
$ git merge beam
Updating d832ee1..6e0c135
Fast-forward
 fight_kokaton.py | 1 +
 1 file changed, 1 insertion(+)
```

ブランチのマージ

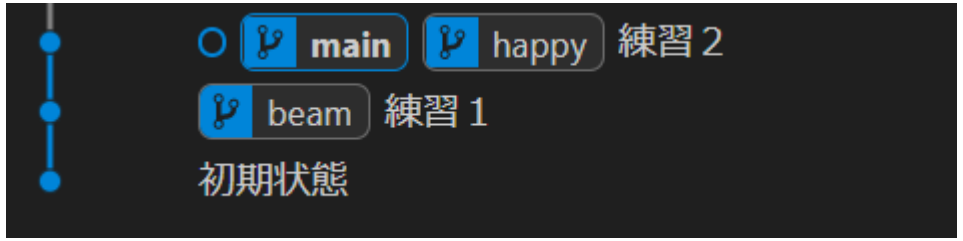


2回のコミット履歴を
mainブランチにマージした結果

練習問題（つづき）

2. happyブランチにて，爆弾を打ち落とし時に**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（つづき）

3. switchブランチにて，飛ぶ方向に従ってこうかとん画像が
   切り替わる機能を実装せよ．

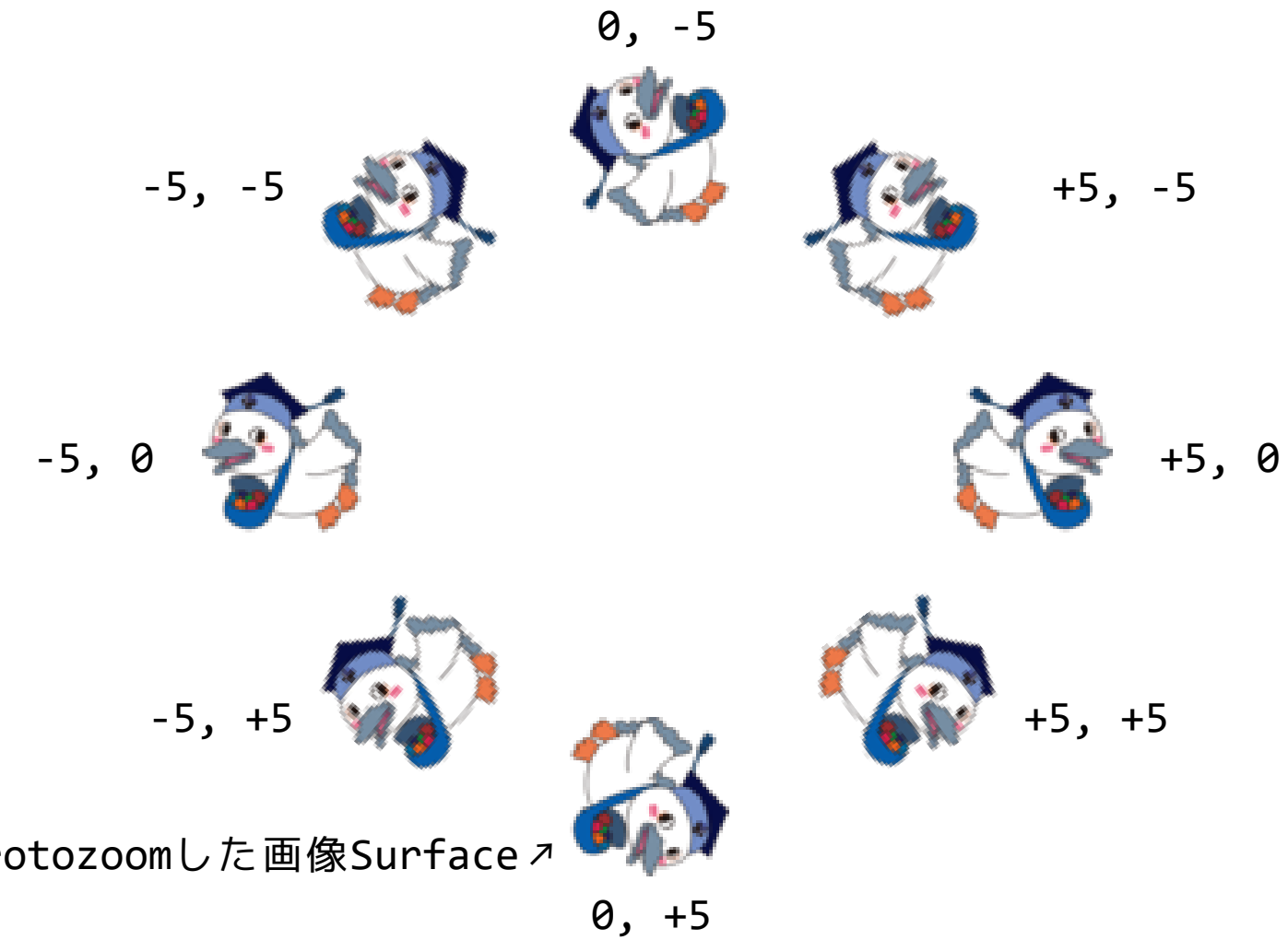
Birdクラス
- イニシャライザ：移動量の合計値タプル vs 回転後画像Surfaceの辞書を定義
- イニシャライザ：上記辞書から（+5，0）の画像Surfaceをデフォルトとして選択
- updateメソッド：上記辞書から押下キーの合計値に応じた画像Surfaceを選択

- 実装→ステージング→コミット→mainブランチにスイッチ
  →マージせずに次の練習に取り組んでみよう

3.飛ぶ方向に従ってこうかとん画像を切り替える

- 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
  の合計値タプルをキー，
  rotozoomしたSurfaceを
  値とした辞書を用意しておく



練習問題（つづき）

switch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

4. bombブランチにて，複数の爆弾を実装せよ．

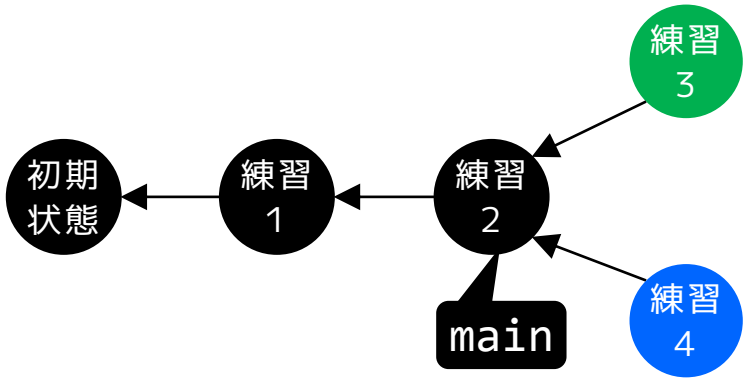
- トップ：爆弾の数を表す定数NUM_OF_BOMBSを定義

Bombクラス
- イニシャライザ：色，移動方向，サイズをランダムに決定
  - 色と方向は，候補リストを作っておき，random.choiceするとよい
- イニシャライザ：上記でランダムに選択するため，引数は不要になる

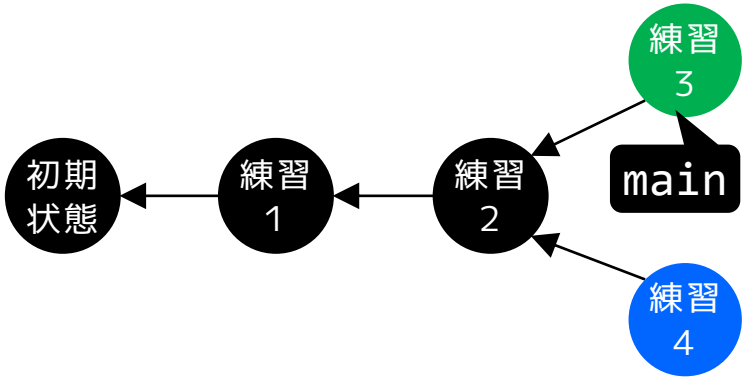
- main関数：NUM_OF_BOMBS個のBombインスタンスを要素とするリストを生成
- main関数：リストの要素1つずつに対して，
  こうかとん／ビームと衝突判定し，衝突した要素はNoneとする
- main関数：爆弾リストに対して，要素がNoneでないものだけのリストに更新

ブランチをマージしてみる（1/2）

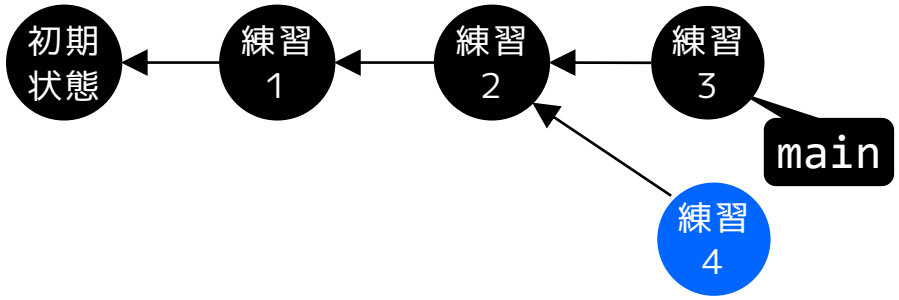
- main, switch, bombが独立している（同じ部分の修正がない）ので，コンフリクト（競合）は発生しないはず



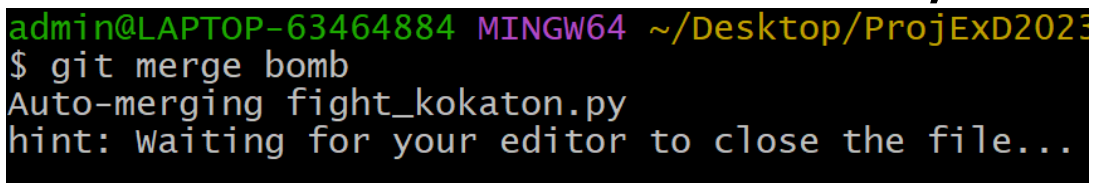
- switchブランチをマージする → Fast-Forwardマージされる
  = mainがswitchの場所に来るだけ（枝分かれしているが，直列）

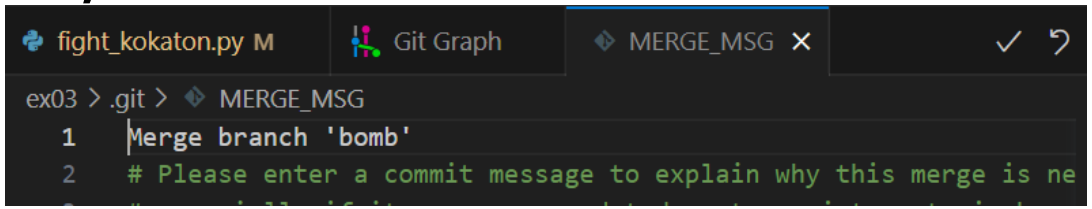


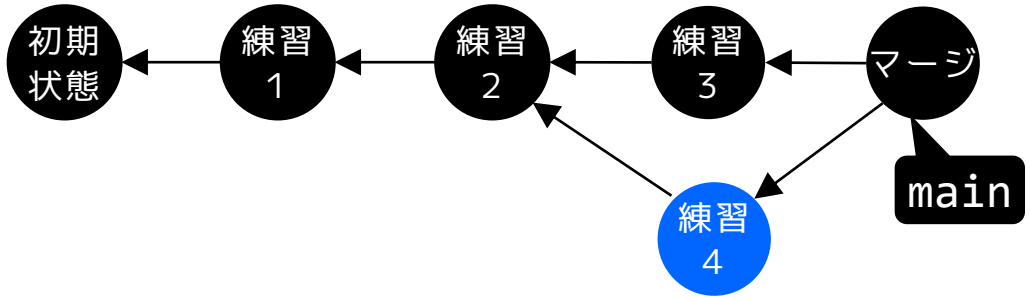
ブランチをマージしてみる（2/2）



- bombブランチをマージする → 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

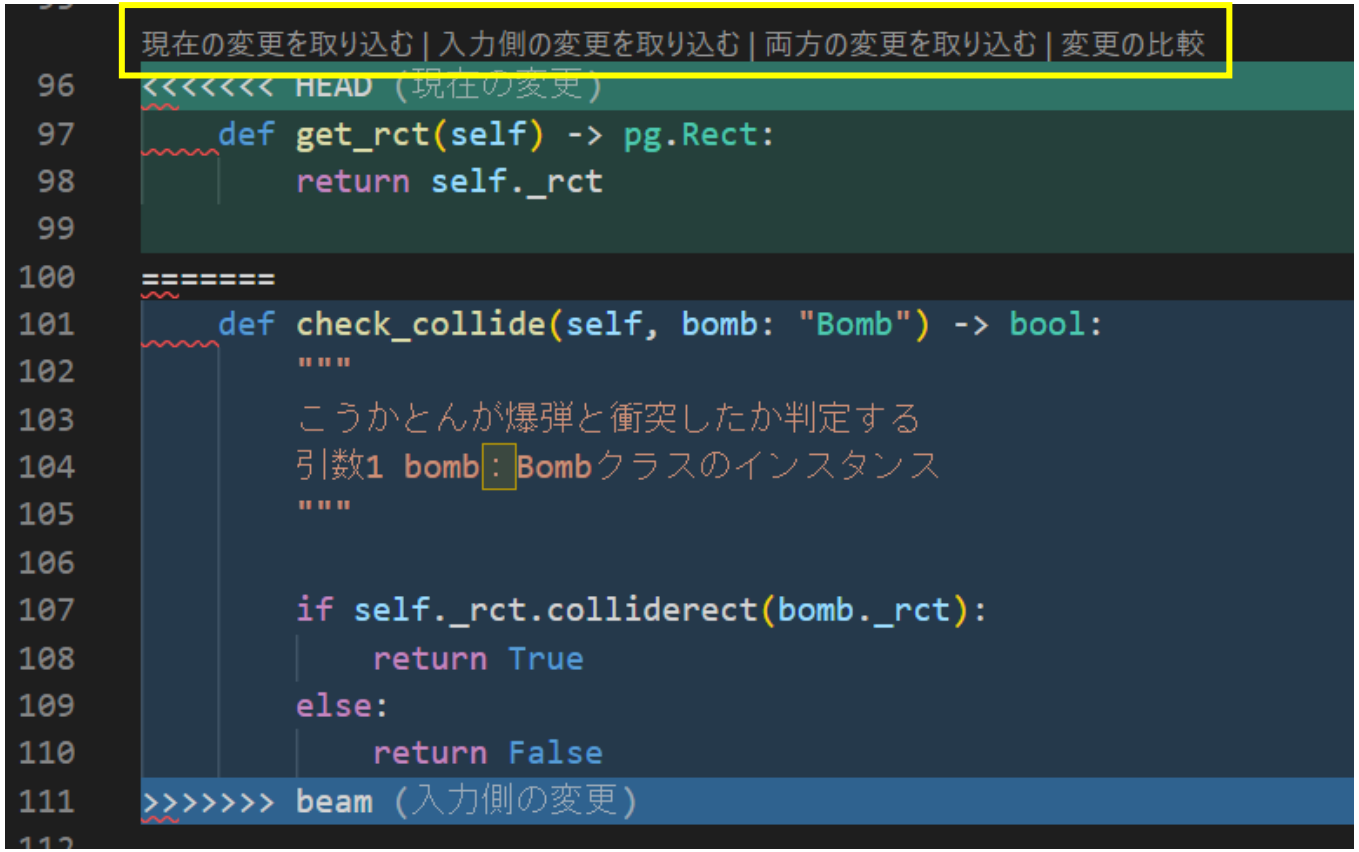






コンフリクトの解消

- 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
- エディタにおいてコンフリクトを手動で解消して，コミットする必要がある．



← どちらのブランチの変更を採用するかを
判断してから，ここをクリックする

← あるいは，手動で不要部分を消したり，
追記したりすることもできる．

全てのコフリクト箇所に対応したら，
いつもと同じように
保存，ステージング，コミットする．

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

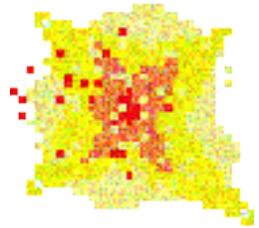
- 以下の機能を，機能ごとのブランチにて実装せよ
  1. 爆弾打ち落とし時の爆発エフェクトクラス
  2. こうかとんの向きに応じたビーム
  3. 打ち落とした爆弾の数を表示するスコアクラス
  4. 複数のビーム
  5. その他，独自の機能

各機能の実装完了後に

- 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"`

- 1つの機能を実装し終えたら，
  - ブランチをプッシュ：`git push origin ブランチ名`
  - 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`
  - 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

1.爆発エフェクト：explosionブランチ



- Explosionクラス
  - イニシャライザ
    - ex03/fig/explosion.gifと上下左右にflipしたものを画像リストに格納
    - 爆発した爆弾のrct.centerに基づき，生成場所を決定
    - 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の値に応じて，画像リストを交互に切り替えて爆発を演出

- main関数
  - 複数爆弾と同様に，生成した爆発をリストに格納
  - lifeが0より大きい爆発インスタンスだけのリストに更新

- その他：explosion.gifをex03/fig/フォルダに配置

2.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：方向タプル用のインスタンス変数`self.dire=(+5,0)`を定義
  - updateメソッド：移動量の合計値タプルで`self.dire`を更新
- Beamクラスを改良
  - イニシャライザ
    - 上記変数にアクセスし，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

3.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.font = 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.font.render("表示させる文字列", 0, 色)`
    - 文字列の中心座標：画面左下 → 100，画面下部から50の位置
  - 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
  - Scoreインスタンスの生成
  - 爆弾を打ち落としたらスコアアップ（1点）
  - updateメソッドの呼び出し

4.複数のビーム：multibeamブランチ

- Beamクラスのインスタンスを複数扱うためのリストを作る
- 爆弾と衝突したらリストから削除する
- 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

5限：コードレビュー／ブランチ間差分表示

始める前に，全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

- グループメンバー1人の公開リポジトリURLを入手する
- 次ページを参照し，クローン＋ローカルでゲームを実行してみる
- 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
- 自分の作業ディレクトリに戻る
- Issueを受信したら，新たなブランチを切ってコードを修正する
  - 詳細は，5ページ後ろの「コード修正の手順」を参照のこと
- 修正後は，マージせずに上記ブランチをプッシュする
- mainブランチと上記ブランチの差分を表示させる

クローンと実行

- git bashでProjExD2023フォルダに移動する：`cd ../`
- 他メンバーのリポジトリをクローンする：
  `git clone https://github.com/メンバーのアカウント名/ProjExD_03.git`

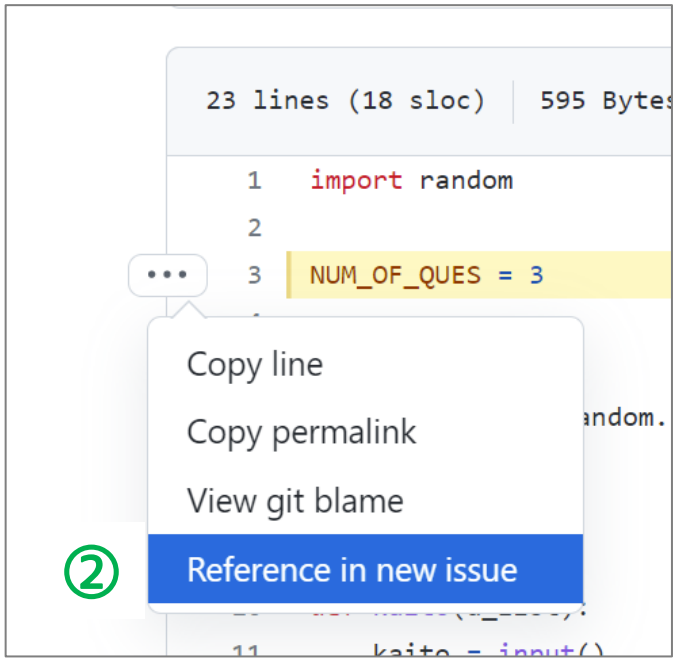
  ※ex03フォルダではなく，ProjExD2023フォルダでcloneすること

- クローンしたコードをVScodeで開き，「Ctrl＋F5」で実行する

- 用が済んだら，自分のリポジトリのフォルダに戻る：`cd ../ex03`

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/fushimity/ProjExD_03/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

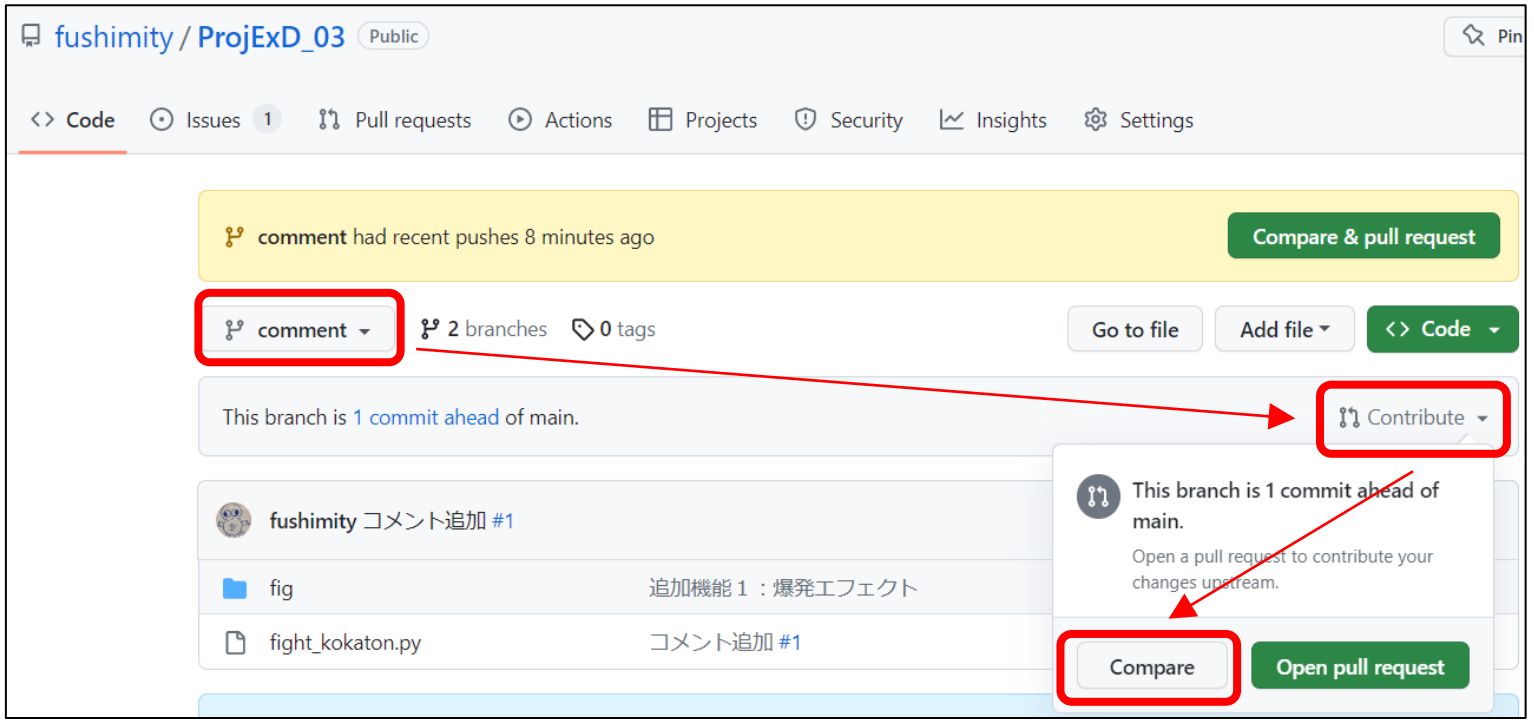
- Issuesタブから，Issueコメントを読み，対応する

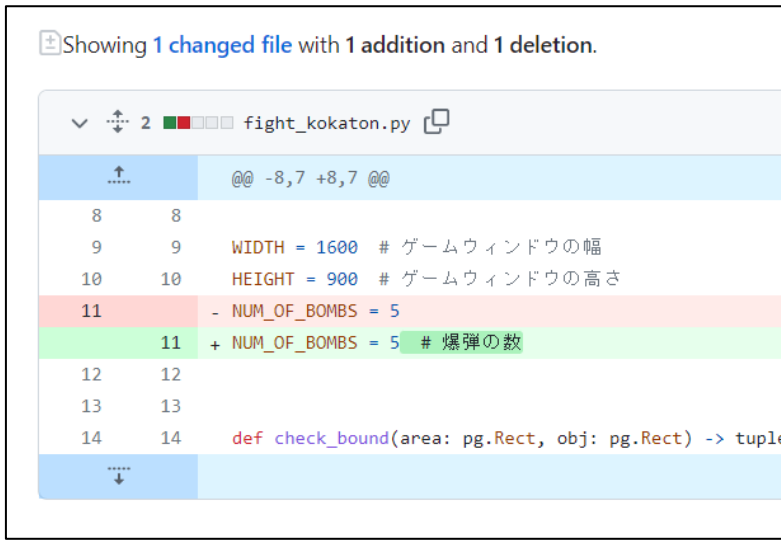
コード修正の手順

0. 自分の作業ディレクトリに戻る
1. 新たなブランチを切り，切り替える：`git switch -c ブランチ名`
   - 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 #Issue番号"`
   - 「#」の前に半角スペースが必要
   - 「#Issue番号」は半角で入力する
4. 上記ブランチをプッシュする：`git push origin ブランチ名`
   - 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_03/compare/ブランチ名 にアクセス→PDF化





提出物

学籍番号は，半角・大文字で

- ファイル名：C0A22XXX_kadai03.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_03/blob/main/fight_kokaton.py
  - ブランチ一覧
    https://github.com/fushimity/ProjExD_03/branches
  - Issue対応ブランチの差分表示
    https://github.com/fushimity/ProjExD_03/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

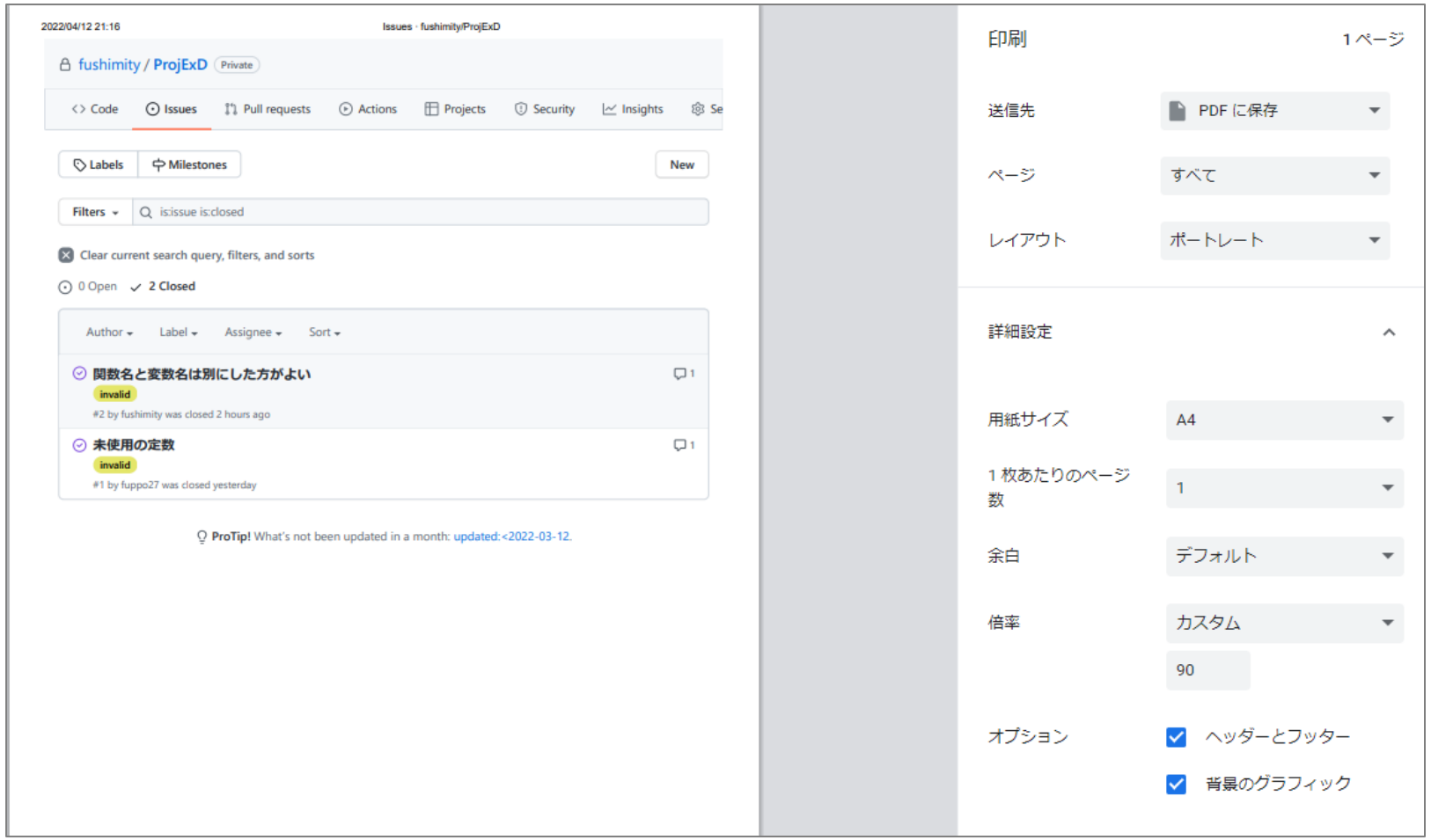
1. ChromeでPDFとして保存する（次ページを参照）

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「C0A22XXX_kadai03.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



←送信先：PDFに保存

←ページ：すべて

←レイアウト：ポートレート

←用紙サイズ：A4

←余白：デフォルト

←倍率：90

←両方チェック

チェック項目

1. 追加機能の実装数(4機能まで) [0 ～ 8]
   - 新たなクラス定義を伴う追加機能：2点
   - 上記以外の追加機能：1点

2. 複数のブランチをプッシュしているか [0 or 2]
   - ブランチ一覧にて確認

3. Issue [0 ～ 5]
   - コード修正がある：2点
   - Issue番号を付してコミットし，紐づけできている：1点
   - mainでないブランチでIssue対応：2点

4. 提出物不備は1点ずつ減点

チェックの手順

※基本的に再提出できません．どうしてもの場合は要相談．

1. 受講生：提出物（pdf）を作成し，Moodleに提出する
2. 受講生：担当TASAに成果物（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：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