

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

- explosion.png . . . 爆発画像

← 指示があるまで追加しないで

← 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

```
1  import random
2  import sys
3  import time
4
5  import pygame as pg
6
7
8  WIDTH = 1600 # ゲームウィンドウの幅
9  HEIGHT = 900 # ゲームウィンドウの高さ
10
11
12 > def check_bound(obj_rct: pg.Rect) -> tuple[bool, bool]:...
24
25
26 > class Bird:...
77
78
79 > class Bomb:...
108
109
110 > def main():...
139
140
141 if __name__ == "__main__":
142     pg.init()
143     main()
144     pg.quit()
145     sys.exit()
146
```

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__`
 - 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
 - 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

←前回main関数内while文前に記述した内容

←前回の追加機能候補の1つ

←前回main関数内while文内に記述した内容

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)
```

← こうかちゃんと爆弾の生成を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

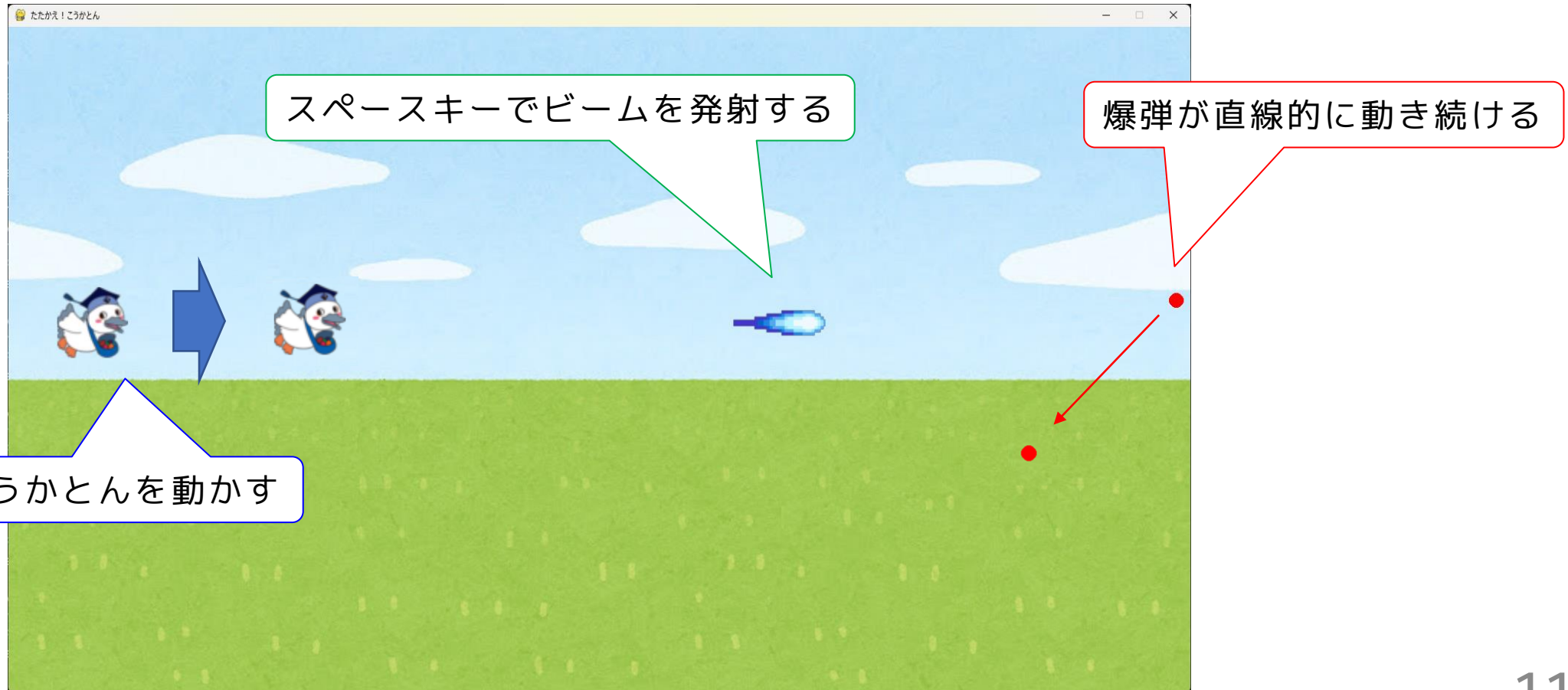
← こうかちゃんと爆弾の衝突判定

← 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ちゃん画像を切り替える

← こうかちゃ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

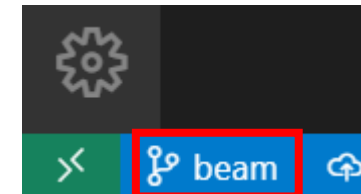
`git switch -c ブランチ名`

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

admin@LAPTOP-63464884 MINGW64 ~/Desktop/ProjExD2023/ex03 (main)
$ git branch
beam
* main      ←現在のブランチ

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

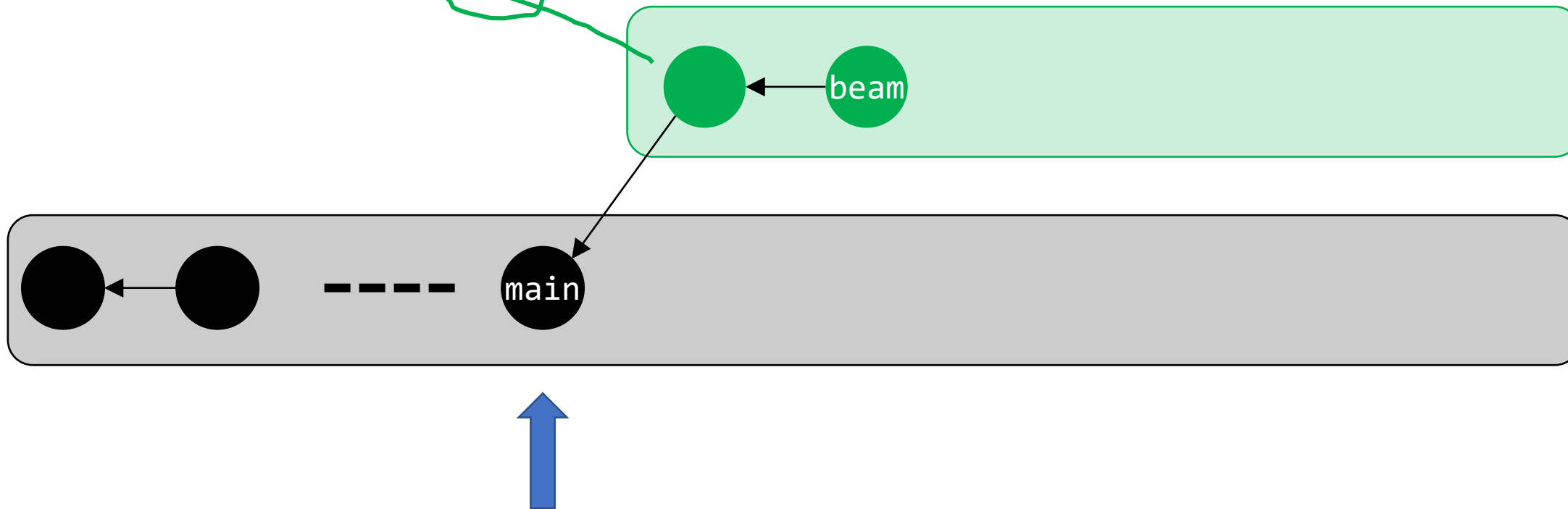
admin@LAPTOP-63464884 MINGW64 ~/Desktop/ProjExD2023/ex03 (beam)
```



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: こうかとの右に配置
 - 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
 - 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
- ←Birdクラスを参考に
- ←Bombクラスを参考に
- 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

たとえばBirdクラス
を丸ごと消してみる

- beamブランチで、コードをちょっと変更する
- 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）

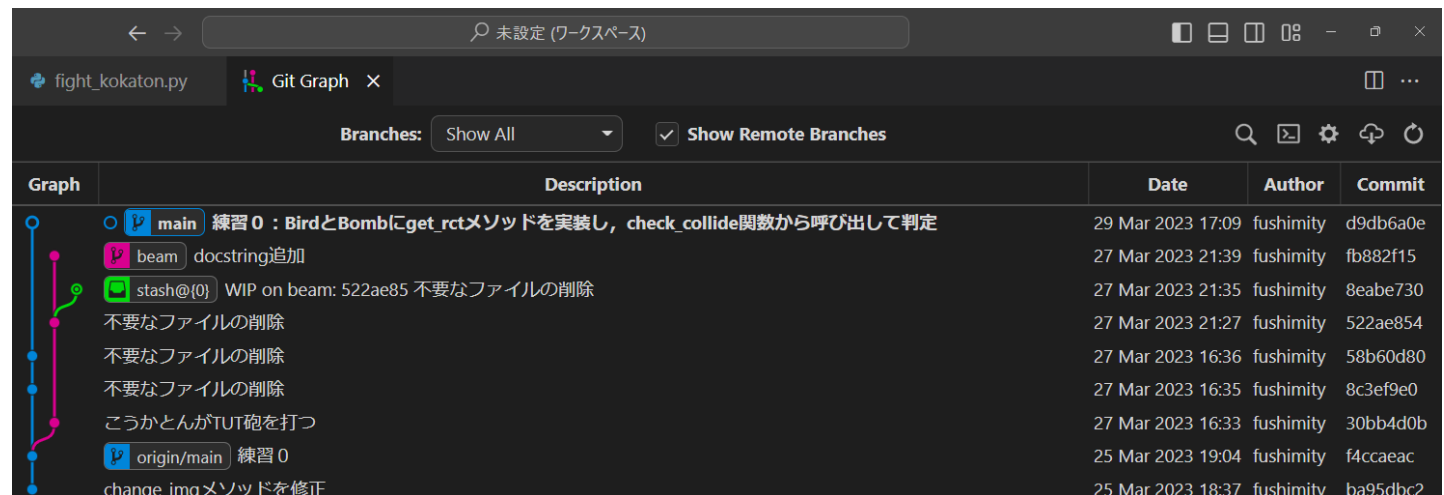
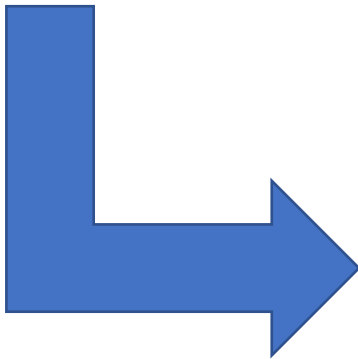
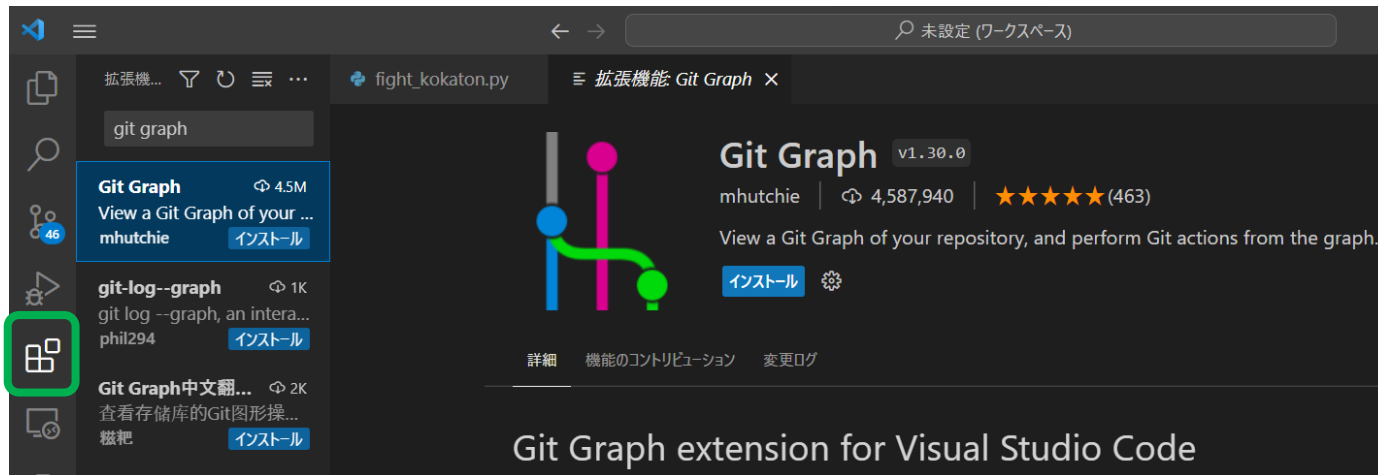
```
admin@LAPTOP-63464884 MINGW64 ~/Desktop/ProjExD2023/ex03 (beam)
$ git switch main
error: Your local changes to the following files would be overwritten by checkout:
      fight_kokaton.py
Please commit your changes or stash them before you switch branches.
Aborting
```

- 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
- 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

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

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いう。
- 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。

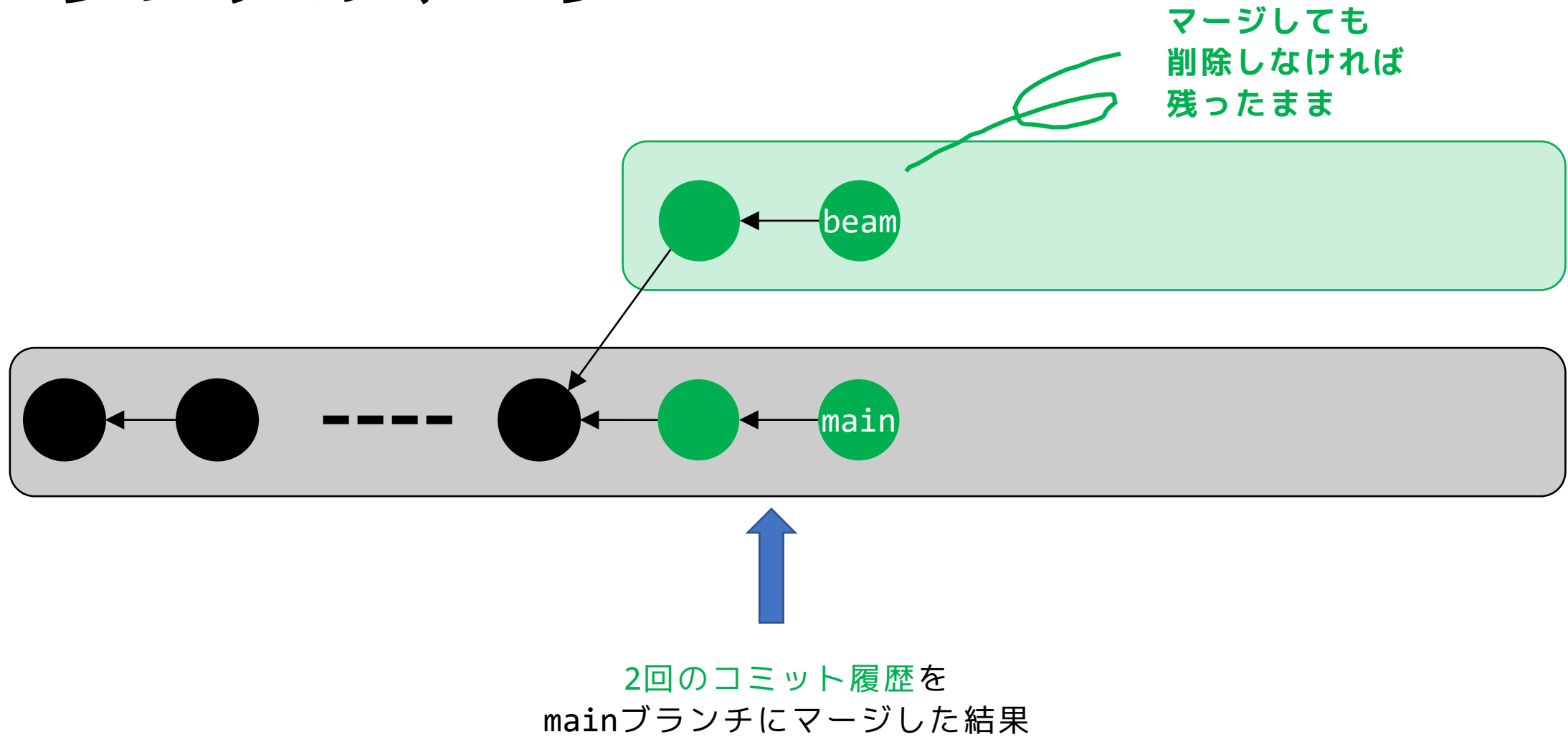
【beamブランチをmainブランチにマージする場合】

- 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(+)
```

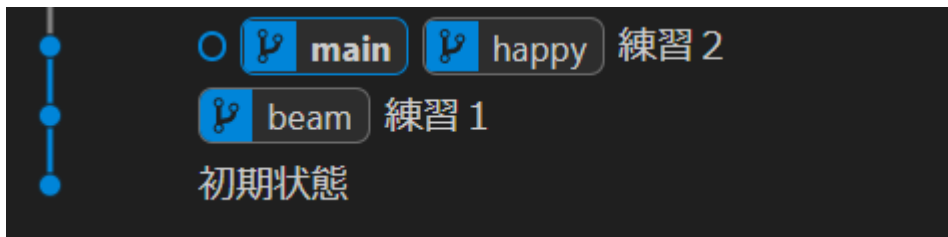
ブランチのマージ



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かん画像が切り替わる機能を実装せよ。

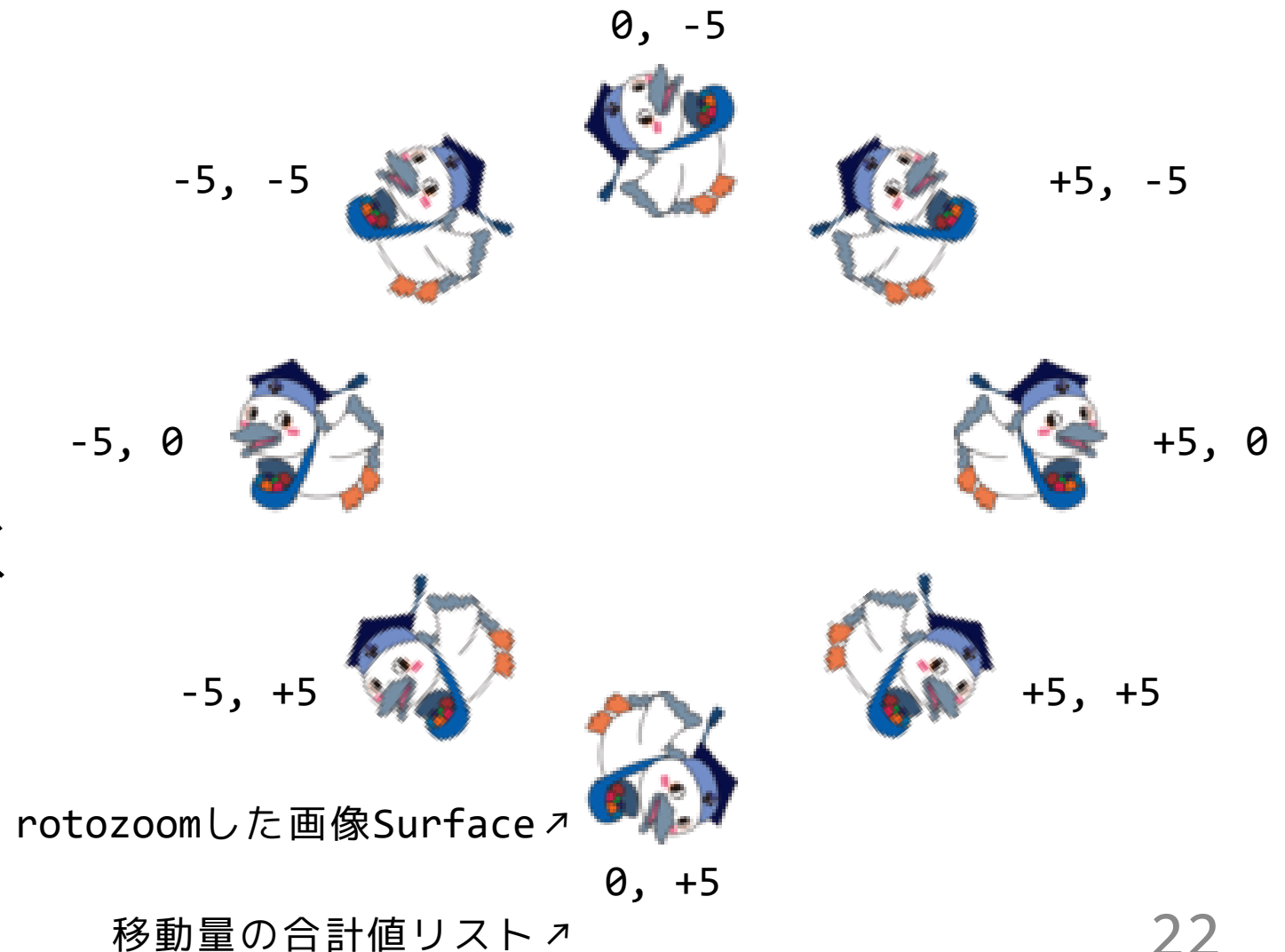
Birdクラス

- イニシャライザ：移動量の合計値リスト 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い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
の合計値リストをキー,
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）

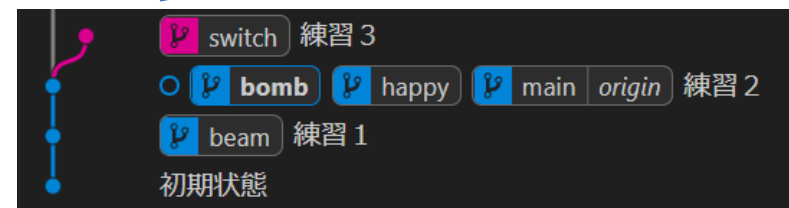
4. bombブランチにて，複数の爆弾を実装せよ．

- トップ：爆弾の数を表す定数NUM_OF_BOMBSを定義

Bombクラス

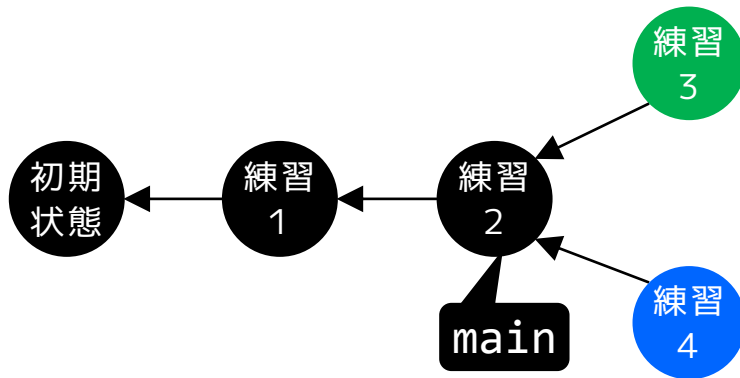
- イニシャライザ：色，移動方向，サイズをランダムに決定
 - 色と方向は，候補リストを作っておき，random.choiceするとよい
- イニシャライザ：上記でランダムに選択するため，引数は不要になる
- main関数：NUM_OF_BOMBS個のBombインスタンスを要素とするリストを生成
- main関数：リストの要素1つずつに対して，
こうかとん／ビームと衝突判定し，衝突した要素はNoneとする
- main関数：爆弾リストに対して，要素がNoneでないものだけのリストに更新

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

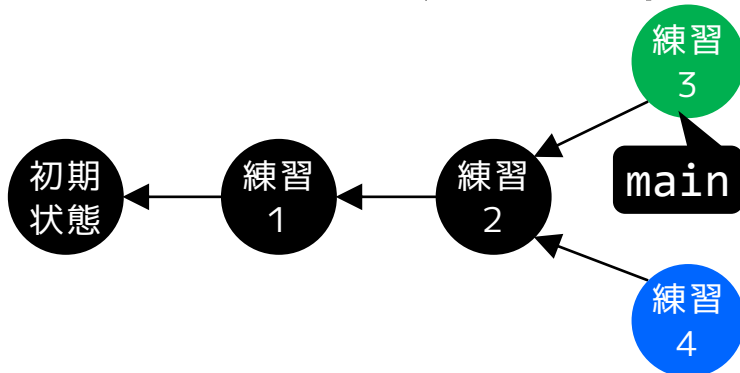


ブランチをマージしてみる (1/2)

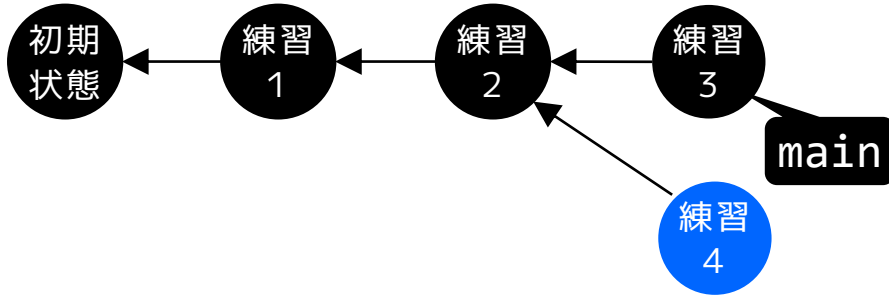
- main, switch, bombが独立している (同じ部分の修正がない) ので, コンフリクト (競合) は発生しないはず



- 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)

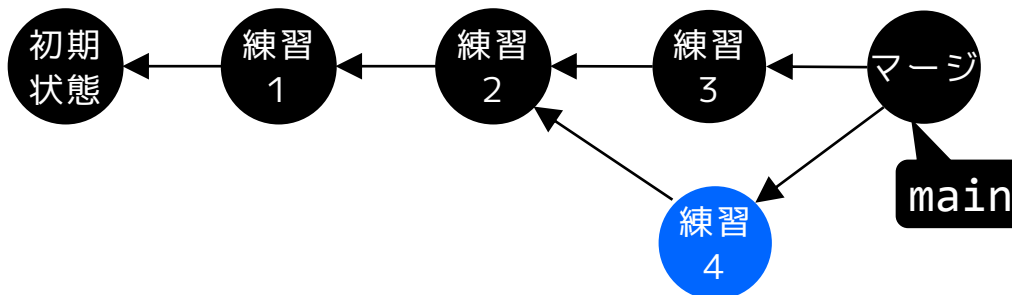


ブランチをマージしてみる (2/2)



- 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

```
admin@LAPTOP-63464884 MINGW64 ~/Desktop/ProjExD2023
$ 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
- 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。

```
96 <<<<<<< HEAD (現在の変更)
97     def get_rct(self) -> pg.Rect:
98         return self._rct
99
100 =====
101     def check_collide(self, bomb: "Bomb") -> bool:
102         """
103         こうかとんが爆弾と衝突したか判定する
104         引数1 bomb: Bombクラスのインスタンス
105         """
106
107         if self._rct.colliderect(bomb._rct):
108             return True
109         else:
110             return False
111 >>>>>> beam (入力側の変更)
```

← 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。

全てのコン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`

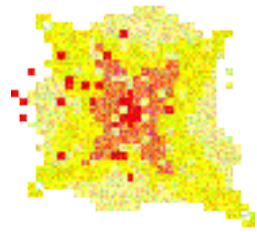
演習課題：「たたかえ！こうかとん」の改良

- 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わったら、
 - 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
- 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. 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
 - Explosionインスタンス用の空リストを作る
 - bombとbeamがぶつかったらExplosionインスタンスを生成し, リストにappend
 - `life`が0より大きいExplosionインスタンスだけのリストにする
 - updateメソッドを呼び出して爆発を描画
- 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：移動量の合計値リストが`[0,0]`でない時, `self.dire`を合計値リストの値で更新
- 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
 - こうかとの`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 = こうかとの中心横座標 + こうかとの横幅 × ビームの横速度 ÷ 5
 - ビームの中心縦座標 = こうかとの中心縦座標 + こうかとの高さ × ビームの縦速度 ÷ 5

※なんか知らんけど, 右上, 右下はビームが出ないので, 気にしないこと.

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創英角ポツ体", 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 メソッドを呼び出してスコアを描画

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空のリストを作る
- 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，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

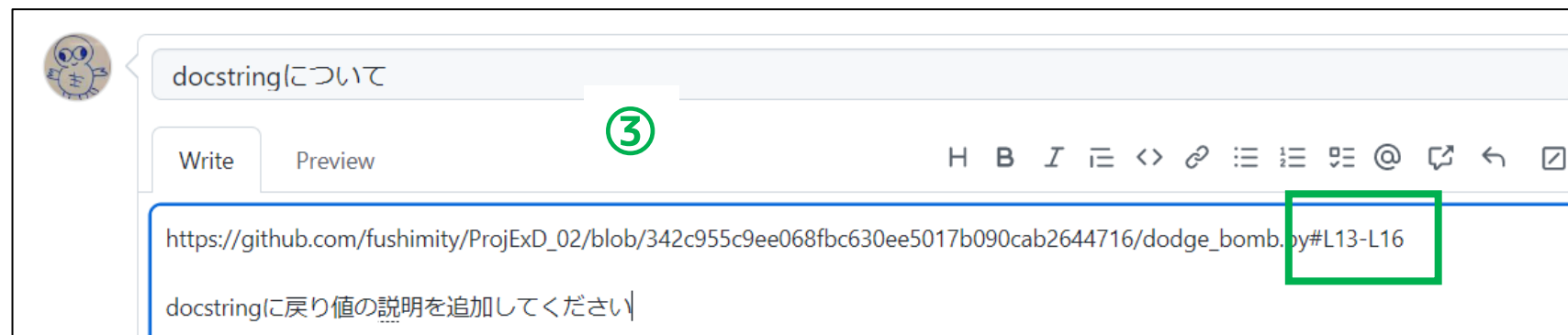
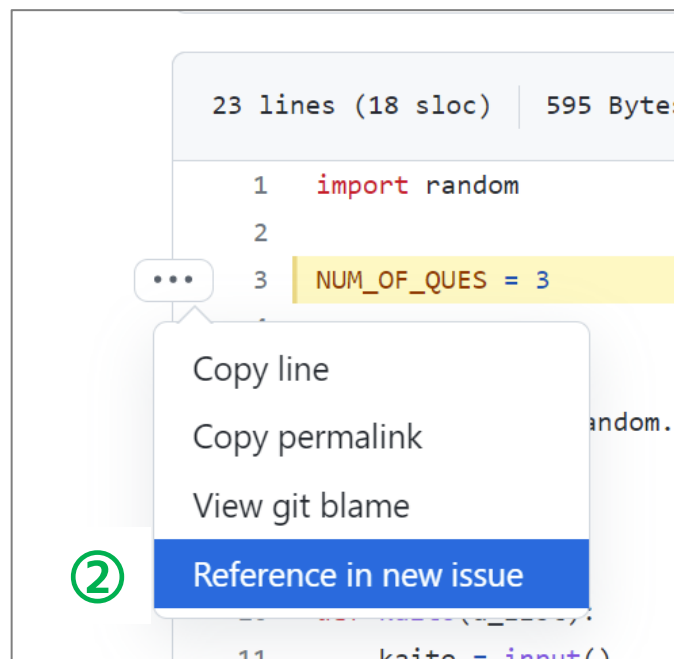
1. グループメンバー1人の公開リポジトリURLを入手する
2. 次ページを参照し、クローン+ローカルでゲームを実行してみる
3. コードを見て、修正すべき点をIssueにより送信する
 - 修正すべき点がない完璧なコードの場合、他のメンバーにレビューを依頼する
 - それでも修正すべき点がない場合、TASA、教員にレビューを依頼する
4. 自分の作業ディレクトリに戻る
5. Issueを受信したら、新たなブランチを切ってコードを修正する
 - 詳細は、5ページ後ろの「コード修正の手順」を参照のこと
6. 修正後は、マージせずに上記ブランチをプッシュする
7. mainブランチと上記ブランチの差分を表示させる

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



The screenshot shows the GitHub issue submission interface. At the top, there is a text input field containing "docstringについて". Below the input field are two tabs: "Write" and "Preview", with the "Preview" tab selected and marked with a green circle containing the number 4. The preview area displays the file path "ProjExD_02/dodge_bomb.py" and "Lines 13 to 16 in 342c955". The code snippet shows a docstring for a function that checks if an object is on the screen. Below the code, there is a text input field with the placeholder text "docstringに戻り値の説明を追加してください". At the bottom right, there is a green button labeled "Submit new issue", which is marked with a green circle containing the number 5. At the bottom left, there is a small icon and the text "Styling with Markdown is supported".

docstringについて

Write Preview ④

ProjExD_02/dodge_bomb.py
Lines 13 to 16 in 342c955

```
13      """
14      オブジェクトが画面内か画面外かを判定し、真理値タプルを返す
15      引数1 area: 画面SurfaceのRect
16      引数2 obj: オブジェクト（爆弾，こうかとん）SurfaceのRect
```

docstringに戻り値の説明を追加してください

⑤

Submit new issue

Styling with Markdown is supported

Issueが届く

- Issuesタブから，Issueコメントを読み，対応する

Issues 1 Pull requests Actions Projects Security Insights Settings

docstringについて #1 ← Issue番号

Open fushimity opened this issue now · 0 comments

fushimity commented now

ProjExD_02/dodge_bomb.py
Lines 13 to 16 in 342c955

```
13      ""  
14      オブジェクトが画面内か画面外かを判定し，真理値タプルを返す  
15      引数1 area: 画面SurfaceのRect  
16      引数2 obj: オブジェクト（爆弾，こうかとん）SurfaceのRect
```

docstringに戻り値の説明を追加してください

Write Preview

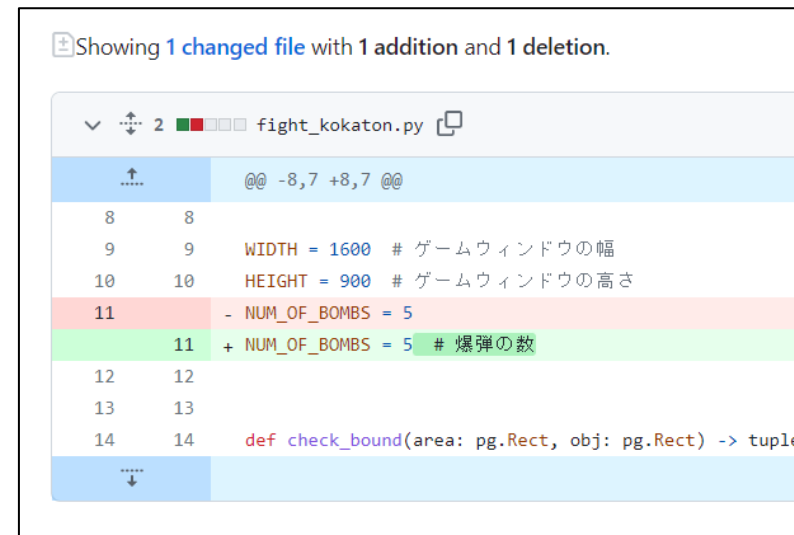
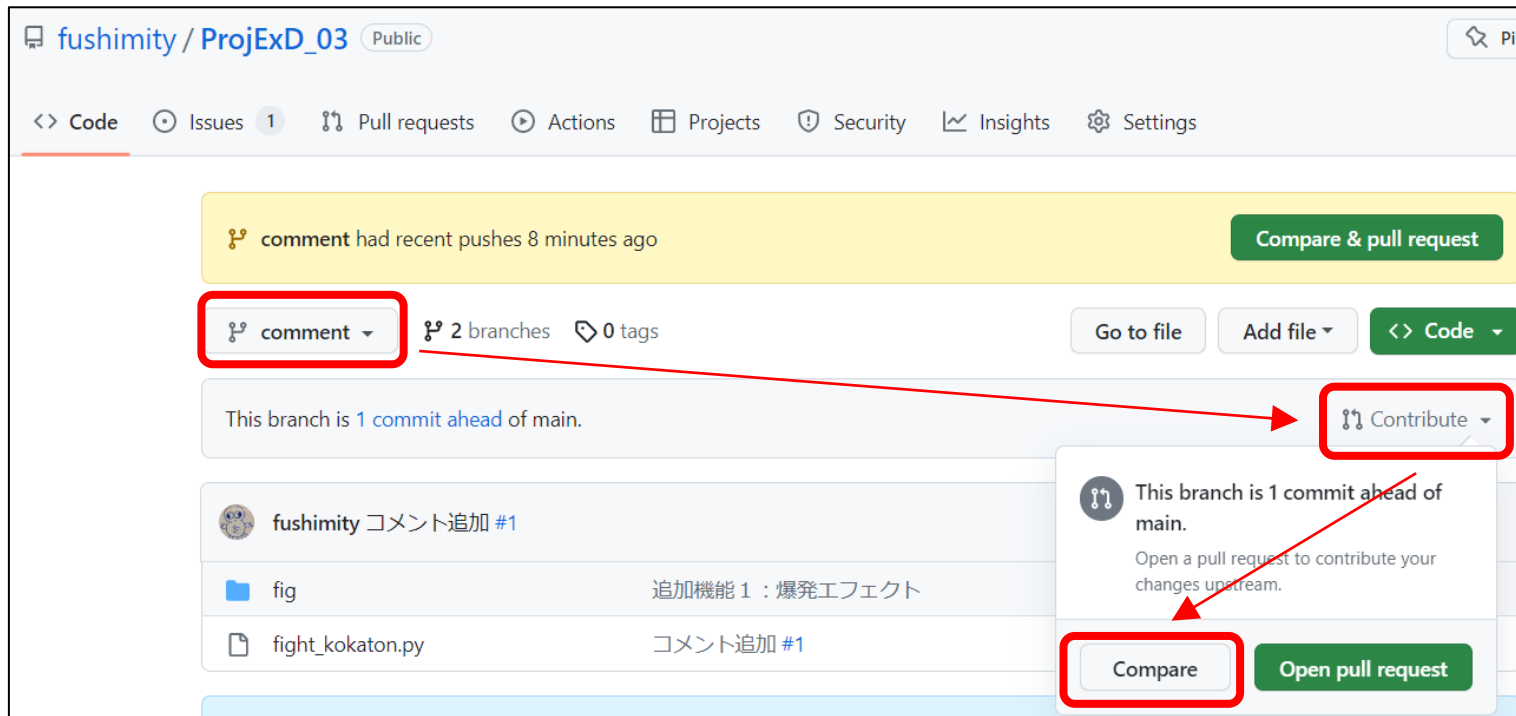
Leave a comment

コード修正の手順

0. 自分の作業ディレクトリに戻る：`cd ../ex03`
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
 - 例：`git commit -m "変数名統一 #2"`
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](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

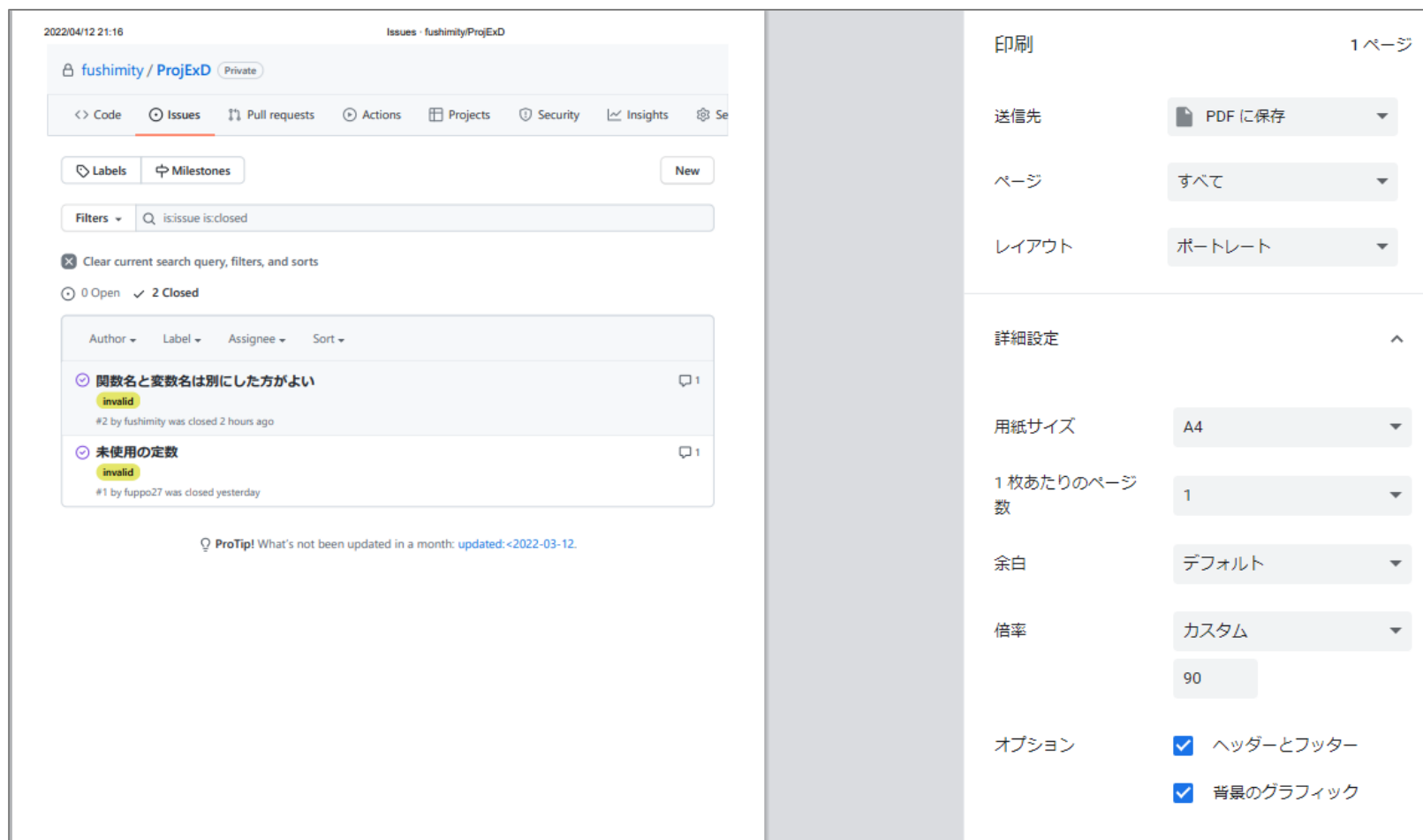
※スクショは認めません。

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