2024年12月10日(火)
3,4,5限 ＠研A302

プロジェクト演習 テーマD

第4回

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

次回までの宿題

- 次回以降：共同でオリジナルゲームを開発する
- 第5回：グループで共通基本機能＋個人で担当追加機能を実装する
- 第6回：グループで担当追加機能をマージする
- 第7回：発表準備＋成果発表会

- 次回までに，どのようなオリジナルゲームを開発するか
  おおよその案を考えてくること→ 第5回に開発の時間を多くとれる
- 次回の演習資料に細かい条件など書かれているので，参照すること

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）

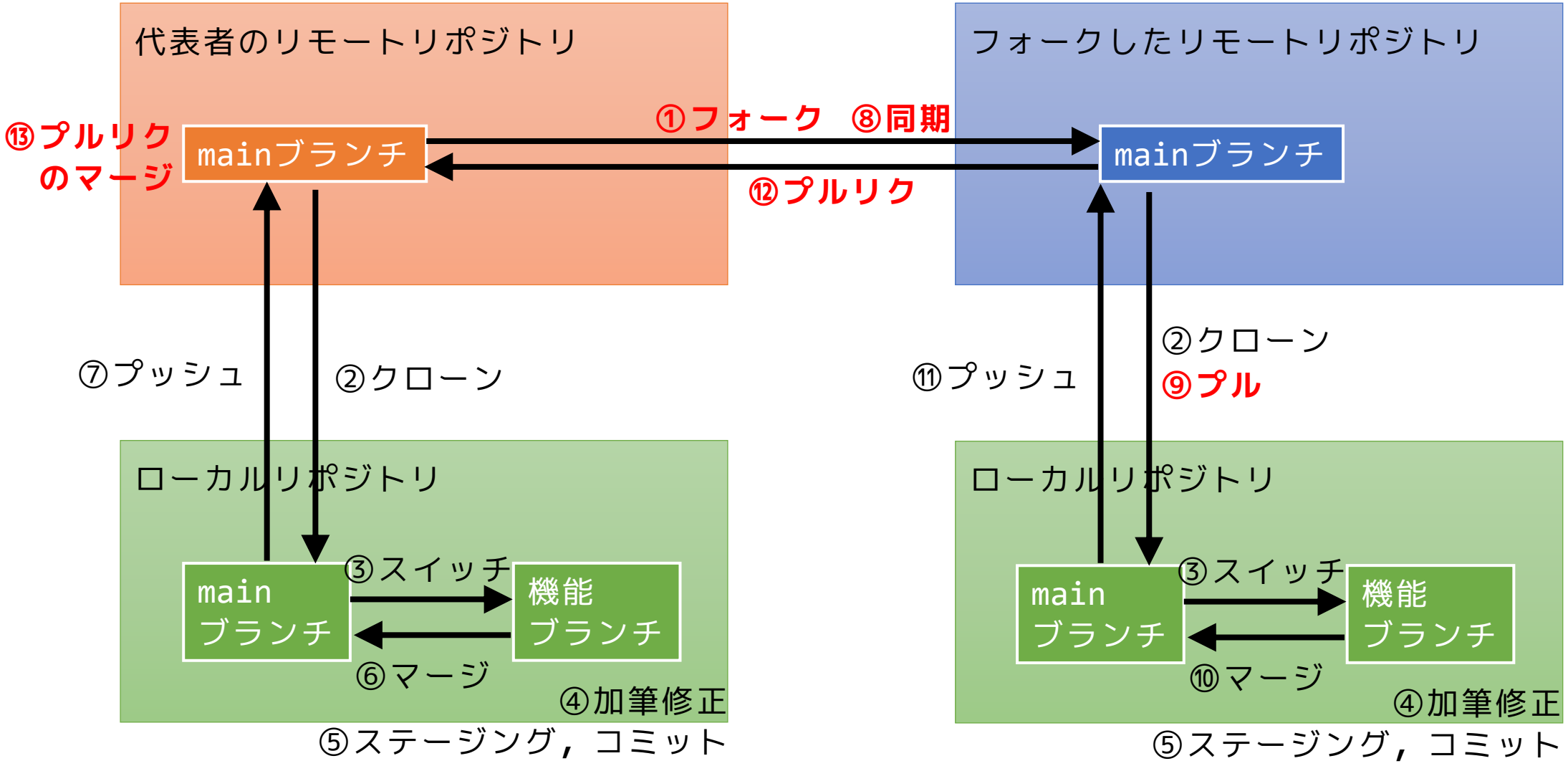
本日のお品書き

1. GitHubによる共同開発
   - フォーク
   - プル
   - フェッチ
   - プルリク

2. SpriteクラスとGroupクラス

目標：
- SpriteクラスとGroupクラスを活用し，Pygameらしいコードを実装できる
- GitHubを活用し，グループで共同開発できる

GitHubを利用した共同開発



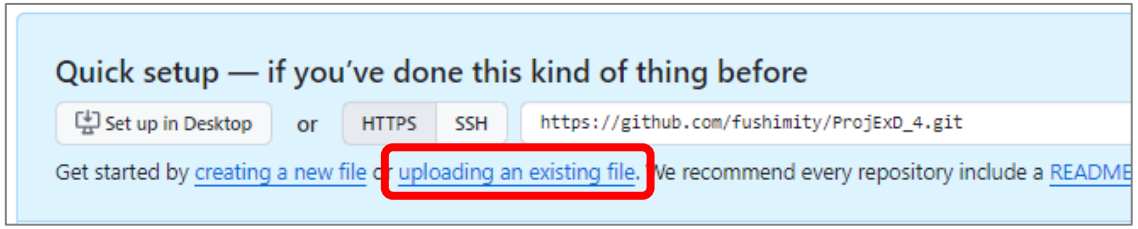
配布物の確認（代表者）

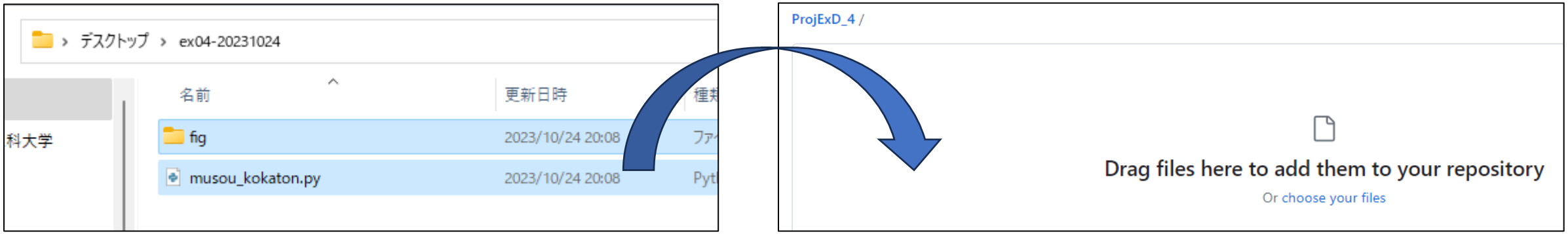
- ex4.zipをどこかにDLし，展開する　あとで削除するので，どこにDL，展開してもOK
- フォルダの中身をGitHubにアップロードする（次ページ参照）
  ※注意ex4フォルダ自体はアップしない

- アップロードするもの：
  - musou_kokaton.py　・・・真・こうかとん無双
  - 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.gif　・・・爆発画像
    - alien{1, …, 3}.png・・・敵画像

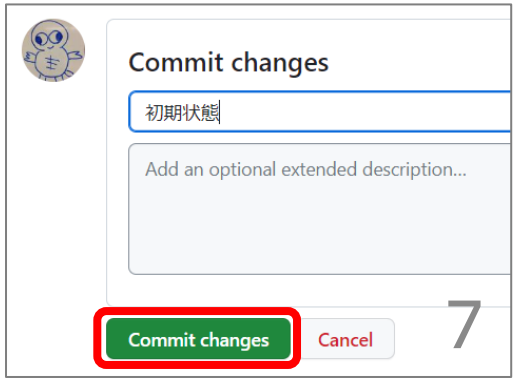
GitHubでの作業（代表者）

- GitHubに「ProjExD_4」という公開リポジトリを作成する
- 全ファイルをアップロードする





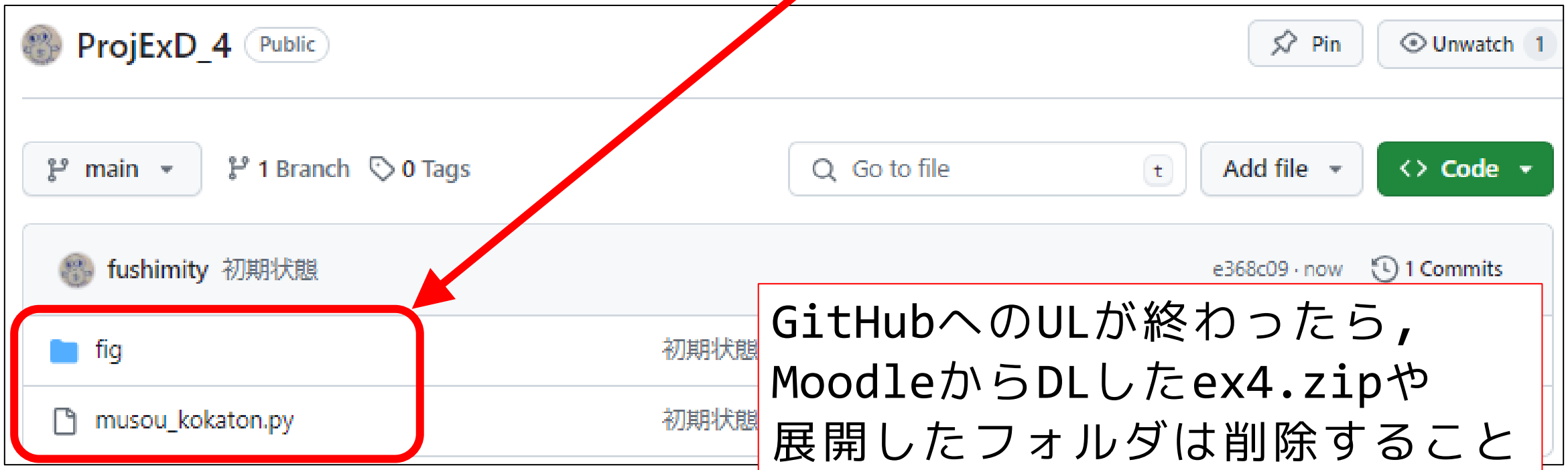
- 「初期状態」というコメントで`Commit changes`する



GitHubでの作業（代表者）

- 公開リポジトリの確認

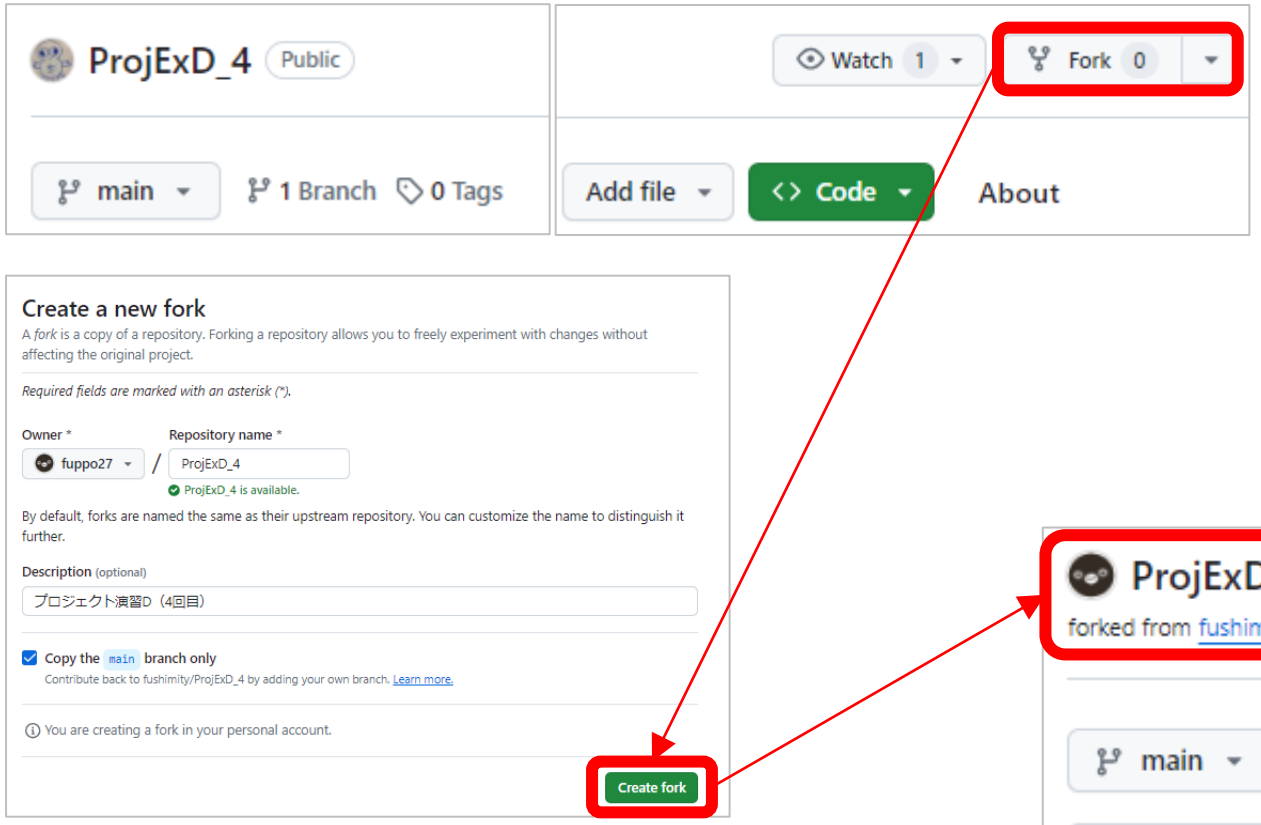
このようになっていない場合は，
イチからリポジトリを作り直すこと



GitHubへのULが終わったら，
MoodleからDLしたex4.zipや
展開したフォルダは削除すること

GitHubでの作業①（その他）

- 代表者のリポジトリを自身のリモートリポジトリとしてフォークする
  - 代表者のリポジトリで，「Fork」→「Create fork」

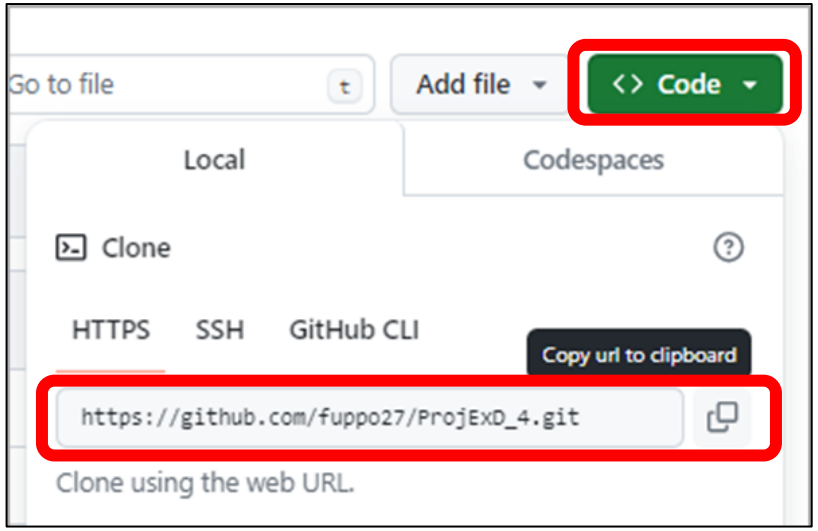


フォーク：
他のユーザが所有するリポジトリを
自分のリポジトリにコピーすること．
フォークしたリポジトリで行った変更を
プルリク機能によりマージを依頼できる

← 自身のリポジトリ名の下に
「forked from 代表者リポ名」
が表示される

gitでの作業②（全員）

- ProjExDフォルダにて，自身の公開リポジトリをクローンする：
  `git clone 自身の公開リポジトリのURL ex4`

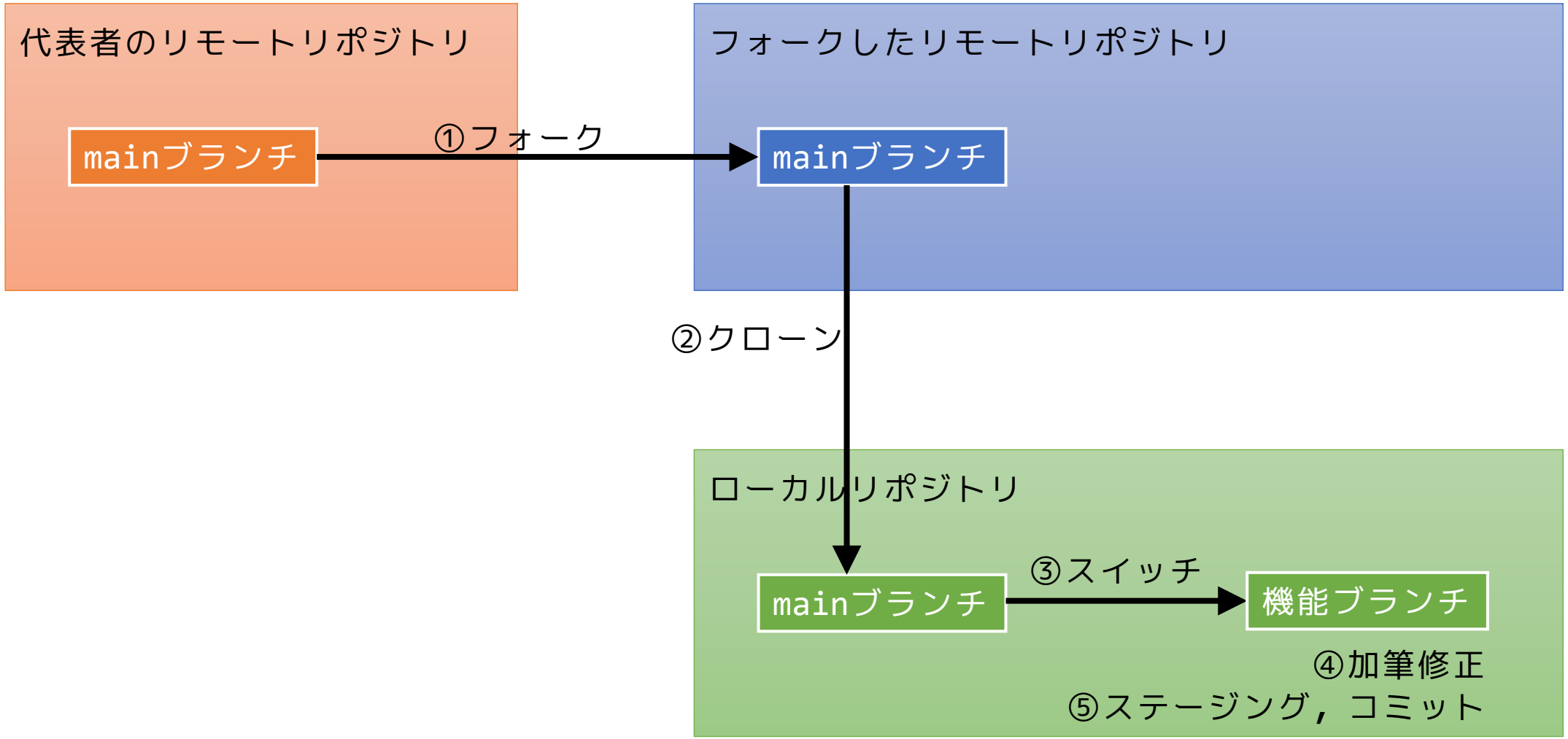


- 作業フォルダex4に移動する：`cd ex4`
- リモートリポジトリのURLが自身のリポジトリであるか確認する：
  `git remote -v`

```
Admin@MSI MINGW64 ~/Desktop/ProjExD/ex4 (main)
$ git remote -v
origin  https://github.com/fushimity/ProjExD_4.git (fetch)
origin  https://github.com/fushimity/ProjExD_4.git (push)
```

作業の流れ（イメージ図）

- その他メンバーの視点



「真・こうかとん無双」で遊んでみよう

musou_kokaton.pyの概要

- Spriteクラスのサブクラス
  - Birdクラス：主人公キャラクター → Spriteクラスを継承
  - Bombクラス：敵機が放つ爆弾
  - Beamクラス：こうかとん砲
  - Explosionクラス：爆発エフェクト
  - Enemyクラス：敵機

  （Bomb, Beam, Explosion, Enemy：Spriteクラスを継承＋Groupオブジェクトに追加）
- Spriteクラスのサブクラスの共通メソッド
  - イニシャライザ：初期化
  - updateメソッド：座標更新，状態更新，発動時間減算
    - ※Groupオブジェクトに追加しないBirdクラスの場合，blitも行う
    - ※Groupオブジェクトに追加するクラスの場合，blitは行わない（drawで一括描画）
- Scoreクラス：打ち落とした爆弾，敵機に基づくスコア

クラス設定の詳細（1/3）

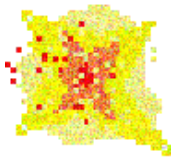
- Birdクラス

  - 矢印キーで操作する
  - スペースキーでビーム発射する
  - 爆弾を打ち落とすとスコアが1上がる
  - 敵機を撃ち落とすとスコアが10上がる＋喜びエフェクト
  - 爆弾にあたるとゲームオーバー＋悲しみエフェクト
  - 飛ぶ方向によって画像が変わる

- Bombクラス
  - 敵機からこうかとんに向かって直線的に動く
  - ランダムに決まる色とサイズに意味はない
  - ビームにあたるとGroupオブジェクトから削除される
  - 画面外に出るとGroupオブジェクトから削除される

クラス設定の詳細（2/3）

- Beamクラス 
  - スペースキー押下により，こうかとんから発射される
  - 複数回の押下により，複数のビームが発射される
  - 爆弾または敵機にあたるとスコアがあがる＋爆発エフェクト
  - 爆弾または敵機にあたるとGroupオブジェクトから削除される
  - 画面外に出るとGroupオブジェクトから削除される

- Explosionクラス 
  - ビームが爆弾または敵機にあたると，指定された爆発時間の間表示される
  - 反転した画像と切り替えることで，爆発エフェクトを演出する
  - 爆発時間を過ぎるとGroupオブジェクトから削除される

クラス設定の詳細（3/3）

- Enemyクラス
  - 200フレーム（4秒）に1体，画面上部から出現し，画面中央部まで降下する
  - 降下が終了したら，50以上300以下の乱数で決められたインターバルで爆弾を投下する
  - ビームに当たるとGroupオブジェクトから削除される

- Scoreクラス
  - 画面左下にスコアを表示する
  - 爆弾1つで1点，敵機1体で10点アップする
  - 追加機能の使用可能条件／消費スコアを規定する予定

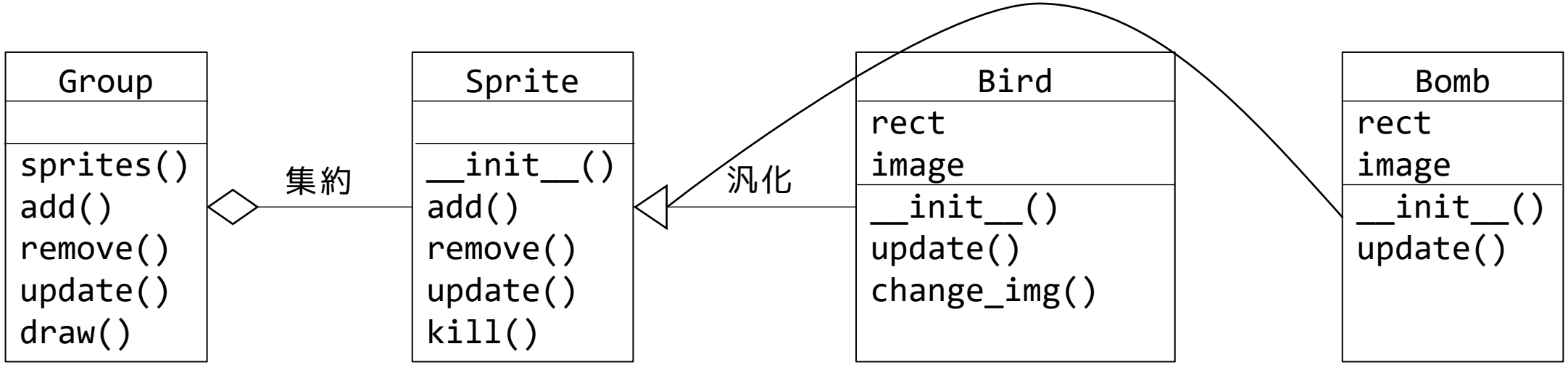
fight_kokaton.pyからの変更点

- クラスの属性（変数）名　　∵ Spriteクラスの制約
  - Surface：xx_img → self.image
  - Rect：xx_rct → self.rect

- こうかとんと爆弾，ビームの移動スピード
  - こうかとん：10
  - ビーム：10
  - 爆弾：6
  - 敵機：6

spriteモジュール

- コンピュータ上で動く図形を表現する際に，動かす図形と固定された背景とを別に作成し，ハードウェア上で合成することによって表示を高速化する手法＝スプライト処理
- SpriteクラスとGroupクラスが用意されている
- Spriteクラスを継承したオリジナルクラス（BirdやBomb）を作成し，そのインスタンス群を保持するコンテナとしてGroupクラスを用いる



Spriteクラス

- スプライト処理をするための基底クラス（継承される前提）である
- Spriteクラスを継承することで，簡単にスプライト処理を実装できる
- Spriteクラスを継承する際には，image属性，rect属性，update()メソッドを実装する必要がある

- image属性：画面に表示させるSurfaceオブジェクト
- rect属性：上記Surfaceオブジェクトに対応するRectオブジェクト

```
class Bomb(pg.sprite.Sprite):
    def __init__(self, emy: "Enemy", bird: Bird):
        super().__init__()

        self.image = pg.Surface((2*rad, 2*rad))
        self.rect = self.image.get_rect()
```

・・・Spriteクラスの継承

・・・Spriteクラスのinit

・・・画像Surface

・・・のRectオブジェクト

Groupクラス

- Groupオブジェクトに Spriteオブジェクトを追加することで，
  複数のSpriteオブジェクトを効率的に扱うことができる
- **Groupオブジェクトに追加する際には，Spriteクラスのイニシャライザ**
  __init__()を呼び出す必要がある

- add(Spriteオブジェクト)メソッド：
  Spriteオブジェクトを Groupオブジェクトに追加する
- update()メソッド：Groupオブジェクトに含まれる
  Spriteオブジェクトのupdateメソッドを呼び出す
- draw(画面Surface)メソッド：Groupオブジェクトに含まれる
  Spriteオブジェクトを一括して画面Surfaceに描画する

```
bombs = pg.sprite.Group()
bombs.add(Bomb(emy, bird))
bombs.update()
bombs.draw(screen)
```

・・・空のGroupオブジェクトに
・・・Bombオブジェクトを追加する
・・・全Bombオブジェクトのupdateメソッドを呼び出す
・・・全Bombオブジェクトを描画する

Groupに追加されるSpriteでは，

- Groupオブジェクトに追加されるSpriteオブジェクトのupdateメソッドでは，Groupオブジェクトからの削除条件を記述しておくことが多い

- kill()メソッド：
  Groupオブジェクトから当該オブジェクトを削除する

```
class Bomb(pg.sprite.Sprite):
    def update(self):
        self.rect.move_ip(略, 略)
        if check_bound(self.rect) != (True, True):
            self.kill()
```

・・・爆弾が画面外に出たら
kill()メソッドを呼び出して
爆弾をグループから削除する

SpriteとGroupを使う流れ

【Bombクラスの例】

1. Spriteクラスを継承したクラスを定義　← main関数の外
   - Bomb(pg.sprite.Sprite)
2. Groupオブジェクトの生成（インスタンス生成）　← main関数の中・whileループの前
   - bombs = pg.sprite.Group()
3. インスタンスの生成
   - bomb = Bomb()
4. インスタンスをGroupオブジェクトに追加
   - bombs.add(bomb)

（3.〜4.：main関数の中 爆弾生成時）

5. Groupオブジェクトに含まれるインスタンスを**すべて**更新
   - bombs.update()
6. Groupオブジェクトに含まれるインスタンスを**すべて**描画
   - bombs.draw(screen)

（5.〜6.：main関数の中 whileループの中）

※第3，4引数の真理値は重要

spriteモジュールの衝突判定関数

- spritecollide()関数：Sprite単体とGroupの衝突判定
  - 引数１：Spriteオブジェクト
  - 引数２：Groupオブジェクト
  - 引数３：衝突したオブジェクトをGroupから削除するか否かの真理値
  - 戻り値：衝突したオブジェクトの**リスト**

```
for bomb in pg.sprite.spritecollide(bird, bombs, True): return  # ゲームオーバー
```

- groupcollide()関数：Group同士の衝突判定
  - 引数１：Groupオブジェクト1
  - 引数２：Groupオブジェクト2
  - 引数３：衝突したオブジェクトをGroup1から削除するか否かの真理値
  - 引数４：衝突したオブジェクトをGroup2から削除するか否かの真理値
  - 戻り値：衝突したオブジェクトの**辞書**（キー：Group1のオブジェクト／値：キーのオブジェクトと衝突したGroup2のオブジェクト**リスト**）

```
for bomb in pg.sprite.groupcollide(bombs, beams, True, True).keys():
    exps.add(Explosion(bomb, 50))  # 爆発エフェクト
```

編集内容の取り消し

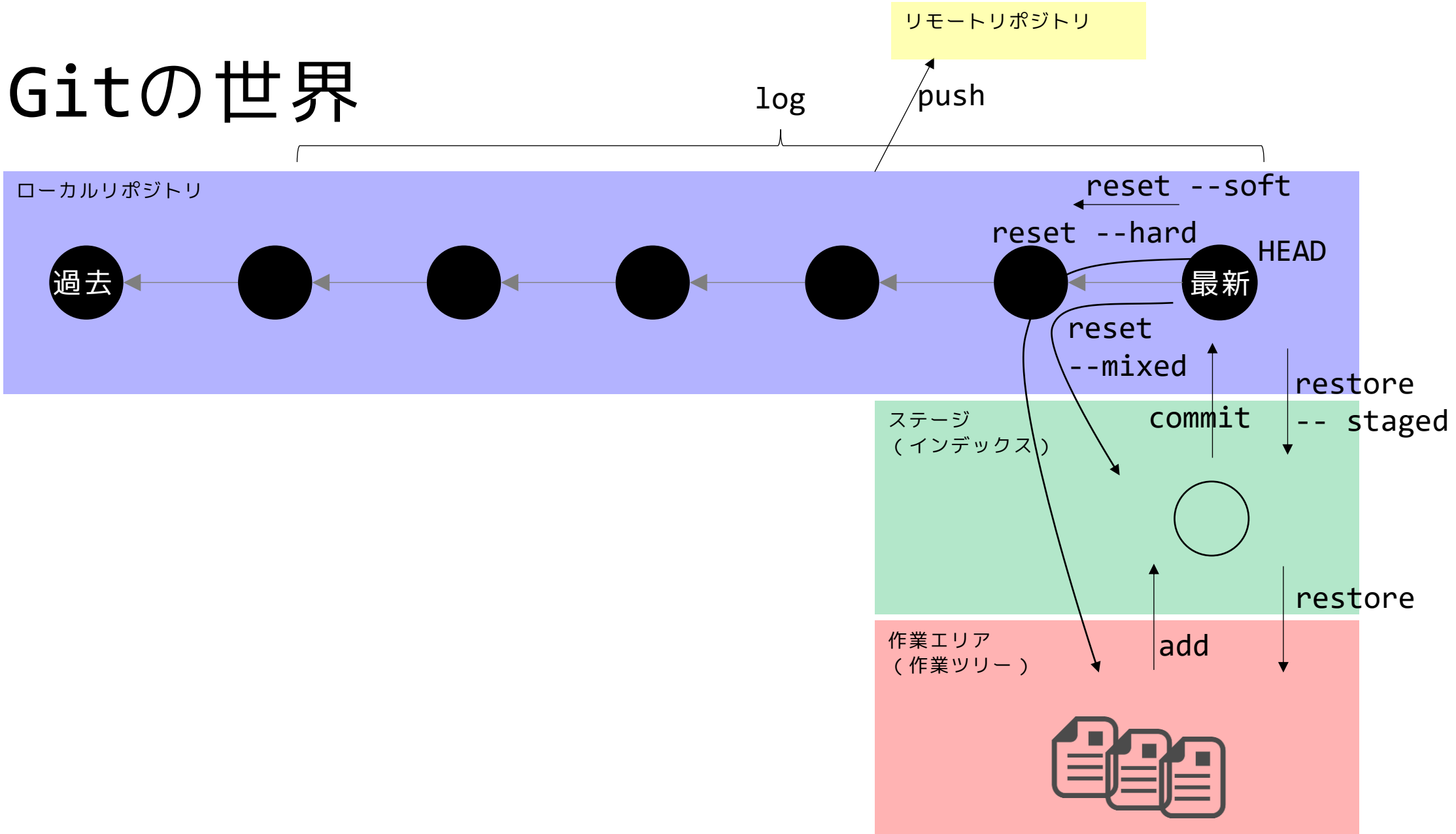
いろいろいじった方は，演習前に作業エリアを初期状態に戻そう

- コミットしてない人 → 作業エリアをステージに合わせる：
  `git restore musou_kokaton.py`

- コミットした人 → 打ち消しコミット【推奨】：
  `git revert HEAD`
  ←最新コミットであるHEADを打ち消すコミットを実行する例

- コミットした人 → 作業エリアをあるコミットに合わせる：
  `git reset --hard HEAD^`
  ←「^」X個で，HEADからX個前のコミットを表す

Gitの世界


リモートリポジトリ
log
push
ローカルリポジトリ
reset --soft
reset --hard
HEAD
過去
最新
reset
--mixed
restore
-- staged
commit
ステージ
（インデックス）
restore
add
作業エリア
（作業ツリー）
status

演習課題：「真・こうかとん無双」の改良

- 以下の６つの機能を，機能ごとのブランチにて実装せよ
- グループメンバー1人1機能を実装し，後ほどマージする
- 担当以外の追加機能も実装しても問題ない（後でマージするかは別）
- 人数が少ないグループは，実装しない機能があっても問題ない
- 追加機能（おおよその難易度順）
  1. 高速化・・・・高速化する
  2. 重力場・・・・重力場の領域内に存在する敵機と爆弾をせん滅させる
  3. 電磁パルス・・パルス発動時に存在する敵機と爆弾を無効化する
  4. 無敵状態・・・無敵状態時は爆弾に当たっても死なない
  5. 防御壁・・・・敵の放つ爆弾から防御壁で身を守る
  6. 弾幕・・・・・一度に多方向のビームを放ち，敵機や爆弾を破壊する

gitでの作業③（全員）

- 機能ごとにブランチを作り，切り替える：
  追加機能１の場合は「C0A23XXX/feature1」というブランチ名にする
  `git switch -c C0A23XXX/feature1`

学籍番号：自分のもの／半角・大文字

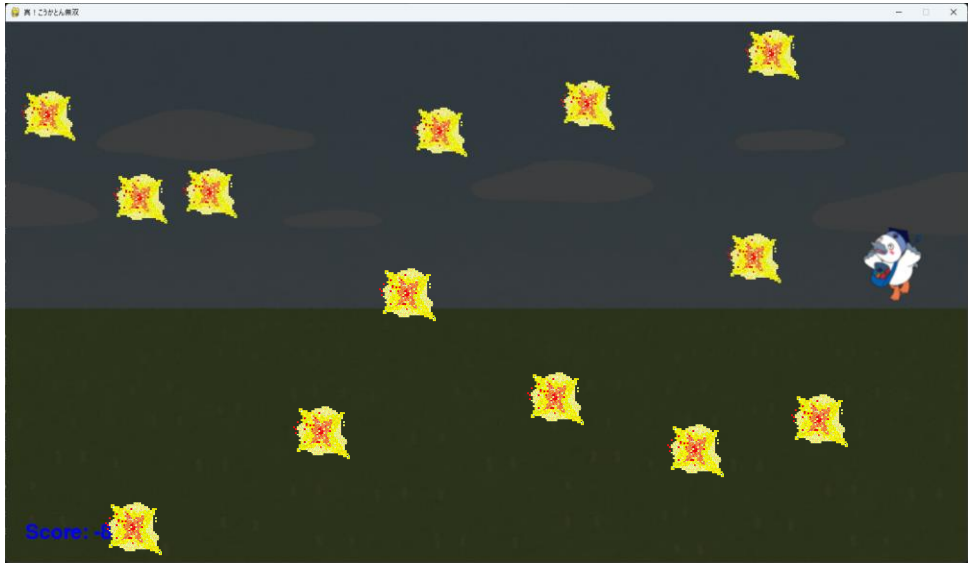
- 追加機能実装後，ステージング，コミットすること
- mainブランチなどにマージしないこと
- 担当以外の機能を実装する時は，mainブランチに戻ってから，
  新たなブランチを切って作業すること

追加機能１：高速化

- 設定：こうかとんを高速化させる
  - デフォルト`speed`：10／高速化時`speed`：20
  - 発動条件：左Shiftキー押下しながら矢印キーで移動
  - 消費スコア：なし
- 実装例
  - 簡単すぎるのでNoヒント・No解説.

発動条件の押下キーは自由だが
必ずグループで相談すること

追加機能２：重力場



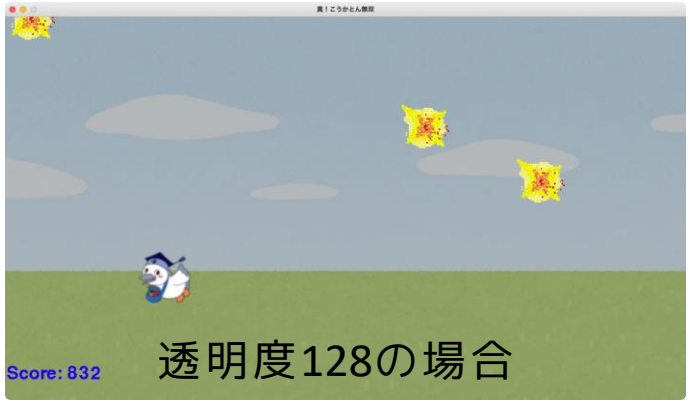
- 設定：画面全体を覆う重力場を発生させる
  - 重力場：画面全体に透明度のある黒い矩形
  - 発動時間：400フレーム
  - 効果：重力球の範囲内の爆弾を打ち落とす
  - 発動条件：リターンキー押下，かつ，スコアが200より大
  - 消費スコア：200
- 実装例
  - Gravityクラスのイニシャライザの引数を，発動時間の変数`life`とする
  - イニシャライザで重力場のSurfaceと対応するRectを生成する
  - updateメソッドで`life`を1減算し，0未満になったらkillする
  - インスタンスをGravityグループに追加する
  - 爆弾，敵機との衝突判定で，範囲内の爆弾，敵機を処理（爆発エフェクトと削除）する

発動条件の押下キーは自由だが
必ずグループで相談すること

半透明な矩形の描き方

- 手順１：空のSurfaceインスタンスを生成する
  - `self.image = pg.Surface((横幅, 高さ))`
- 手順２：上記Surfaceにrectをdrawする
  - `pg.draw.rect(self.image, 色タプル,`
    `(左上横座標, 左上縦座標, 右下横座標, 右下縦座標))`
  - ここまでで，画面全体が真っ黒な矩形で覆われればOK
- 手順３：上記Surfaceに透明度を設定する
  - `self.img.set_alpha(透明度)`

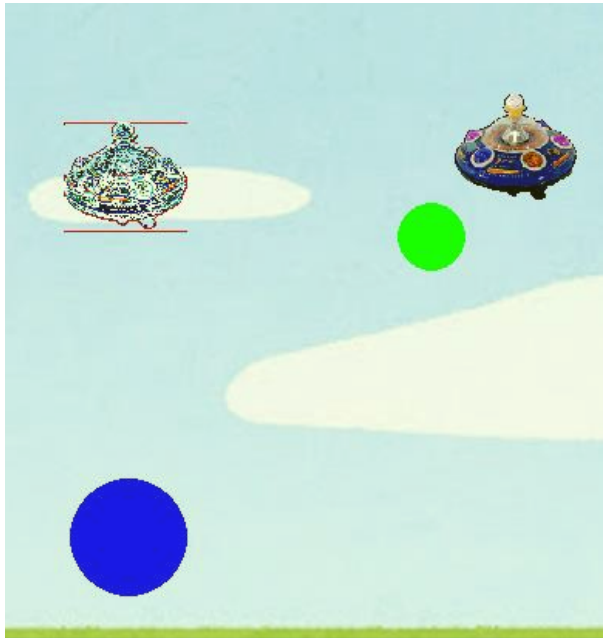
透明度0の場合



透明度128の場合

透明度255の場合

追加機能３：電磁パルス（EMP）



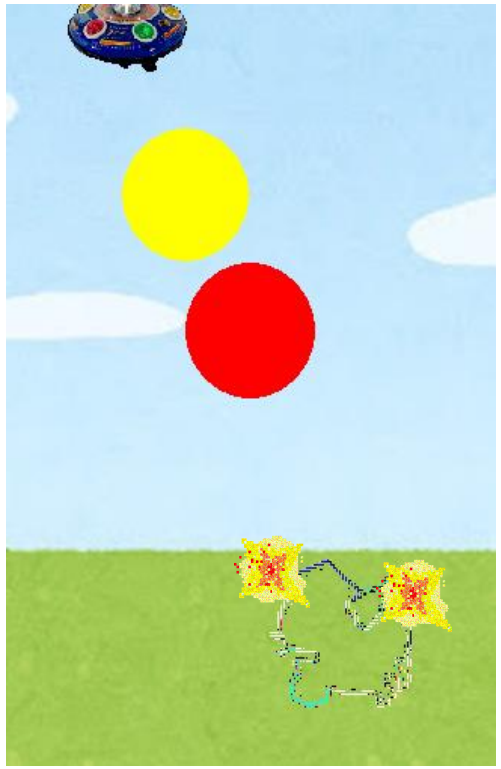
- 設定：発動時に存在する敵機と爆弾を無効化する
  - 無効化
    - 敵機：爆弾投下できなくなる（見た目はラプラシアンフィルタ）
    - 爆弾：動きが鈍くなる／起爆しなくなる
  - 見た目：画面全体に透明度のある黄色の矩形を0.05秒表示
  - 発動条件：「e」キー押下，かつ，スコアが20より大
  - 消費スコア：20
- 実装例
  - EMPクラスのイニシャライザにEnemyインスタンスのグループ，Bombインスタンスのグループ，画面Surfaceを渡す
  - Enemyインスタンスを無効化する
    - Enemyインスタンスのintervalを無限大infにする
    - Enemyインスタンスのimageにラプラシアンフィルタを掛ける

```
Enemyインスタンスのimage = pg.transform.laplacian(Enemyインスタンスのimage)
```

  - Bombインスタンスを無効化する：speedを半減する／stateをinactiveにする

発動条件の押下キーは自由だが
必ずグループで相談すること

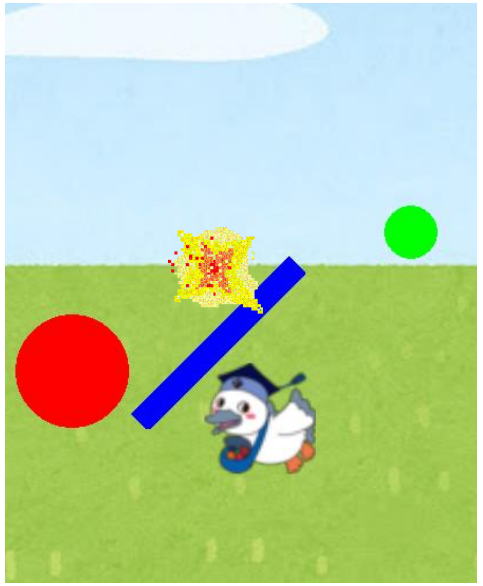
追加機能４：無敵状態



- 設定：爆弾に当たっても死なない
  - 発動中の画像例：`self.image = pg.transform.laplacian(self.image)`
  - 発動時間：500フレーム
  - 発動条件：右Shiftキー押下，かつ，スコアが100より大
  - 消費スコア：100
- 実装例
  - Birdクラスに，状態の変数`state`と発動時間の変数`hyper_life`を追加する
  - 発動条件が満たされたら，`state="hyper"`，`hyper_life=500`とする
  - Birdクラスのupdateメソッドで，
    - 画像`image`を変換したものに切り替える
    - `hyper_life`を1減算し，0未満になったら`state="normal"`とする
  - こうかとんと爆弾の衝突判定で，`state="hyper"`ならゲームオーバーにならずに爆弾を爆発させ，スコアを1アップさせる

発動条件の押下キーは自由だが
必ずグループで相談すること

追加機能５：防御壁



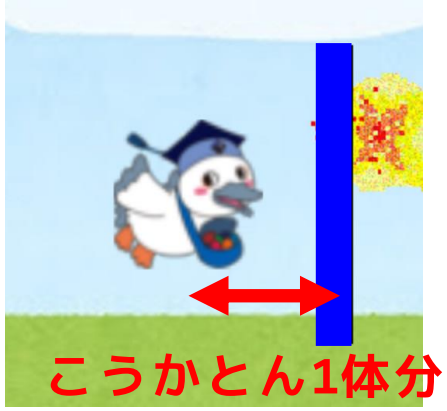
- 設定：こうかとんの前に防御壁を出現させ，着弾を防ぐ
  - 防御壁：青い矩形（横幅：20／高さ：こうかとんの身長の2倍）
  - 発動時間：400フレーム
  - 発動条件：「s」キー押下，かつ，スコアが50より大，
    かつ，防御壁が他に存在しない（一度に1壁のみ）
  - 消費スコア：50

発動条件の押下キーは自由だが
必ずグループで相談すること

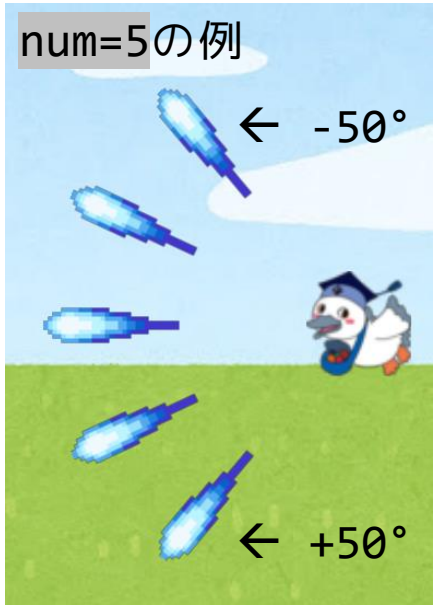
- 実装例
  - Shieldクラスのイニシャライザの引数を，こうかとん`bird`と発動時間`life`とする
  - Shieldクラスの`update`メソッドで`life`を1減算し，0未満になったら`kill`する
  - インスタンスをShieldグループに追加する（1つしか存在できないけど）
  - `groupcollide`で衝突した爆弾を検出し，破壊する

防御壁を向いている方向に表示させる方法

- 手順１：幅，高さを指定した空のSurfaceを生成する
- 手順２：上記Surfaceにrectをdrawするpgdrawrectselfimage色タプル左上横座標左上縦座標右下横座標右下縦座標
- 手順３：こうかとんの向きを取得するvxvy こうかとんdire
- 手順４：角度を求める角度 degreesatan2-vyvx
- 手順５：上記Surfaceを回転させる
- 手順６：向いている方向に，こうかとんの中心からこうかとん１体分ずらした位置に配置する



追加機能６：弾幕



- 設定：一度に複数方向へビームを発射する
  - 発動条件：左Shiftキーを押下しながらスペースキー
  - 消費スコア：なし
- 実装例
  - Beamクラスのイニシャライザの引数に回転角度angle0（デフォルトで0）を追加し，ビームの回転角度に加算する
  - NeoBeamクラスのイニシャライザの引数を，こうかとんbirdとビーム数numとする
  - NeoBeamクラスのgen_beamsメソッドで，
    -50°～+51°の角度の範囲で指定ビーム数の分だけBeamインスタンスを生成し，リストにappendする → リストを返す
  - 発動条件が満たされたら，NeoBeamクラスのイニシャライザにこうかとんとビーム数を渡し，戻り値のリストをBeamグループに追加する

発動条件の押下キーは自由だが
必ずグループで相談すること

異なる角度のビームを生成する方法

- 手順１：複数ビームの角度を`range(-50, +51, ステップ)`で求める
  - 例１：ビーム数が3の場合，range(-50, +51, 50)とすれば
    [-50, 0, +50] が得られる
  - 例２：ビーム数が5の場合，range(-50, +51, 25)とすれば
    [-50, -25, 0, +25, +50] が得られる
  - このようになるステップの求め方を考えよう

- 手順２：求めた角度をBeamクラスの引数に渡し，Beamインスタンスを生成する
- 手順３：上記のBeamインスタンスのリストを返す

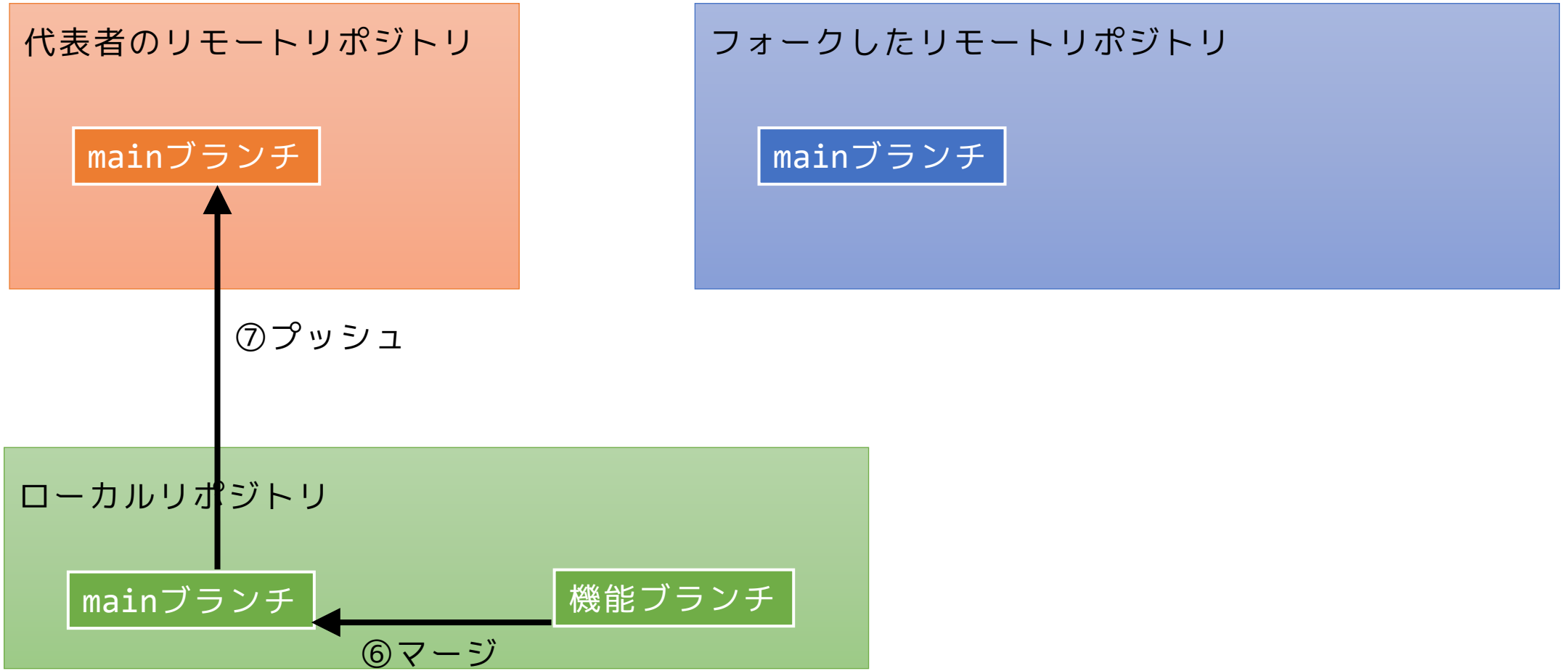
5限：プルリクによる機能ブランチのマージ

（全員）⑤実装完了したら，担当の追加機能ブランチにて

- ステージング：`git add musou_kokaton.py`
- コミット：`git commit -m "追加機能１実装完了“`
- スイッチ：`git switch main`

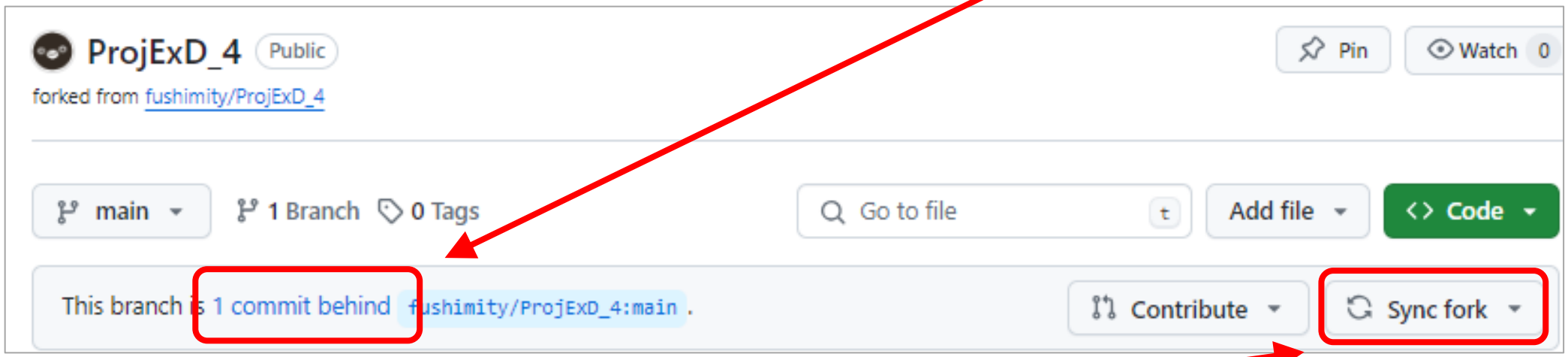
gitでの作業⑥⑦（代表者）

- ⑥機能ブランチをマージする：`git merge C0A23XXX/featureX`
- ⑦mainブランチをプッシュする：`git push origin main`



GitHubでの作業⑧（その他全員）

- リモートリポジトリのmainブランチを同期する
  - その他メンバーのリモートリポジトリは「1 commit behind」となっている



  - Sync fork → Update branch で代表者のmainブランチと同期させる

gitでの作業⑨⑩（その他一人目）

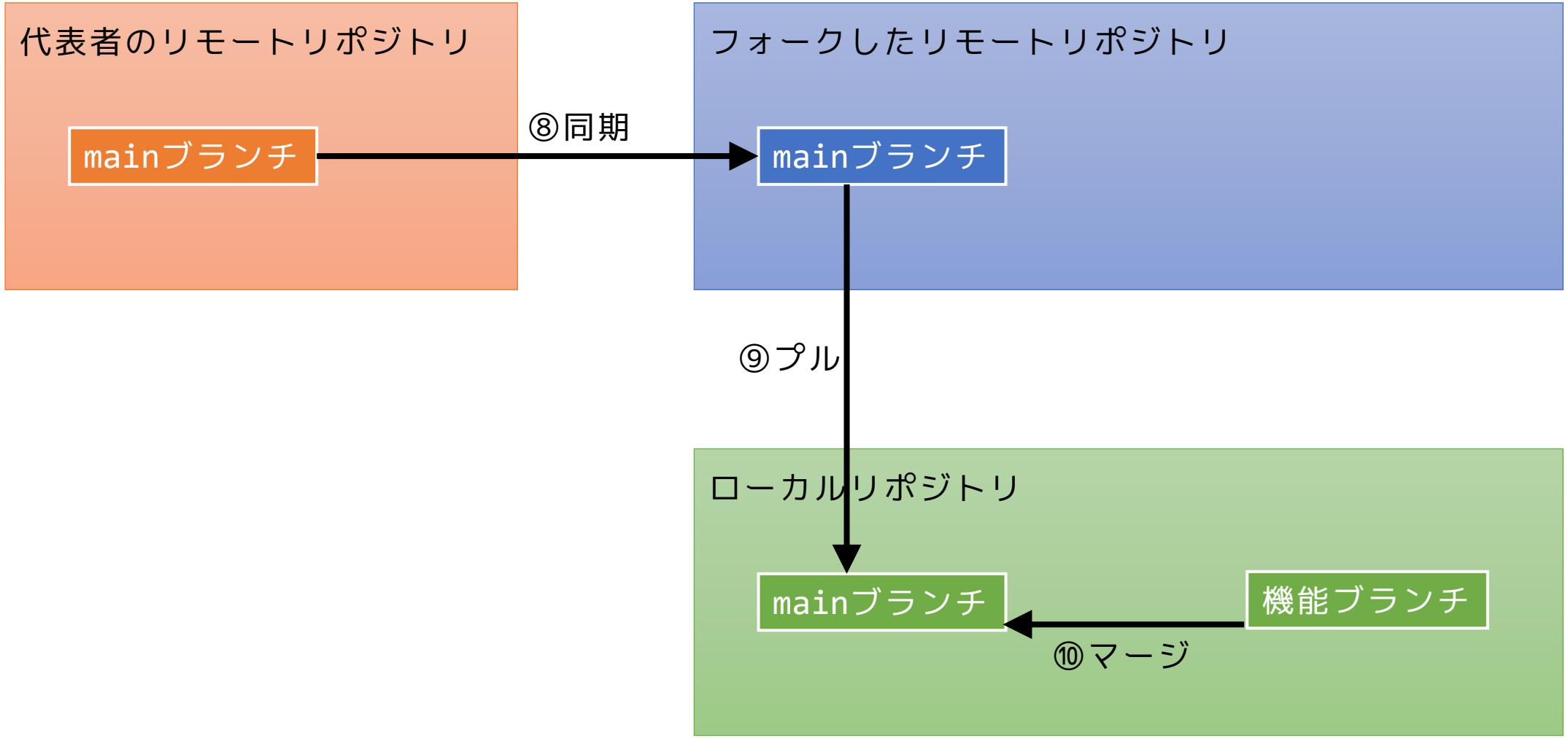
- ⑨自身のリモートリポジトリのmainブランチをローカルにプルする：
  `git pull origin main`

> **プル：**
> リモートリポジトリのコミットをダウンロード（**フェッチ**）し，
> それをローカルリポジトリに統合（**マージ**）すること

- 代表者が実装した追加機能が反映されている確認する
- ⑩追加機能ブランチをmainブランチにマージする：
  `git merge C0A23XXX/featureX`
- ※コンフリクトしたら，次々ページを参照して解消しよう
- 代表者と自身の実装した追加機能が反映されているか確認する

作業の流れ（イメージ図）

- その他メンバーの視点

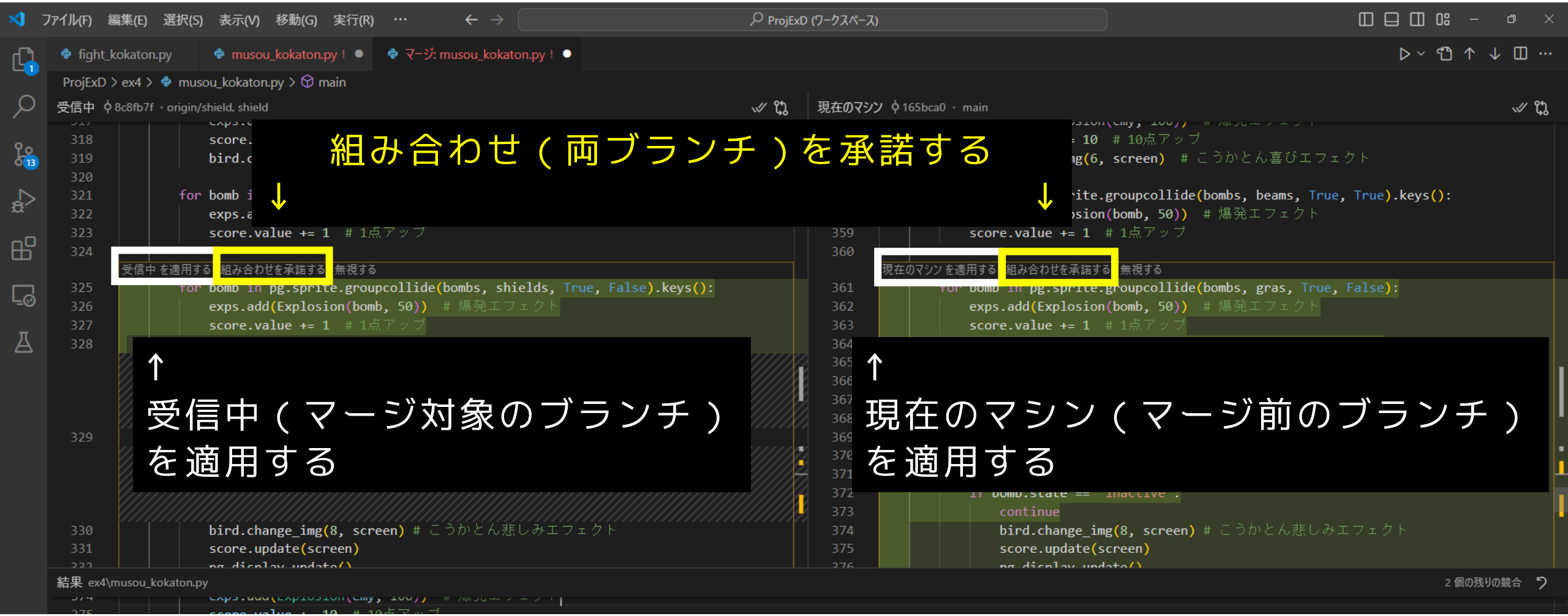


コンフリクトの手動解消方法

- コンフリクト：同じ場所に異なる編集が加わっている状態
- 解消方法：「どちらか一方だけが正しいというわけではない」ので，それぞれの編集意図を組み，相談しながら慎重にコードを**修正する**
- 必要に応じて，加筆しても構わない
- 【手順】
  1. マージエディタでコードを修正する
  2. 保存する
  3. コードが実行できることを確認する
     ※代表者と自身の実装した追加機能が反映されているか確認する
  4. ステージング，コミットする

     マージコミットと呼ばれる ↑

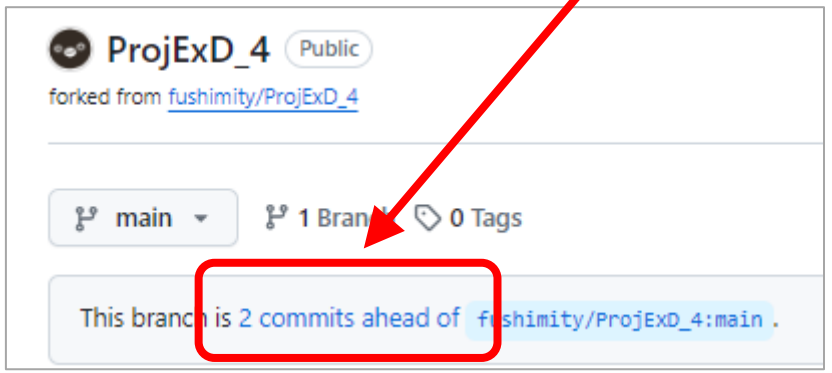
マージエディタの使い方



※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

git,GitHubでの作業⑪⑫（その他一人目）

- ⑪mainブランチをプッシュする：`git push origin main`
  - その他メンバーのリモートリポジトリは「2 commit ahead」となっている

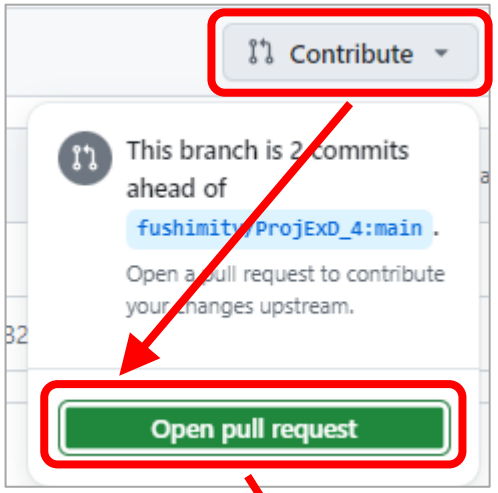


- ⑫代表者にプルリクする（次ページ参照）

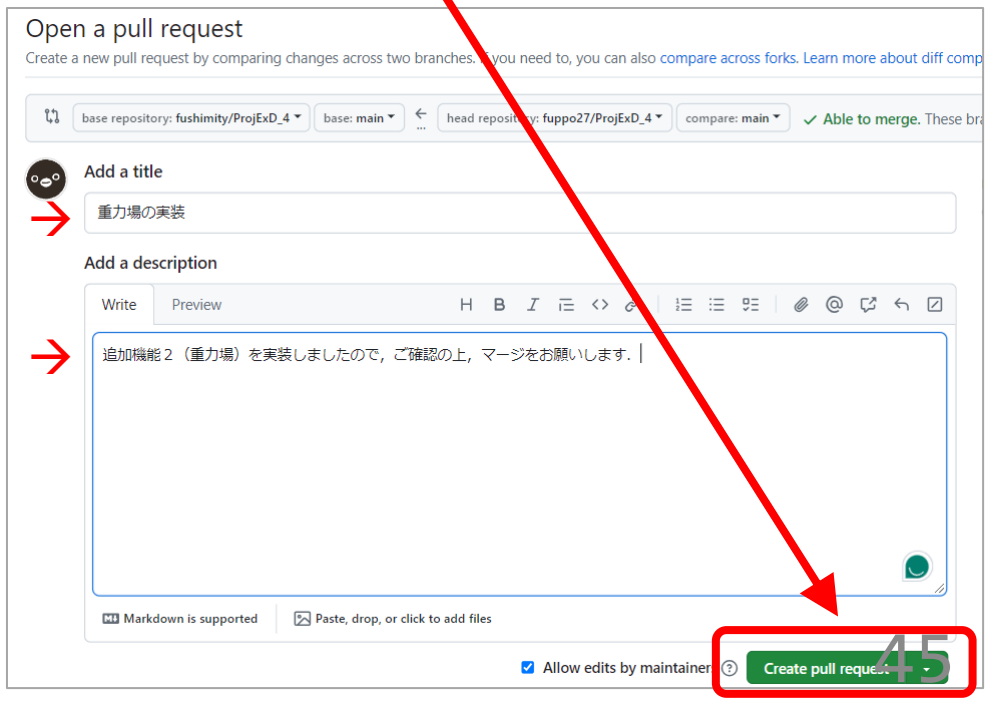
プルリクエスト（プルリク）：
自身が行った変更を反映してほしい旨をリクエストすること

プルリクの送信方法

- **自身のリモートリポジトリ**にて
  - 「Contribute▼」をクリックする
  - 「Open pull request」をクリックする

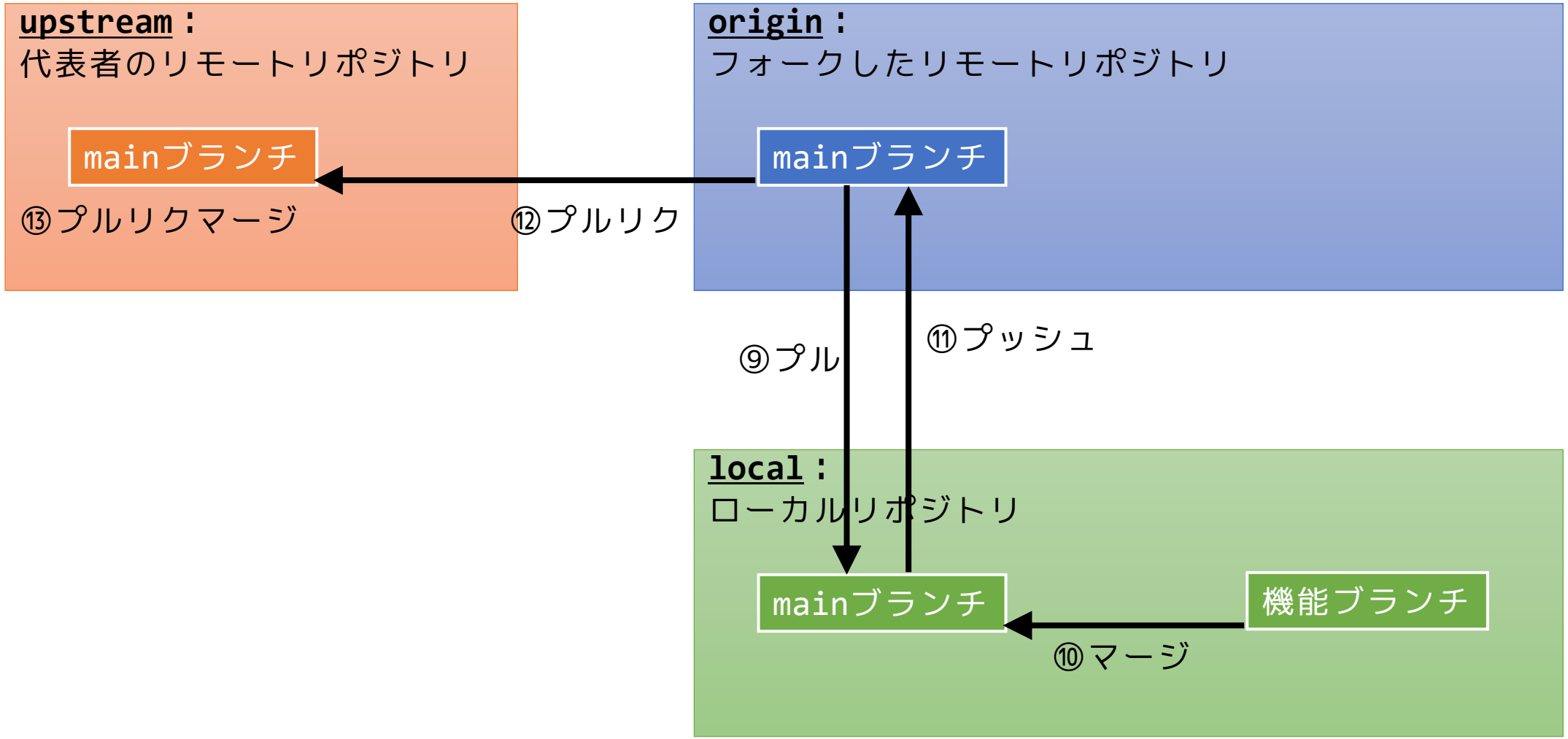


  - タイトルとメッセージを入力する
  - 「Create pull request」をクリックする



作業の流れ（イメージ図）

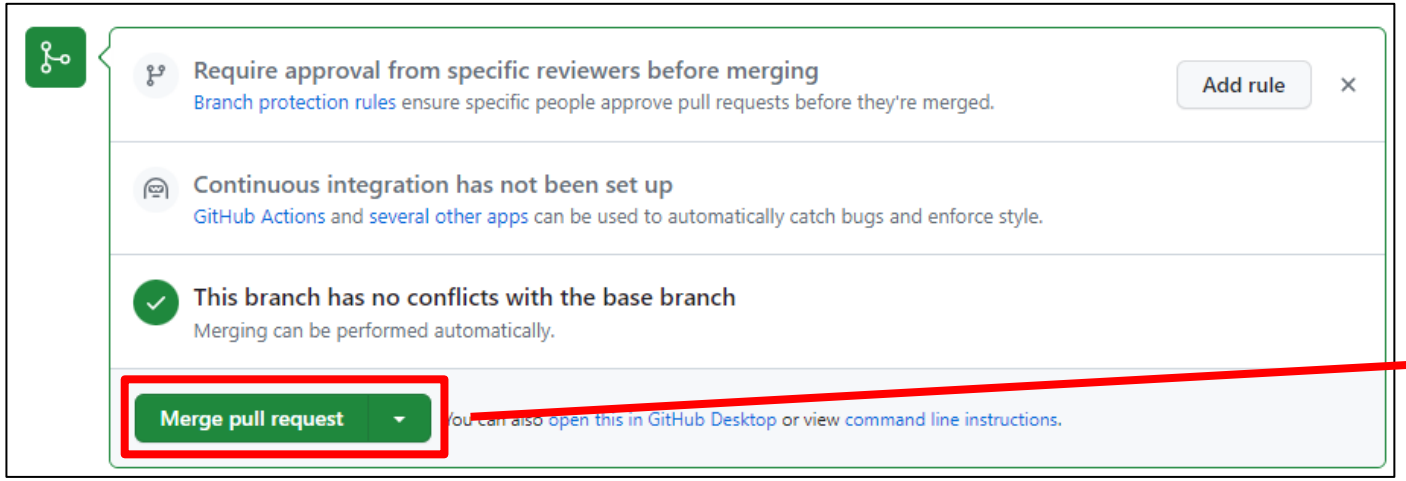
- その他メンバーの視点

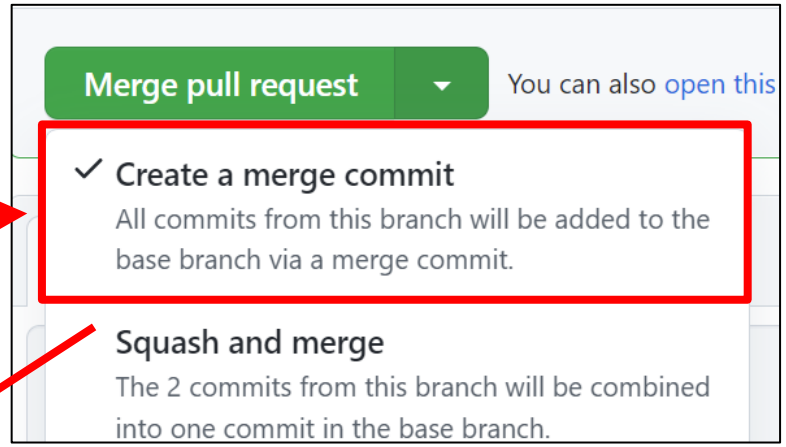


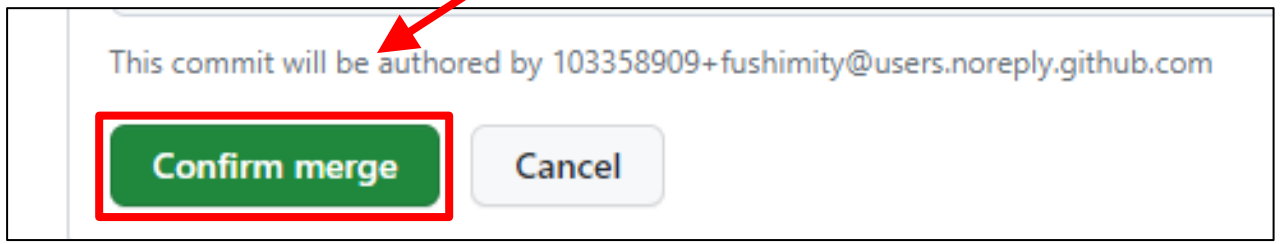
GitHubでの作業⑬（代表者）

- コードを確認後，プルリクをマージする

本当は，ローカルで実行して
バグがないことを確認する必要がある
が今回は割愛する







その他二人目以降の流れ

- P.39：GitHubでの作業⑧（その他全員）
  - ⑧Sync fork
- P.40：gitでの作業⑨⑩（その他二人目）
  - ⑨プル
  - ⑩マージ
- P.44：git,GitHubでの作業⑪⑫（その他二人目）
  - ⑪プッシュ
  - ⑫プルリク
- P.47：GitHubでの作業⑬（代表者）
  - ⑬プルリクマージ

ローカルのmainブランチ＝完成版

- 完成版の「真・こうかとん無双」で遊んでみよう
- マージされた全ての追加機能が動作するか確認しよう
- 不備が見つかったら，
  - 代表者：コード修正 → ステージング → コミット → → プッシュ `git push origin main`
  - その他メンバー：Fork Sync → プル `git pull origin main`
- バグフィックスが完了したら，提出物の準備をしよう

チェック項目

1. 複数人の機能ブランチが**マージ**されているか（人数 x 2） [0 ~ 10]
   - コミット履歴でユーザ名を確認
2. コード最終版にマージされた**機能は全て動くか**（機能数） [0 ~ 5]
   - 動いても，**思いもよらない動作をしたらアウト**
   - デモとコード最終版で確認

※上記チェック項目は減点方式で，最大5点から出席者のうち
{ **マージ**，**正しく実装** } されていない人数を減点する

3. 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：C0A23XXX_kadai4.pdf
- 内容：以下の順番でPDFを結合して提出すること
  - 代表リポジトリのコミット履歴
    https://github.com/fushimity/ProjExD_4/commits/main
  - コードの最終版
    https://github.com/fushimity/ProjExD_4/blob/main/musou_kokaton.py

代表者のアカウント名

※提出物，および，点数はメンバー全員同じになる
※代表者のみチェックを受け，チェック終了後，全員Moodleに提出
※提出物のファイル名は自身の学籍番号にすること

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

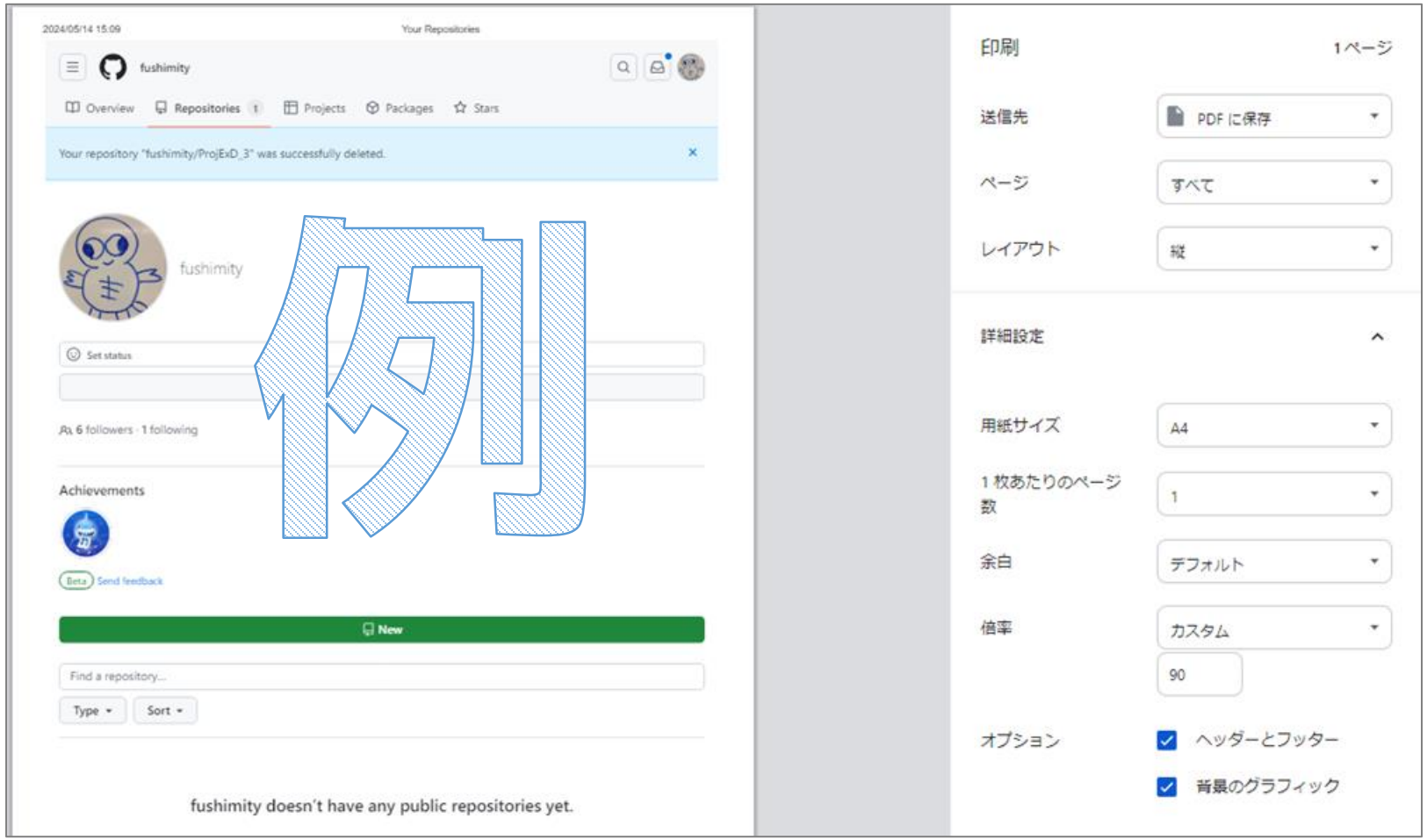
1．ChromeでPDFとして保存する（次ページを参照）

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「C0A23XXX_kadai4.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：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